

AWS IAM Identity Center (formerly AWS SSO) Master File

20-Question Master Framework for AWS IAM Identity Center (70× Depth)

1. Deep Introduction to AWS IAM Identity Center and Its Role in Access Management

A full conceptual foundation explaining what IAM Identity Center is, why AWS built it, how it replaces/extends AWS SSO, and how it fits into enterprise identity and multi-account governance.

2. Internal Architecture of IAM Identity Center

Full deep-dive into the internal components, backend systems, data flows, identity store architecture, directory integration model, SCIM sync engine internals, permission orchestration pipeline, and how Identity Center interacts with AWS Organizations.

3. Identity Sources: Built-in, AWS Managed AD, and External Identity Providers

Detailed analysis of the three identity source modes, their internal mechanics, synchronization flows, trust relationships, password/attribute ownership boundaries, and constraints.

4. Authentication & Single Sign-On Flows (SAML, OIDC, External IdPs)

Full 70× breakdown of authentication chains, token issuance, IdP redirection logic, session handling, federation cookies, role assumption flows, and AWS console + CLI authentication routes.

5. Permission Sets Deep-Dive (Core Concept, Internals, Mechanics)

Detailed modeling of permission sets, creation lifecycle, assignment mechanics, how they convert into IAM roles, propagation flows, consistency models, and scaling behavior.

6. AWS Account Assignment Pipeline and Multi-Account Access Management

How user/group → permission set → account assignments work, mapping logic, provisioning workflows, role creation behind the scenes, error conditions, throttling behaviors, and reconciliation frameworks.

7. SCIM Provisioning Internal Architecture and Flow Mechanics

Full deep explanation of SCIM inbound provisioning, attribute mapping, delete/deactivate flows, re-sync behavior, drift management, group propagation, and troubleshooting SCIM issues at scale.

8. Federation Mechanisms: SAML Internals, Assertions, Tokens, Attributes

How SAML assertions are used, role of ACS URLs, relay states, attributes, signature validation, trust setup with external IdPs, and high-depth federation security considerations.

9. Federation Mechanisms: OIDC Internals, Tokens, Lifecycles, Device Auth

Deep explanation of OIDC flows, ID tokens, access tokens, refresh tokens, cryptographic validation, device authentication, CLI/SDK OIDC flows, and mobile/desktop authentication.

10. CLI & Programmatic Access Through Identity Center

How AWS CLI v2 integrates with Identity Center, token caching, device authorization flow, identity-store-sync, role resolution, domain handoff, and multi-environment workflows.

11. Application Assignments and SSO Integration (Custom and SaaS Apps)

Mechanics of assigning SAML-based SaaS apps, attribute mapping, session duration, IdP-initiated vs SP-initiated sign-on, SAML relay-state behaviors, and application catalog internals.

12. Cross-Account & Cross-Organization Access Patterns

How Identity Center handles multi-OU, multi-account, or multi-Organization setups, isolation boundaries, consolidated access strategies, and advanced enterprise patterns.

13. Identity Center Security Best Practices, Hardening, and Zero Trust Considerations

Deep coverage of MFA, attribute-based security, conditional access integrations with third-party IdPs, session policies, permission boundary usage, and enterprise governance.

14. Auditing, Monitoring, Logging, and Observability for Identity Center

CloudTrail events, API calls, identity-store logs, SSO logs, login flows, failures, drift detection, monitoring strategies, and deep incident response visibility.

15. Troubleshooting Identity Center at Production Scale

Systematic diagnosis of SAML errors, SCIM issues, login failures, token validation problems, assignment propagation delays, and CLI authentication breakdowns.

16. Operational Excellence and Large-Scale Administration Practices

Operational models for enterprises: role lifecycle automation, CI/CD for permission sets, identity governance patterns, and automation frameworks for Identity Center.

17. Cost Structure and Optimization for Identity Center + External IdPs

Although Identity Center itself has no direct cost, this section covers indirect costs: People/Process overhead, IdP licensing, directory sync costs, large-scale maintenance, and optimization strategies.

18. Integration with AWS Services (Organizations, IAM, Directory Services, CloudTrail, CloudWatch)

Full 70× depth explanation of how Identity Center interacts with AWS Organizations, IAM role provisioning, Directory Services, event logs, monitoring services, and automation tooling.

19. Full Consolidated 70× Depth Summary (Single Deep Narrative)

A unified long-form, fully combined explanation that merges all 18 questions into one continuous mega-summary without sub-section separation.

20. Misconceptions, Pitfalls, Anti-Patterns, and Architecture Mistakes

Common misunderstandings (SAML vs OIDC, users vs identities vs subjects, permission-set drift, SCIM vs Just-in-Time, IdP boundaries), with detailed explanation of how to avoid them.

1 — Deep Introduction to AWS IAM Identity Center and Its Role in Access Management

1 — What IAM Identity Center actually is in plain but deep terms

AWS IAM Identity Center is the central “who-can-access-what” hub for people (human identities) across all your AWS accounts and many of your business applications.

- At the highest level, we can think of it as an orchestration layer that sits on top of AWS Organizations and IAM, and it connects three big worlds together: the world of identities (users and groups in a directory or IdP), the world of AWS accounts/resources (Organization, accounts, roles), and the world of applications (AWS console, CLI, SDKs, and SAML/OIDC apps).
 - Historically, AWS had a service called AWS Single Sign-On (AWS SSO). IAM Identity Center is the evolution and replacement of that service. Conceptually, the “core idea” remains the same: give users one place to log in, and from there they can access multiple AWS accounts and applications with appropriate permissions. But the new branding emphasizes that it is part of the broader IAM family and is more of a central identity and access management control plane rather than “just SSO.”
 - The key mental model is: IAM Identity Center does not replace IAM; instead, it orchestrates IAM roles and trust relationships on your behalf. It centralizes access in a human-friendly way while still relying on IAM roles, policies, and AWS Organizations under the hood.
-

2 — The problem space: why IAM Identity Center exists

Before IAM Identity Center (and AWS SSO), organizations typically had a messy mix of patterns: IAM users created directly in each AWS account, standalone federated roles per account with complex SAML configurations, and manual role management per account.

- This led to three core problems:
 1. Identity sprawl: the same person could exist as multiple IAM users or role mappings across accounts, making lifecycle management (joiner/mover/leaver) fragile and error-prone.
 2. Permission inconsistency: every account might implement roles with different naming conventions, different policies, and ad hoc trust relationships, making it almost impossible to guarantee consistent least-privilege access.
 3. Poor admin and user experience: admins had to touch every account to create roles, and users often had multiple login URLs, tokens, and credential sets.
- IAM Identity Center exists to centralize and standardize all of that. It gives us: a single login portal for users, a consistent pattern for permissions (permission sets), a single place to manage assignments across accounts, and a clean integration with an external identity provider (IdP) if we already have one (like Azure AD, Okta, etc.).

- From an enterprise governance point of view, IAM Identity Center is a major building block of “multi-account done properly.” It turns the multi-account model from a sprawl of local decisions into a centrally controlled access fabric.
-

3 — Where IAM Identity Center sits in the AWS security and identity stack

We can imagine AWS identity and access in layers: at the bottom we have IAM (roles, policies, permission boundaries), then AWS Organizations for account structure and SCPs, and on top of that we place IAM Identity Center as the human-access coordination service.

- IAM (the service) is still the ultimate enforcement engine: every API call is authorized by IAM evaluation. IAM Identity Center doesn’t bypass that; instead, it arranges IAM resources (mainly roles and policies) in a structured way. It is a control plane that instructs AWS to create and configure IAM roles in each account, with specific trust policies and permission policies derived from “permission sets.”
- AWS Organizations defines which accounts exist, how they are grouped (OUs), and what organization-wide guardrails (SCPs) apply. IAM Identity Center leverages this organization structure to know “which accounts exist” and “to which accounts we can assign access.” But it does not replace Organizations; it sits on top of it, consuming its configuration.
- On the identity side, IAM Identity Center either manages its own identity store (built-in users/groups) or connects to an external identity source (like an external IdP or AWS Managed Microsoft AD). Those identities are then used to drive assignment and access.
- So, the position is:

Identities (IdP/Identity Store) → IAM Identity Center (assignments, permission sets, SSO) → IAM roles in accounts → actual resource access.

4 — High-level mental model of core building blocks

To understand Identity Center deeply, we should fix a few core objects in our mind: identity source, users/groups, permission sets, AWS accounts, assignments, and applications.

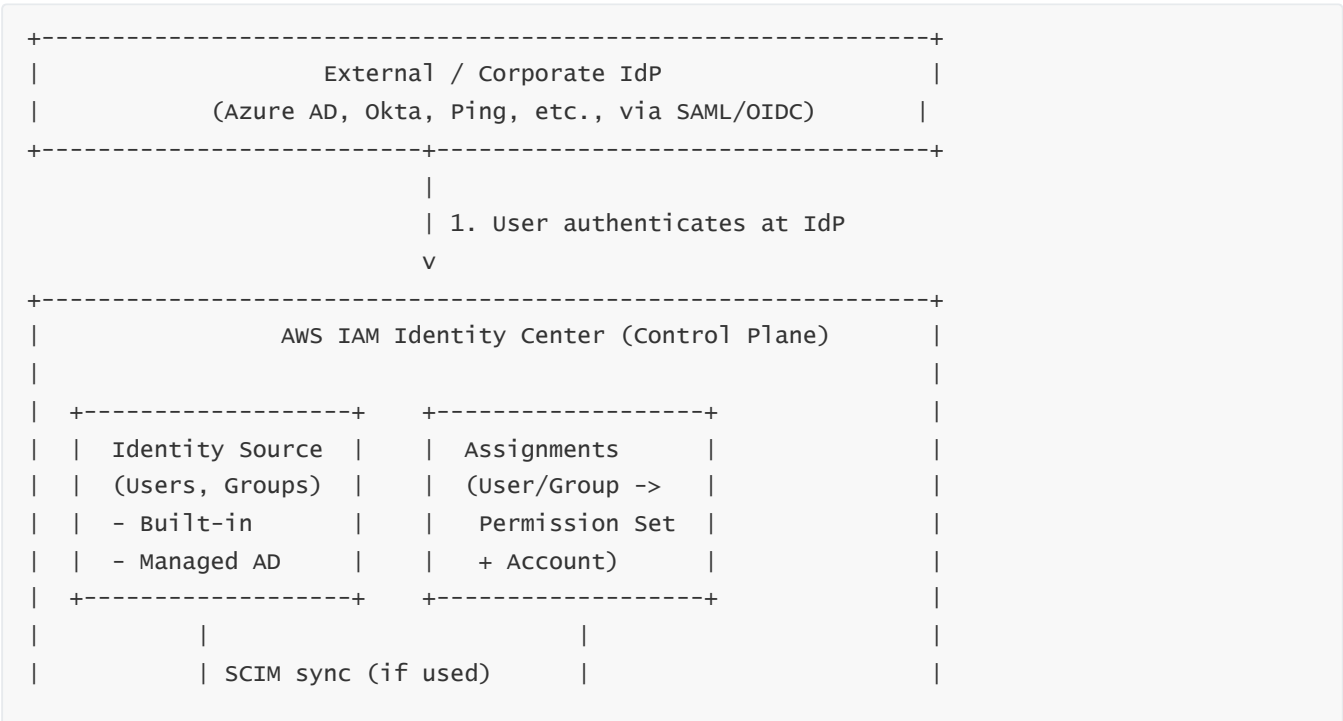
- Identity source: this is where primary records of users and groups live. It can be the built-in Identity Center directory, AWS Managed Microsoft AD, or an external IdP via SCIM and federation. The identity source is the “source of truth” for who exists and which groups they belong to.
 - Users and groups: these are human identities aggregated from the identity source. Identity Center keeps a representation of them in its own identity store (even when sourced externally) so it can attach assignments. Groups are crucial because permission schemes are often built around groups rather than individuals.
 - Permission sets: these are high-level templates that define the exact IAM permissions a user will have when they access a particular AWS account via Identity Center. A permission set isn’t itself an IAM role; rather, Identity Center uses it to create and maintain IAM roles in each target account.
 - Assignments: the “who gets what where” mapping. An assignment links a user or group, plus a permission set, to one or more AWS accounts. This is the central configuration that determines access.
 - Applications: these represent AWS accounts (console access), AWS CLI/SDK access, and external SAML/OIDC apps that users can launch from the Identity Center portal using SSO.
-

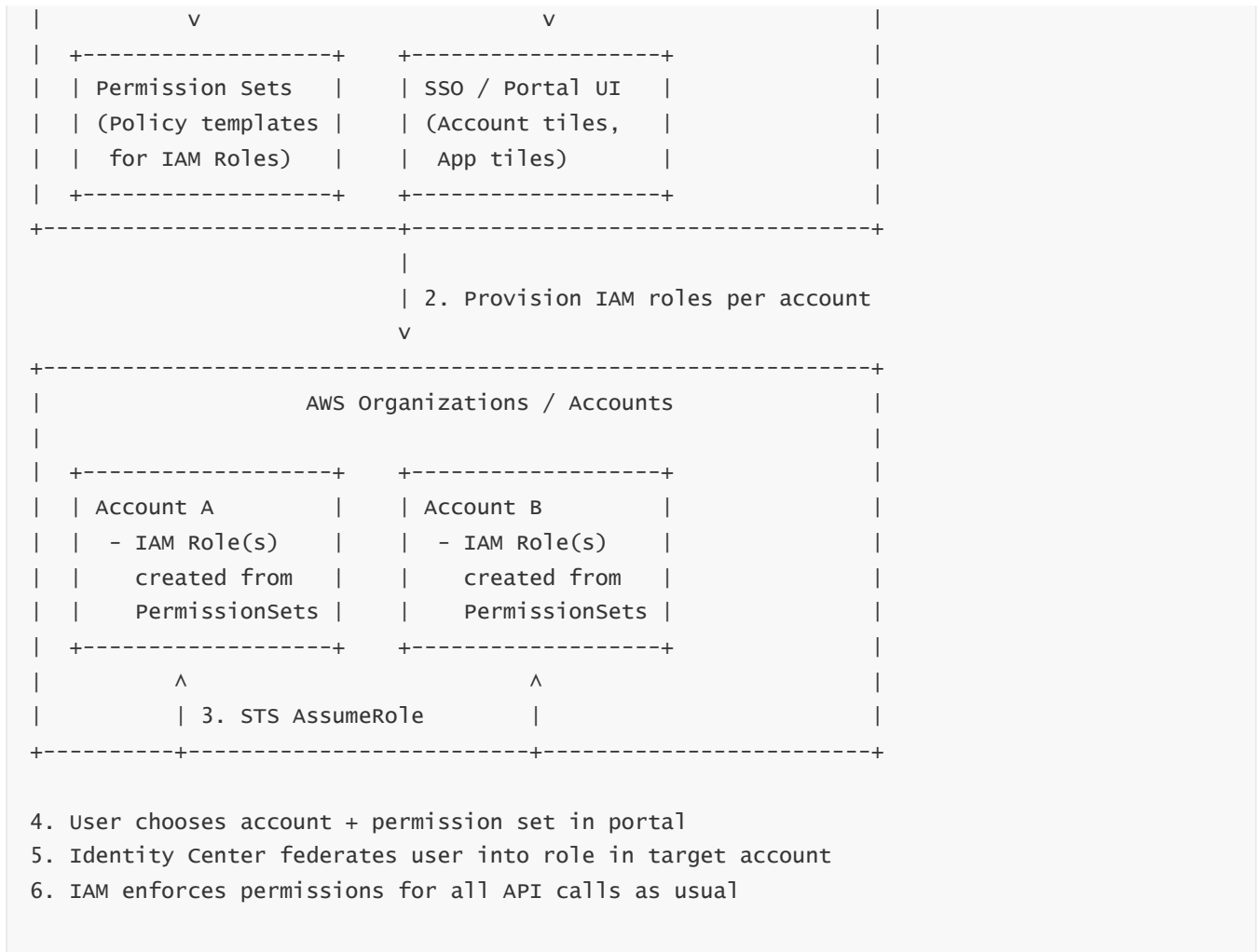
5 — A simple but deep end-to-end flow: user login and console access

Let’s walk through a conceptually detailed scenario: a user wants to log into the AWS Management Console for an account using IAM Identity Center.

- Step 1: The user opens the IAM Identity Center user portal URL. If an external IdP is configured, the user may first authenticate at the IdP using SAML/OIDC; if the built-in identity source is used, they log in directly with credentials managed by Identity Center. After successful authentication, IAM Identity Center knows “who” this is and which groups they belong to.
- Step 2: Identity Center looks up the user’s assignments. It evaluates which AWS accounts and which permission sets are assigned to that user (either directly or via groups). It also takes into account any account or OU-level constraints from AWS Organizations (for example, whether that account still exists and is in scope).
- Step 3: The user sees a portal page listing “tiles” for each AWS account and application they’re allowed to access. For AWS accounts, each tile may show multiple permission sets (for example, “AdminAccess”, “PowerUser”, “ReadOnly”).
- Step 4: When the user clicks an account tile with a particular permission set, Identity Center uses its managed IAM role in that account. Behind the scenes, for that account and that permission set, an IAM role already exists, created and maintained by Identity Center, with a trust policy that allows Identity Center to issue a role-assumption on behalf of that user.
- Step 5: Identity Center then performs a secure, temporary federation into that IAM role for the user, generating temporary credentials (STS) bound to that role’s permissions. The user is then redirected to the AWS Management Console for that account with that effective role. The entire process is transparent to the user; they just see that they clicked a tile and landed in the console.
- Step 6: When the user later logs out or the session expires, the temporary credentials are invalidated, and the trust chain is enforced by IAM as usual. Identity Center never bypasses IAM; it only orchestrates the STS flow based on centralized assignments.

6 — ASCII architecture diagram: where Identity Center fits





– In this diagram, the “External / Corporate IdP” represents any external directory where users authenticate. The connection to Identity Center is typically SAML or OIDC for authentication plus SCIM for provisioning users and groups.

– The central “AWS IAM Identity Center (Control Plane)” block shows the main conceptual subsystems: identity source mirror, assignments, permission sets, and the SSO portal. This is the brain that calculates who can see which tiles and which roles to assume.

– “AWS Organizations / Accounts” shows that Identity Center doesn’t store resources itself; instead, it configures IAM roles in each AWS account and then uses STS to federate users into those roles. Permissions are still enforced in each account by IAM. Identity Center is orchestrating and federating, not directly authorizing every API call.

7 — Relationship with the old “AWS SSO” and why the rename matters

Originally, the service was called “AWS Single Sign-On (AWS SSO).” The new name, IAM Identity Center, reflects that the service is now structurally part of the IAM family and should be viewed as the central human identity access layer for AWS rather than “one more SSO product.”

– Conceptually, AWS SSO focused in people’s minds on the login experience. IAM Identity Center shifts the narrative toward end-to-end identity lifecycle, multi-account access governance, and integration with IAM and Organizations.

- Feature-wise, Identity Center inherits and builds on all the SSO capabilities but also improves its fit with modern identity patterns like external OIDC, better SCIM provisioning, more flexible permission set management, and tighter integration with AWS IAM.
 - From a design and exam perspective, when we see “AWS SSO” in old docs or diagrams, we should mentally map it to IAM Identity Center and ask ourselves: “Where is the identity source? How are permission sets defined? How is the multi-account mapping done?” That mindset is what makes it easy to upgrade understanding from AWS SSO to Identity Center.
-

8 — Control plane vs data plane mental model for Identity Center

A very useful deep mental model is to think of IAM Identity Center almost entirely as a control plane service.

- Control plane: where we configure identities, groups, permission sets, and account assignments. These configurations cause AWS to create and maintain IAM roles in target accounts. All of this is done via Identity Center APIs and console, and it’s mostly metadata and IAM role provisioning.
 - Data plane (actual authorization): when a user makes API calls or uses the console, IAM in the target account evaluates policies, SCPs, and other controls to decide allow/deny. Identity Center isn’t sitting in the middle of every call; it only influenced which role the user assumed and what policies are attached to that role.
 - This separation is important for troubleshooting and security reasoning. If a user can’t perform some action, we ultimately debug IAM policies in that account (plus SCPs), not Identity Center’s runtime. If a user isn’t seeing an account tile or can’t assume a role at all, we debug Identity Center’s control plane (assignments, permission sets, identity mapping, SCIM data).
-

9 — How IAM Identity Center changes access management workflows for admins

From the admin perspective, the biggest change introduced by Identity Center is moving from account-local role management to centralized, template-based role management.

- Instead of going into each account to create roles, attach policies, and set up SAML trusts with an external IdP, admins now define permission sets one time at the Identity Center level. These permission sets encapsulate the IAM permissions and session settings that should be created as roles in each account.
 - Admins then attach these permission sets to users/groups and accounts as assignments. Identity Center automatically handles the creation and maintenance of the corresponding IAM roles in each account. If we change the permission set, Identity Center can propagate that change out to all the roles derived from it.
 - In effect, access management becomes “identity central”:
 1. Manage identities (mostly) where they already live (corporate IdP or directory).
 2. Manage permissions centrally as permission sets.
 3. Manage which identity gets which permission set in which accounts from one console or API.
 - This drastically reduces operational friction and makes multi-account governance more realistic and robust, especially in large environments with tens or hundreds of accounts.
-

10 — How IAM Identity Center changes the experience for end users

For end users, the presence of IAM Identity Center means “one portal, one login, many accounts and apps.”

- Instead of memorizing multiple accounts' console URLs or using different credential sets, users authenticate once (usually leveraging corporate SSO) and then land on a single portal page that shows all the AWS accounts and applications they are entitled to.
 - From there, they simply click the appropriate account and permission set tile. They don't have to understand roles, trust policies, or STS; all of that is abstracted away by Identity Center. They just know "I am a Developer in Account A" or "I am a ReadOnly user in Shared Services account" based on the tile labels.
 - For programmatic access (AWS CLI v2, SDKs), Identity Center provides integration so that users can obtain temporary credentials via an SSO-style workflow without dealing with static access keys. This improves security (no long-lived keys) and user experience (centralized login, role selection, profiles).
-

11 — Big-picture view: IAM Identity Center as the human edge of AWS IAM

When we put all of this together, the simplest deep description is: IAM Identity Center is the human edge of AWS IAM in a multi-account world.

- IAM remains the engine that evaluates permissions for every API call. Organizations remains the engine that defines global account boundaries and guardrails. Identity Center is the layer that binds human identities to specific IAM roles in specific accounts using a centralized, policy-driven, SSO-friendly model.
 - It leverages modern identity standards (SAML, OIDC, SCIM) so that we can plug in the enterprise's existing identity infrastructure. It leverages AWS-native constructs (IAM roles, policies, Organizations) so that access is enforced the same way as any other IAM-based access.
 - This is why modern AWS multi-account best practices almost always mention IAM Identity Center: it is the recommended way to manage human access at scale, especially when we have an existing corporate IdP and want least-privilege, short-lived, centrally governed access to dozens or hundreds of accounts.
-

2 — Internal Architecture of AWS IAM Identity Center

1 — Instance model, regional scope, and relationship to AWS Organizations

At the deepest architectural level, IAM Identity Center is an *instance-based* control plane service. When we "enable" Identity Center, AWS creates an *Identity Center instance* in a specific Region. That instance becomes the authoritative control plane for all its configuration: identity source, users and groups mirror, permission sets, assignments, and app configurations. Architecturally there are **two instance types**:

- **Organization instance** – created in the *management account* of AWS Organizations. This is the recommended and primary architecture because only the organization instance can manage user access to AWS accounts across the Organization. It integrates directly with AWS Organizations to discover accounts and to provision IAM roles into those accounts. ([AWS Documentation](#))
- **Account instance** – a local, per-account instance that can be used only for application access (no cross-account AWS account management). It still has similar internal components (identity source, assignments, apps) but does not integrate with Organizations for multi-account role provisioning. ([AWS Documentation](#))

– Architecturally, *one* Identity Center organization instance exists per Organization (per Region), and it has a **single Region scope**. This means its directory and configuration are Region-bound. That is why in AWS security reference architecture guidance, DR strategies often require planning for a different identity source or recovery pattern if the Region is impacted, because the Identity Center directory (if using internal directory) is regional. ([AWS Documentation](#))

2 — High-level internal subsystems inside an Identity Center instance

Inside that instance, we can think of Identity Center as composed of several logical subsystems, each responsible for a different part of the lifecycle:

– **Identity Store / Directory Mirror** – stores logical representations of users and groups known to Identity Center (either native users managed directly in Identity Center or objects synced from an external IdP or Active Directory). Even when an external IdP is used, Identity Center maintains a “shadow” or mirror record for each relevant user and group so it can attach assignments. ([AWS Documentation](#))

– **Identity Source Integration Layer** – manages the connection to the configured identity source (Identity Center directory, AWS Managed Microsoft AD, or external IdP via SAML + SCIM). This layer owns the SCIM endpoints, SAML/OIDC metadata, and any AD connector/trust configuration. ([AWS Documentation](#))

– **Permission Set Engine** – stores and processes permission sets (templates of IAM policies and session settings). When assignments or configuration changes occur, this engine translates them into IAM roles and policies in each target account. ([Frontegg](#))

– **Account & Application Assignment Store** – the persistent mapping of (user or group) → (permission set or application configuration) → (AWS account or app). This is effectively the central access graph from which Identity Center derives which tiles appear in the portal, and which IAM roles to create in which accounts.

– **Account Provisioning Engine (Role Provisioner)** – responsible for creating, updating, and deleting IAM roles in AWS accounts based on permission sets and assignments. This component integrates with AWS Organizations to enumerate accounts and uses IAM APIs in each account to manage roles and policies.

– **SSO / Federation & Session Service** – handles the actual SSO experience: authenticating via the identity source, issuing and managing sessions, performing STS AssumeRole calls into target accounts, issuing OIDC tokens for CLI/SDK, and generating SAML/OIDC assertions for integrated apps. ([AWS Documentation](#))

– **User Portal / Access Portal** – the web UI that surfaces the assignment results to the end user: account tiles, permission set options, and application tiles. Internally, this portal queries the assignment store and uses the SSO service to perform federations when tiles are clicked.

– **Logging & Audit Integration** – hooks into CloudTrail and related event sources to emit management-plane events for Identity Center APIs and provisioning operations, which are then consumed by CloudTrail, CloudWatch, and other monitoring tools. ([AWS Documentation](#))

These subsystems together form a control plane that *describes* and *orchestrates* access, while IAM and STS in each account remain the ultimate enforcement/data-plane engines.

3 — Identity source integration architecture (built-in, AD, external IdP)

Architecturally, the “identity source” is a pluggable component in front of the Identity Store:

– **Built-in IAM Identity Center directory** – Identity Center itself runs a managed directory. All user objects, group objects, and credentials (password hashes, MFA settings, etc.) are stored in this managed directory. The Identity Store reads from this internal directory directly. This is the simplest architecture and is often used in small or isolated environments or labs. ([AWS Documentation](#))

– **AWS Managed Microsoft AD / On-prem AD** – In this mode, Identity Center integrates with AWS Directory Service. Either an AWS Managed Microsoft AD is connected directly, or a self-managed AD is connected via AD Connector or trust. Identity Center then synchronizes users and groups from that directory into its Identity Store. Authentication may be routed to AD via Kerberos/LDAP and trusts, while Identity Center maintains only identifiers and group memberships needed for authorization. ([AWS Documentation](#))

– **External IdP via SAML + SCIM (Okta, Entra ID, etc.)** – Here the architecture splits:

– *Authentication path*: Users authenticate at the external IdP (Azure AD / Microsoft Entra ID, Okta, Ping, etc.) using SAML or OIDC. Identity Center trusts the IdP as a SAML/OIDC IdP, validating assertions and tokens. ([AWS Documentation](#))

– *Provisioning path*: Users and groups are *provisioned* into Identity Center using SCIM v2.0. The SCIM engine inside Identity Center exposes endpoints that the IdP uses to create, update, and deprovision users/groups. Identity Center keeps these in sync in its Identity Store. ([AWS Documentation](#))

– Architecturally, this means the Identity Store is the central representation of identities that Identity Center uses for assignments, but the *identity source integration layer* determines *how* those objects are populated and which system is the source of truth for attributes and group membership.

4 — Permission Set Engine and IAM role provisioning in accounts

Internally, permission sets are not IAM roles themselves; they are templates that specify:

- Which IAM policies to attach (AWS managed, customer managed, or inline JSON policy).
- Session duration, relay state, and other session-related settings for console access.
- Possibly permission boundary or tags that should apply to the resulting roles (depending on configuration). ([Frontegg](#))

Architecturally, when we create or update a permission set:

- The **Permission Set Engine** stores the template in Identity Center's configuration store. It does not immediately create roles in all accounts; the actual role creation is triggered when assignments link that permission set to specific accounts.
- When an assignment is created (or when either the permission set or account state changes), the **Account Provisioning Engine** runs. It:
 - Enumerates target accounts via AWS Organizations.
 - In each account, creates or updates an IAM role whose name and trust policy are controlled by Identity Center.
 - Attaches the policies defined in the permission set to that role.
 - Maintains a mapping between the internal permission set ID and the IAM role ARN in each account.

– This engine is asynchronous and eventually consistent: changes to permission sets or assignments may take some time to fully propagate to all accounts, which is important when we reason about architecture and troubleshooting. It uses IAM APIs in each account and respects rate limits, so it may batch or retry operations under the hood.

5 — Assignment computation and access graph resolution

The **Assignment Store** is essentially a relational graph:

- Nodes: Users, Groups, Permission Sets, Accounts, Applications.
- Edges: “User is member of Group”; “Group has Permission Set X in Account Y”; “User directly has Permission Set Z in Account W”; “Group is assigned to App A”, and so on.

Architecturally, whenever a user session is established (for portal or CLI), the **Assignment Resolver** performs:

1. **Identity Expansion** – Resolves the user’s identity from the Identity Store and expands group memberships (based on the identity source state synced via SCIM or AD integration).
2. **Assignment Lookup** – Queries the assignment store for any mappings involving that user or any groups they belong to.
3. **Policy & Role Resolution** – For each relevant assignment, it locates the corresponding IAM role ARN in each target AWS account (that ARN is maintained by the Permission Set Engine / Account Provisioning Engine).
4. **Portal Tile Construction** – For the portal, it compiles a list of “resources” (AWS accounts + permission set labels, applications) and returns them to the portal UI layer. For CLI/SDK, it gives the CLI profile layer enough information to know which roles/profiles the user can use.

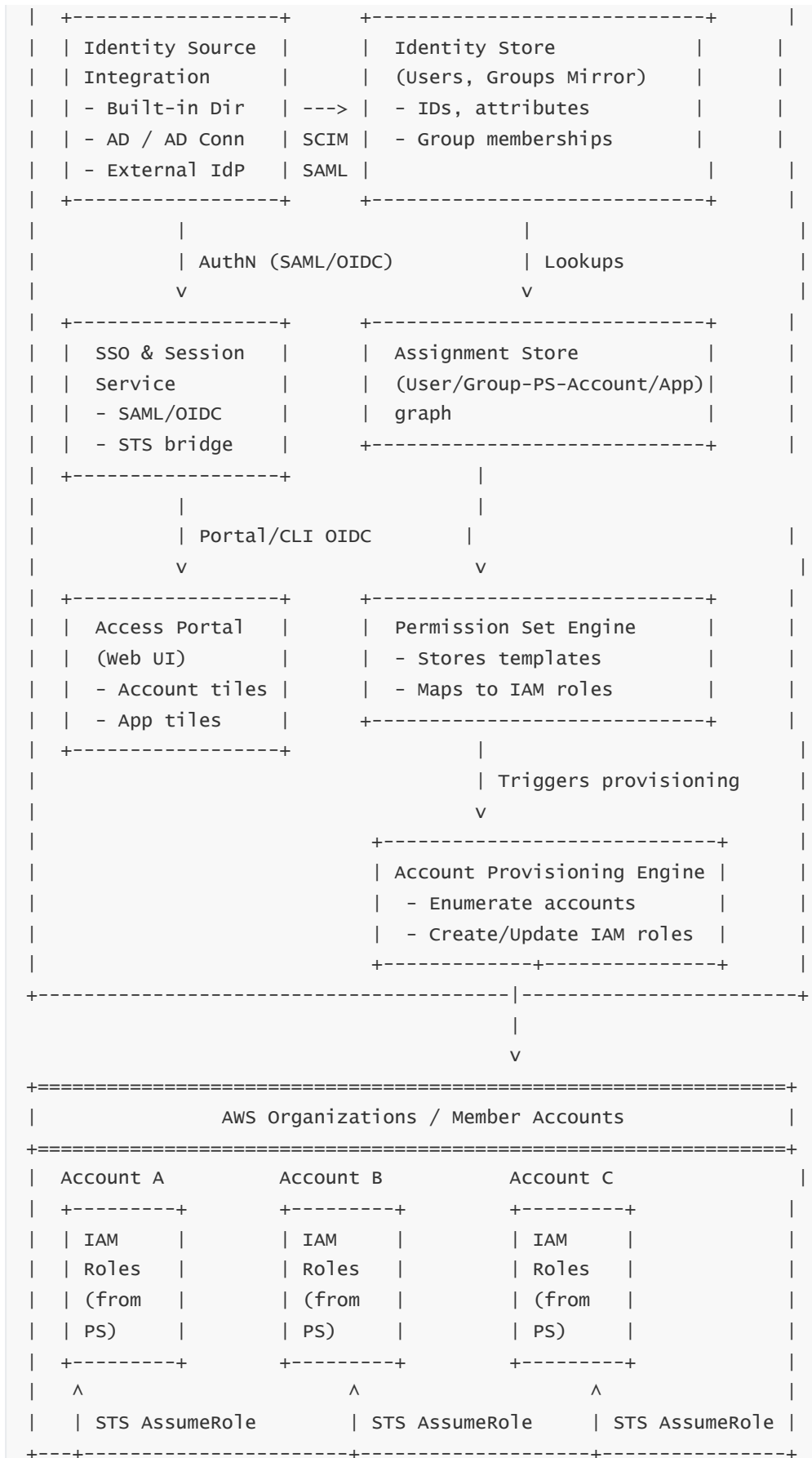
From an internal perspective, this is a heavily read-optimized workflow: the assignment graph must be quickly queryable based on a user’s identity and group memberships, which is why Identity Center keeps its own Identity Store and assignments rather than querying external IdPs on every request.

6 — SSO, STS, and token services as the federation core

The **SSO / Federation Service** is the runtime component that actually hands out sessions and credentials. Its architecture is tightly integrated with:

- **STS** (AWS Security Token Service) – used to assume IAM roles in target accounts on behalf of the authenticated user. Identity Center’s managed IAM roles trust the Identity Center service principal, which allows Identity Center to call `AssumeRole` with an appropriate session name and attributes.
- **SAML/OIDC endpoints** – used to authenticate via external IdPs and to issue SAML assertions or OIDC tokens to apps or the AWS console. The service validates incoming SAML assertions from IdPs and issues outbound SAML assertions when it is acting as an IdP for AWS apps or other integrated services. ([AWS Documentation](#))
- **Session and token store** – manages Identity Center sessions, including session duration and token lifecycles for portal access and for CLI OIDC flows. The AWS CLI v2 uses a device authorization flow (OIDC) where the user completes the browser login and Identity Center issues OIDC tokens that CLI uses to exchange for STS credentials.

Architecturally, this service sits between identity source authentication and AWS account role assumption, bridging the identity world and the role world with a coherent token lifecycle management engine.



- The **top half** shows the Identity Center instance as a cluster of internal components connected by identity and configuration flows: identity source integration feeding the Identity Store, which feeds assignments and the SSO/session service, which then interacts with permission sets and provisioning.

- The **bottom half** shows that these control-plane decisions materialize as IAM roles in each AWS account, where IAM and STS are responsible for actual authorization of API calls.
 - Architecturally, this makes clear that Identity Center’s instance is the “brain” while accounts are the “muscles”: the brain decides who should have which role, and the muscles (IAM/STS) enforce it.
-

10 — Resilience, limitations, and architectural implications

Finally, we should understand a few important architectural characteristics and their implications:

- **Single-region directory scope** – When using the built-in Identity Center directory as the identity source, that directory lives in the Region where the instance is enabled. In the unlikely event that this Region is severely impacted, the directory (and thus user access governance) is also impacted. AWS guidance for high resilience therefore often recommends using an external IdP / AD as the identity source, so critical identity data is not bound to a single AWS Region. ([AWS Documentation](#))
 - **Dependency on AWS Organizations for account management** – The organization instance relies on AWS Organizations to discover and manage member accounts. If organization structures change (accounts moved between OUs, created, or closed), the Account Provisioning Engine and assignment logic must respond accordingly.
 - **Eventual consistency in role provisioning** – Because IAM role creation and updates are performed asynchronously per account, there is an inherent propagation delay. Architecturally, we must design operations and automation under the assumption that new assignments or permission set changes may take some time before every target account reflects them.
 - **Security boundary clarity** – Identity Center is *not* an additional enforcement engine; the real security boundary remains IAM in each account. Thus, designing SCPs, IAM policies, and permission boundaries remains critical. Identity Center centralizes administrative intent, but security posture is still determined by IAM evaluation combined with Organizations guardrails.
-

3 — Identity Sources: Built-in, AWS Managed AD, and External Identity Providers for IAM Identity Center

1 — Overview of the three identity-source options

When you enable AWS IAM Identity Center (formerly AWS SSO) in your AWS management account, one of your first configuration steps is to choose an *identity source*. This choice determines **where users and groups are managed**, where authentication takes place, and which directory or IdP (identity provider) is the source of truth for your workforce. You have **only one identity source per organization instance**. ([AWS Documentation](#))

The three supported options are:

- **Built-in Identity Center directory** – the default directory when you first enable IAM Identity Center. You create and manage users/groups within Identity Center itself. ([AWS Documentation](#))
- **AWS Managed Microsoft AD (or connected on-premises AD)** – you integrate an Active Directory (AD) directory hosted in AWS (via AWS Directory Service) or trust with your on-premises AD, and Identity Center uses that AD as your identity source. ([AWS Documentation](#))
- **External Identity Provider via SAML + SCIM** – you use your corporate IdP (e.g., Okta, Microsoft Entra

ID / Azure AD, Ping Identity, etc.) and configure it via SAML for authentication and SCIM for provisioning. ([AWS Documentation](#))

Each of these has different trade-offs in terms of complexity, control, user lifecycle management, and integration with your existing identity infrastructure.

2 — Built-in Identity Center directory: mechanics, benefits & limitations

Mechanics

When you select the built-in Identity Center directory, you effectively use a managed directory hosted by AWS inside the region where the Identity Center instance lives. You go to the Identity Center console, enable it, and by default this directory is chosen unless you change it. ([AWS Documentation](#)) You then create users, groups, and memberships directly in Identity Center (or via the IdentityStore APIs). Authentication (passwords, MFA devices, etc.) is managed by Identity Center.

Benefits

This is the simplest setup: no external AD, no connectors, no external IdP. It's ideal for small teams, labs, or green-field deployments where you don't already have a corporate directory you must integrate. You get fast access, minimal dependencies, and you can start assigning permission sets to AWS accounts quickly.

Limitations / Architectural implications

- Because the directory is managed within the Identity Center instance (which is regional), you get a **single-region directory scope**, which has implications for disaster recovery or multi-region resilience.
- If you already have a corporate directory with thousands of users and complex group structures, using the built-in directory may require duplication of user lifecycle tasks (onboarding/offboarding) or separate identity workflows.
- You'll need to manage user authentication (password policies, MFA) separately, which might duplicate efforts if you already have a corporate IdP.
- If later you decide to switch identity sources (for example from built-in to external IdP), AWS documents indicate that the change may require recreating or migrating many users, groups and assignments. ([Amazon Web Services, Inc.](#))

In summary: built-in directory is simple and fast, but less integrated and less scalable in large enterprise scenarios.

3 — Active Directory integration (AWS Managed Microsoft AD or on-premises AD)

Mechanics

When you integrate AD, you use the AWS Directory Service (for example AWS Managed Microsoft AD) or AD Connector/trust to your on-premises AD. ([AWS Documentation](#)) You configure your directory, establish the trust or connector, and then configure Identity Center to use that directory as its identity source. Identity Center will then mirror users/groups (not credentials) into its Identity Store for assignment logic, while authentication continues via your AD. For example, if using AWS Managed AD, Identity Center “uses the

connection provided by AWS Directory Service to perform pass-through authentication to the source AD instance.” ([AWS Documentation](#))

Benefits

- Leverages existing investment in AD: users, groups, credentials (passwords, MFA) remain in your corporate directory; no need to duplicate.
- Group membership from AD can drive assignments in Identity Center (once synced). This helps with centralised lifecycle and enterprise governance.
- Better aligned with enterprise identity lifecycle (joiner/mover/leaver) when AD is already central.

Architectural and operational considerations

- The directory must reside in the management account or delegated admin account for Identity Center. ([AWS Documentation](#))
- You must ensure that the directory is available in the same Region as Identity Center if using AWS Managed AD. The directory and Identity Center instance are region-bound together. ([AWS Documentation](#))
- Because authentication passes through AD, latency, connectivity, and availability of AD/contact path must be considered (especially in disaster or network partition scenarios).
- If you later move to an external IdP, or change the identity source, assignments may break or need remapping. It’s more complex to transition. ([Amazon Web Services, Inc.](#))

In summary: using AD integration is ideal for enterprises already using AD, but introduces dependencies and regional availability constraints.

4 — External Identity Provider (SAML + SCIM) integration

Mechanics

In this mode, you use your corporate IdP (for example Okta, Azure AD/Entra, OneLogin, Ping) as the identity source. Authentication is done at the IdP via SAML/OIDC, and provisioning (users and groups) is done via SCIM into the Identity Center store. ([AWS Documentation](#))

From the AWS documentation: “With IAM Identity Center, you can connect your existing workforce identities from external identity providers (IdPs) through the Security Assertion Markup Language (SAML) 2.0 and System for Cross-Domain Identity Management (SCIM) protocols.” ([AWS Documentation](#))

Benefits

- You retain a single corporate IdP to manage all authentication and identities (no separate directory technology inside AWS).
- User lifecycle, group membership, MFA, conditional access etc remain in your existing IdP. This supports “identity as a service” model and good enterprise identity governance.
- Enables centralised identity across clouds, hybrid, and SaaS applications if your enterprise already uses the IdP for many services.

Architectural/operational considerations

- Even when using an external IdP, you must provision users/groups into Identity Center (via SCIM or manual) so that they can receive assignments. The IdP alone cannot “drive” assignments without those representations. ([AWS Documentation](#))
- When using external IdP, authentication delays, metadata updates (SAML exchange), certificate rotations, and directory changes must be managed carefully. The path includes trust relationships and federation flows, which adds complexity.
- Changing identity sources (for example back to AD or built-in) is possible but can lead to deletion of users/groups/assignments unless carefully handled. ([Amazon Web Services, Inc.](#))
- The Identity Center still retains a “shadow” representation of users and groups in its Identity Store. So even if the IdP is external, Identity Center’s internal assignment store uses those mirrored identities to manage assignments.

In summary: external IdP mode offers strong enterprise integration and scalability, but introduces federation and provisioning complexity.

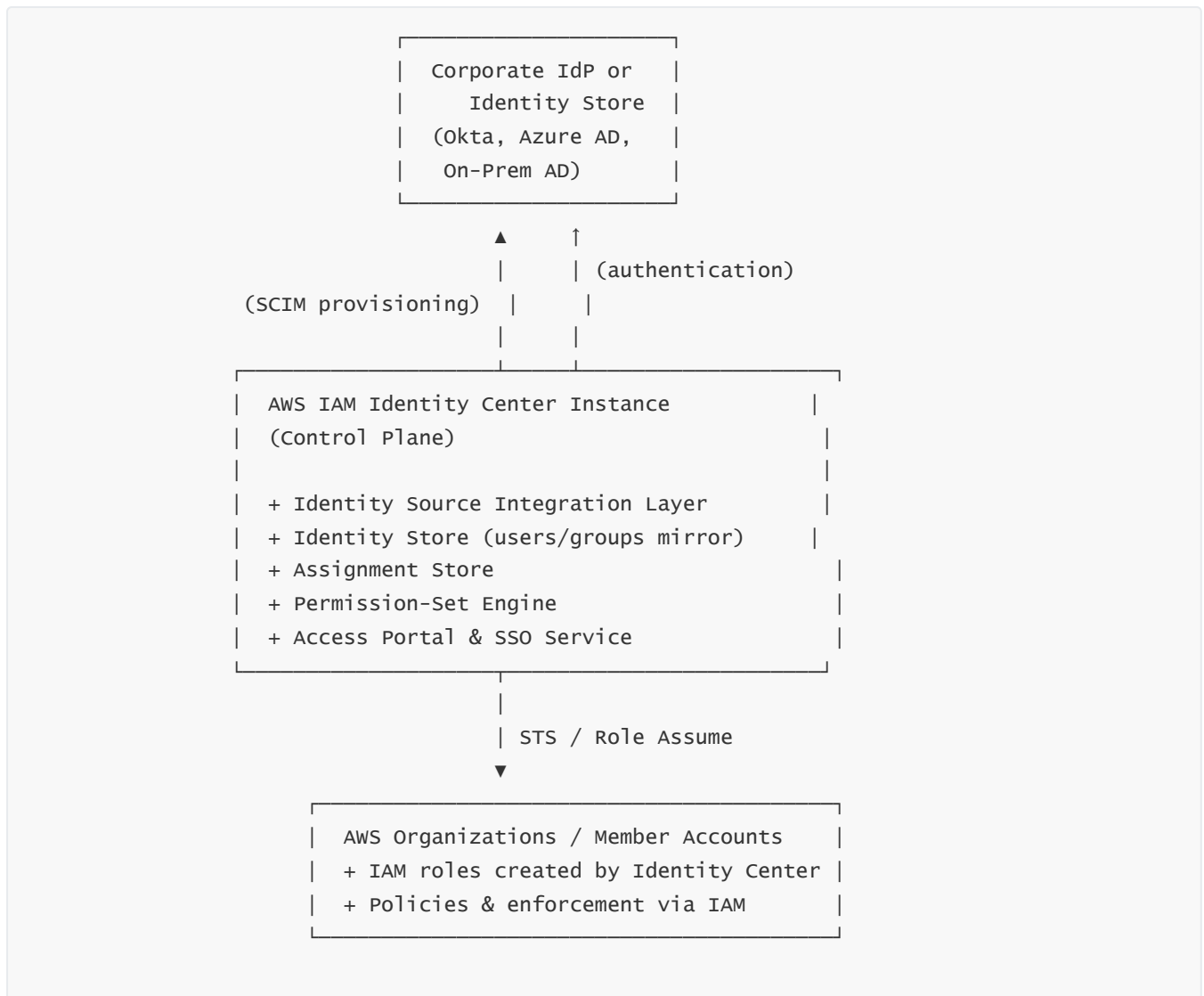
5 — Decision criteria & best practice considerations

When deciding which identity source to choose for your Identity Center implementation, architecturally you should weigh the following factors:

- **Existing identity infrastructure:** If you already have AD and manage users/groups there, AD integration might be the most seamless. If you have an external cloud-based IdP and want to unify identity across many apps, external IdP route may be optimal. If you’re small or just starting with AWS accounts, built-in directory may suffice.
- **User lifecycle and governance:** If you want one user record that is used across many applications (not just AWS), external IdP is likely best. If you only care about AWS accounts and have relatively simple identity needs, built-in may be easier.
- **High availability / disaster recovery:** Built-in directory is region-selected; you’ll want to plan for how to deal with Region failures. AD or external IdP may offer broader resilience but you must integrate locally.
- **Complexity and skillset:** External IdP integration usually requires SAML/SCIM understanding, certificate management, and provisioning flows. AD integration requires Directory Service skills and trust setups. Built-in directory requires least external integration.
- **Assignment and change impact:** Changing the identity source later is possible but costly and disruptive. AWS notes that when changing identity source, “users, groups, and assignments are deleted” in minimal scenarios. ([Amazon Web Services, Inc.](#))
- **Authentication method & MFA:** If you already have robust MFA, conditional access, identity protection in your IdP, external IdP gives you leverage of that. If using built-in directory, you’ll rely on IAM Identity Center’s own MFA/FIDO support. ([Amazon Web Services, Inc.](#))
- **Governance and assignments:** Regardless of identity source, you must still create assignments in Identity Center for users/groups → permission sets → accounts. The identity source choice does *not* replace that assignment logic; it only defines the origin of users and groups.

Architecturally, you should treat the identity source decision as *one of the foundational building blocks* of your access management topology: it affects authentication, identity lifecycle, provisioning, availability, and downstream assignment complexity.

6 — Architecture diagram: Identity Source integration flows



- In the diagram, the top box is your external identity system (could be corporate AD, Okta, Azure AD).
- The middle box is the Identity Center instance which integrates with that identity source (or uses built-in directory). Users/groups are mirrored into the Identity Store inside Identity Center.
- Assignments link those users/groups to permission sets and accounts.
- The bottom box is the multi-account environment where Identity Center provisions IAM roles and users assume them via STS.

This flows helps visualise how the identity source choice affects upstream and downstream components.

7 — Key pitfalls and migration considerations

- **Single identity source limitation:** You can only select *one* identity source at a time for an Identity Center instance. ([AWS Documentation](#)) If you need to support multiple directories or identity systems, you'll likely need federated groups in your IdP or other architectural workarounds.
- **Changing identity sources is disruptive:** When you switch from one identity source to another (e.g., from built-in to external IdP, or AD to external), AWS warns that users, groups, and assignments may be

deleted or need to be recreated. ([Amazon Web Services, Inc.](#)) It's not trivial and must be carefully planned.

- **Attribute mapping and provisioning gaps:** Especially for external IdP + SCIM, attribute mapping (e.g., `userName`, email, group membership) must align between IdP and Identity Center. Mismatch can cause duplicate users or provisioning failures. ([Intuitive](#))
- **Regional dependency:** If using AWS Managed AD or the built-in directory, you are tied to the Region where Identity Center is enabled. The directory's availability affects login and access to all accounts via that instance.
- **Assignments unaffected by identity source:** Changing the identity source does *not* automatically update or migrate assignments; you must plan how assignments map to new user IDs or identities.
- **Hybrid scenarios:** Some enterprises might want to use an external IdP for authentication but still use AD for groups, or vice versa. While technically possible via syncing, it adds complexity in provisioning, mapping, and user lifecycle.

Understanding these pitfalls ensures the identity source decision remains strategic, not just tactical.

8 — Summary: Choosing your identity source with clarity

In summary – architecturally, your identity source in IAM Identity Center determines **where identities live**, **how authentication happens**, **how provisioning is handled**, and **what dependencies exist**.

- If you're just starting or managing a small number of AWS accounts with minimal identity infrastructure, the built-in Identity Center directory is easiest and quick.
- If you already have corporate AD (on-premises or AWS Managed) and want to leverage existing identity infrastructure, integrating AD is the most enterprise-aligned.
- If you already use a cloud IdP (Okta, Azure AD, etc.) for your workforce and want a unified identity fabric across cloud, SaaS, and AWS, then external IdP integration is ideal.

Irrespective of the choice, it's critical to plan for: user lifecycle, group membership, provisioning, attribute mapping, region/resilience, change management, and how assignments will be managed.

As you move onward into the deeper architecture of IAM Identity Center (permission sets, flows, etc.), always keep in mind: the identity source is the *foundation* of the access management model.

4 — Authentication & Single Sign-On Flows (SAML, OIDC, External IdPs) in IAM Identity Center

1 — Big picture: what “authentication and SSO” mean in Identity Center

– When we talk about authentication and SSO in **IAM Identity Center (IIC)**, we're essentially talking about **how a human proves who they are** (authentication) and **how that single proof is reused to access multiple AWS accounts and applications** (single sign-on).

– At a high level, we always have **three layers** in our mental model:

1. **Identity Provider (IdP)** — where the user actually logs in (built-in directory, AD, Entra ID, Okta, etc.).
2. **IAM Identity Center** — the SSO/federation control plane that receives that proof, figures out the user's assignments, and orchestrates access.
3. **Target system** — AWS accounts (via IAM roles) or external apps (via SAML/OIDC) where the user finally ends up.

– All authentication/SSO flows are just **variations of this pattern**:

“User logs in at IdP → IdP sends proof to Identity Center → Identity Center issues appropriate session/token/assertion → user lands in target console/app with the right permissions.”

2 — Authentication modes: where does the actual login happen?

Identity Center supports three broad modes of where the user *authenticates*:

1. Built-in Identity Center directory

- The user logs directly into Identity Center's own login page with a username/password configured in the Identity Center directory.
- MFA (like FIDO2/WebAuthn) can also be configured within Identity Center.
- In this mode, Identity Center itself is the IdP.

2. Active Directory integration (AWS Managed AD / on-prem AD)

- The user authenticates against AD (usually using Kerberos/NTLM via Directory Service or federation).
- Identity Center relies on AD and Directory Service as the authentication backend but still presents a web login page that ultimately checks credentials in AD.

3. External IdP (SAML/OIDC, plus SCIM provisioning)

- The user never logs in “to Identity Center” directly. Instead, they hit a corporate login page (Okta, Entra ID/Azure AD, Ping, etc.).
- After successful login, the IdP issues a **SAML assertion** (or OIDC ID token) to Identity Center that tells AWS “this is user X, with these attributes.”
- Identity Center trusts that IdP and consumes that assertion/token as the authentication proof.

In all three cases, once the user is authenticated, Identity Center has a **logged-in session** for that user and can use it to drive SSO into AWS accounts and apps.

3 — The Identity Center portal session: what happens right after login

Once Identity Center knows **who** the user is, it needs to create a **session** that:

- Identifies the user (internal principal ID).
- Knows which **groups** the user belongs to (from the Identity Store, populated by built-in directory / AD / SCIM).
- Can use that identity to query the **assignment graph** (user/group → permission sets → accounts, and apps).

The high-level steps are:

1. **Assertion / credential validation**

- If external IdP is used, Identity Center validates the SAML assertion signature (ensuring it was issued by the trusted IdP, not tampered with, not expired).
- If built-in directory or AD is used, the credentials are validated, and the authentication is considered successful.

2. **Identity lookup in Identity Store**

- Identity Center maps the subject (e.g., NameID, email, unique user ID) from the assertion or login to a **user object** in its internal Identity Store.
- It retrieves **group memberships** for that user.

3. **Session creation**

- Identity Center creates a web session (with a cookie or equivalent mechanism) that marks this browser as authenticated as that specific user.
- This session has a **lifetime** (configurable to some extent, associated with policies or defaults).

4. **Assignment resolution & portal rendering**

- With user + groups known, Identity Center queries its assignment store and constructs the list of:
- AWS accounts + permission sets (account tiles).
- Applications (SAML/OIDC apps) assigned to that user.
- The portal page is rendered with those tiles, and the user now sees “what they can access.”

From the user’s perspective, this feels like: “I log in once, and I see all my AWS accounts and applications; I can click any tile and get in without logging again.”

4 — SSO flow for AWS Management Console (SAML/STS under the hood)

Let’s walk one deep console access flow step by step, because many other flows are just variations of this.

4.1 — Conceptual console SSO steps

1. **User authenticates** into Identity Center (directly or via external IdP, as described above).
2. Identity Center **builds the portal** view.
3. User clicks an **AWS account tile** and chooses a **permission set** (for example, `Dev-Admin` in `Account-Dev`, `ReadOnly` in `Account-Prod`).
4. Identity Center **maps that tile choice** to:
 - A specific IAM role ARN in the selected account (previously provisioned from the permission set).
 - Session settings (duration, etc.) defined in the permission set.
5. Identity Center’s SSO Service performs a **federation** into that role:
 - It calls STS `AssumeRole` (or equivalent) using an internal service principal that the IAM role trusts.
 - STS returns **temporary credentials** (`AccessKeyId`, `SecretAccessKey`, `SessionToken`) with the permissions of that role.

6. Identity Center then **generates a console URL** and session for the AWS Management Console, embedding or exchanging those STS credentials into a console sign-in token.
7. The browser is redirected to the **target account's console**, now authenticated as that IAM role. The user is inside that account without seeing any STS or SAML mechanics.

4.2 — What SAML is doing implicitly

Even though the user clicked a tile in Identity Center, under the hood AWS is still using a SAML-like or STS federation pattern:

- The Identity Center SSO service is effectively acting as an **IdP** for the AWS account's console, or more precisely, as a trusted principal calling STS on behalf of the user.
- There is a **trust relationship** on the IAM role in the member account, allowing the Identity Center service principal to assume it. The trust policy might look like "Principal: `sso.amazonaws.com` account/instance-specific conditions."
- Identity Center can pass **session tags** or attributes with STS, which later can be used in authorization (for example, attribute-based access control).

From a security perspective, the chain is: user authenticated → Identity Center session → Identity Center assumes role in account via STS → IAM policies of that role + SCPs decide what's allowed.

5 — SSO flow for AWS CLI v2 and SDKs (OIDC device authorization pattern)

Console access is just one side. The other critical path is **CLI and SDK** access via Identity Center.

5.1 — High-level CLI experience

From a user's point of view:

1. They configure an **AWS CLI profile** that uses `sso-session` (Identity Center) instead of static keys.
2. They run `aws sso login --profile <name>`.
3. CLI opens a browser (or prints a device code + URL) where the user logs in and approves.
4. Once done, CLI stores **cached OIDC tokens/STS credentials** locally (in `~/.aws/sso/cache`).
5. Subsequent CLI commands using that profile automatically exchange OIDC tokens for STS credentials and call AWS APIs as the assigned role.

5.2 — Under-the-hood steps (OIDC device auth)

Deep flow:

1. The CLI initiates a **device authorization flow** with Identity Center's OIDC endpoints (the exact endpoints are service-managed and discovered via configuration).
2. Identity Center returns:
 - A **user code** and verification URL (for the browser).
 - A **device code** that the CLI will poll with.

3. The user opens the verification URL, logs into Identity Center (or external IdP), and approves the login for that CLI device/session.
4. Identity Center verifies the device code and marks it as **authorized** for that user's identity and the selected account/permission set (if needed).
5. The CLI keeps polling Identity Center's OIDC token endpoint with the device code until it receives:
 - An **ID token** and **access token** (OIDC tokens) tied to that user and Identity Center instance.
6. The CLI then uses the Identity Center OIDC token to request **STS credentials** for a given account + permission set combination (the same as clicking a tile in the portal, but now via API).
7. STS returns temporary credentials; CLI caches them and uses them for API calls.

So architecturally:

- **Browser login** uses the same authentication path as the portal.
- **CLI** uses OIDC device authorization to link that browser login to the CLI session.
- Identity Center stands in the middle, converting user identity → allowed roles → STS credentials.

6 — SAML federation with external IdPs into Identity Center

When we use **external IdPs** (Okta, Entra/Azure AD, etc.), Identity Center itself is a **SAML Service Provider** (SP). The flow looks like this:

1. User attempts to access the Identity Center portal URL (SP-initiated) or clicks an app tile in the corporate IdP (IdP-initiated).
2. Identity Center redirects the user to the **IdP login page** if not already authenticated.
3. The user authenticates at the IdP (password, MFA, conditional access).
4. The IdP issues a **SAML assertion** to Identity Center's ACS (Assertion Consumer Service) endpoint.
 - Assertion is signed by the IdP's certificate.
 - Contains **NameID** and potentially attributes like email, user ID, etc.
5. Identity Center validates:
 - Signature (trusted cert).
 - Audience (is this assertion meant for Identity Center?).
 - Time validity (NotBefore, NotOnOrAfter).
6. Identity Center maps the **subject** and attributes from the assertion to a **user in the Identity Store**. This is where SCIM-provisioned users come in — Identity Center expects that the subject corresponds to a known user object.
7. If mapping succeeds, Identity Center creates a session and continues as normal (portal display, role assumptions). If mapping fails, the login fails because the user is unknown to Identity Center.

Key points:

- SAML is only used between **external IdP ↔ Identity Center**, not between Identity Center ↔ AWS Accounts (the latter uses STS).

- SCIM is critical to make sure the user that appears in the SAML assertion actually exists as an object in the Identity Store with right groups.

7 — SAML and OIDC SSO out from Identity Center to external applications

Identity Center can also act as an **IdP** for other applications (SaaS or custom). In this direction, Identity Center issues SAML assertions (or OIDC tokens) to apps when users click app tiles in the portal.

7.1 — SAML out to apps

1. After a user is logged into Identity Center and sees the portal, they click an **application tile** configured in Identity Center.
2. Identity Center constructs a **SAML assertion** as an IdP:
 - It uses a configured **SAML application configuration** (ACS URL of the app, expected audience, required attributes).
 - It inserts attributes (e.g., email, username, group) based on the Identity Store objects and attribute mappings.
 - It signs the assertion with Identity Center’s private key; the SP (the app) has the corresponding public certificate.
3. The browser is redirected to the app’s ACS URL, carrying that SAML assertion (POST binding).
4. The application validates the assertion and creates a session for the user, trusting Identity Center as IdP.

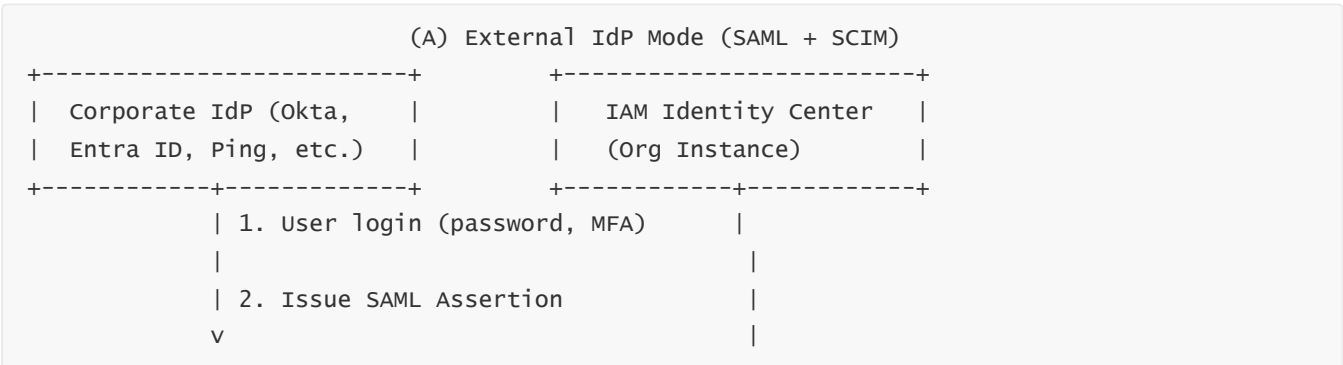
7.2 — OIDC out to apps

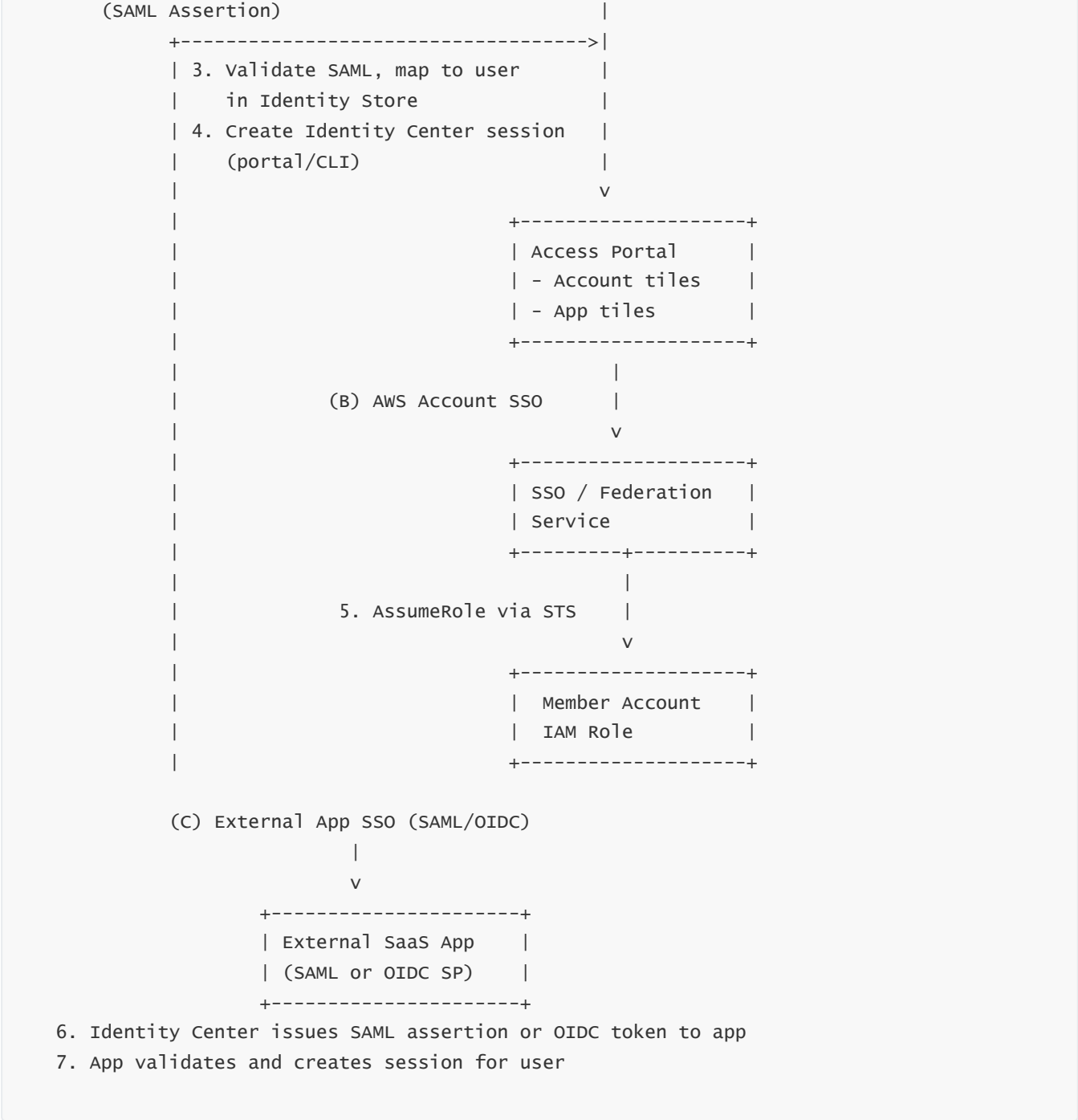
Similarly, if Identity Center supports OIDC for an app:

1. The app is configured as an OIDC client with Identity Center as **authorization server**.
2. When the user clicks the tile, Identity Center uses an **OIDC authorization flow** to issue ID tokens/Access tokens to the app.
3. The app validates the token (signature, audience, issuer) and authorizes the user accordingly.

In both cases, Identity Center is now the **center of trust** for those apps, just as it is for AWS accounts. It uses the same Identity Store and assignments to decide which apps appear in the portal.

8 — Detailed ASCII diagram: end-to-end auth & SSO flows





- **Block (A):** SAML authentication from external IdP into Identity Center.
- **Block (B):** Internal federation from Identity Center into AWS accounts via STS and IAM roles.
- **Block (C):** SAML/OIDC SSO outwards from Identity Center to external apps.

This diagram shows that:

- SAML can appear on **both sides** (inbound from IdP and outbound to apps), but STS/IAM is always the path to AWS accounts.
- Identity Center is the central **federation hub** where decisions and mappings are made.

9 — Session lifetimes, MFA, and security controls in auth flows

For deep understanding, we should think about **session lifetime** and **MFA placement**:

– MFA & conditional access

- If you're using an external IdP, MFA and conditional access policies (device checks, IP rules, risk detection) are **enforced at the IdP**. Identity Center just trusts the SAML assertion.
- If you use the built-in directory, MFA can be configured in Identity Center itself. In that case, Identity Center handles MFA challenge and stores the result in the user's session state.

– Portal session

- Has its own duration. When it expires, the user must reauthenticate at the IdP/Identity Center.
- The lifetime is usually shorter than the absolute maximum allowed by the IdP policies.

– STS session for AWS account roles

- Each permission set can specify a **session duration** (for example 1–12 hours). That controls how long the STS credentials for that IAM role remain valid.
- When STS credentials expire, the user must either:
 - Re-click the account tile in the portal to get a new session, or
 - Re-run the CLI login flow to refresh credentials.

Security design patterns:

- Use short STS session durations for **sensitive roles**, balanced with usability.
- Put strong MFA and conditional access at the **IdP** for all workforce identities.
- Use **session tags** and attribute-based access control when you want to leverage user attributes (department, environment, etc.) in IAM policies.

10 — Comparative view: SAML vs OIDC in Identity Center context

It's easy to mix these two, so let's anchor them:

– SAML

- XML-based, older but very widely used for enterprise SSO.
- Used heavily for:
 - **External IdP** → **Identity Center** (IdP authenticates, sends SAML assertion).
 - **Identity Center** → **external apps** (Identity Center acts as IdP for SAML SPs).
- SAML assertions are pushed via browser redirects/POSTs.

– OIDC (OpenID Connect)

- JSON/REST-based layer on top of OAuth 2.0.
- Used heavily for:
- **CLI device authorization** flows (CLI ↔ Identity Center).
- Some app integrations where Identity Center acts as OIDC provider (if supported).
- Tokens (ID/Access) are obtained via HTTP APIs and validated with JWT semantics.

In practice:

- For **human browser SSO** with corporate IdPs and SaaS, SAML is still extremely common.
- For **programmable client** (CLI/SDK) flows, OIDC is the natural fit.

Identity Center stitches them together by consuming SAML from IdPs, issuing OIDC for CLI, and using STS on the backend to ensure access to AWS accounts.

11 — Putting it all together in one mental model

We can summarize all authentication and SSO flows in IAM Identity Center with one unified perspective:

1. Authentication

- User authenticates at **some IdP** (Identity Center’s own directory, AD, external IdP).
- Identity Center receives a **trusted assertion or credential** representing that user.

2. Identity Resolution & Assignment

- Identity Center identifies the user + groups, via Identity Store (fed by directory or SCIM).
- It resolves assignments: which AWS accounts and apps the user can see.

3. SSO to AWS

- When the user chooses an account + permission set or CLI profile, Identity Center performs **STS AssumeRole** into the target account using a managed IAM role that was created from a permission set.
- The user ends up in the console or making API calls with those temporary credentials.

4. SSO to Apps

- When the user chooses an app tile, Identity Center issues **SAML/OIDC tokens** to that app, acting as IdP.

5. Security & lifetime

- MFA and conditional access typically applied at IdP.
- Session durations (portal, STS, CLI) define how long each hop remains valid.

If we keep that mental model, every detailed flow (browser vs CLI, SAML vs OIDC, internal vs external apps) is just a different “path” through the same Identity Center hub.

5 — Permission Sets Deep-Dive (Core Concept, Internals, Mechanics)

1 — What a Permission Set *actually* is (conceptual model)

– In IAM Identity Center, a **permission set** is not an IAM role and not an IAM policy by itself. It is a **central template** that describes:

1. **Which permissions** a person should get (policies), and
2. **How** their session should behave (session duration, relay state, etc.)
when they access a specific AWS account through Identity Center.

– Internally, we can think of a permission set as a “**role blueprint**” that Identity Center uses to automatically create and keep IAM roles in each target account. For every (permission set, account) pair that you assign, Identity Center will **provision an IAM role** inside that account, based on the blueprint defined by that permission set.

– So the mental mapping is:

Permission set (in Identity Center) → one IAM role per account (in each target AWS account)

– That means:

– When we **change** a permission set, we’re changing the blueprint, and Identity Center will reconcile and update the corresponding IAM roles in all accounts where it is used.

– When we **assign** a permission set to new accounts, Identity Center will create those IAM roles in the new accounts.

– When we **remove** the last assignment of that permission set from an account, Identity Center can delete or de-link the associated role (depending on state).

2 — Internal structure of a Permission Set: what fields it really contains

A permission set contains (conceptually) several major categories of configuration:

1. Permission policies

- You can attach one or more **AWS managed policies** (like `ReadOnlyAccess`, `PowerUserAccess`, etc.).
- You can attach **customer managed policies** that live in the management account.
- You can define an **inline policy** (JSON) directly in the permission set, which Identity Center will copy into an inline policy on the provisioned IAM role.
- Together, these become the *effective* IAM permissions of the role Identity Center creates in each account.

2. Session configuration

- **Session duration** (e.g., 1 hour to 12 hours). This controls how long the STS credentials are valid when a user assumes the role via Identity Center.
- **Relay state** (optional) that defines where in the AWS console to land after SSO (for example, go directly to CloudWatch logs or a certain service console).
- **Session tags** or attribute mappings (in more advanced setups) where user attributes can flow into role sessions.

3. Optional advanced settings (depending on features enabled)

- Whether to use a **permissions boundary** on the target IAM role.
- Tagging for roles that will be created in target accounts (to manage them via automation).
- Any custom namespaces or name-patterns used for the created roles.

From an internal point of view, you can imagine the permission set stored as a structured object like:

```
PermissionSet:
- Id: ps-123
- Name: "Dev-Admin"
- Description: "Developer admin in nonprod"
- AttachedPolicies:
  - AWSManaged: ["AdministratorAccess"]
  - CustomerManaged: ["CompanyCommonDeveloperGuardrails"]
  - InlinePolicy: { ... JSON ... }
- SessionDuration: 4h
- RelayState: "https://console.aws.amazon.com/cloudwatch/home"
- OtherSettings: { ... }
```

Identity Center never exposes it to accounts as a “permission set object”; instead it translates this into **IAM role configuration per account**.

3 — How permission sets turn into IAM roles (provisioning mechanics)

The key internal mechanic is: **for each assignment of a permission set to an account, Identity Center either ensures or creates an IAM role in that account.**

3.1 — Naming and creation

- When you assign permission set `Dev-Admin` to `Account A` and `Account B`, Identity Center:
 1. Checks whether an IAM role already exists in the account for that permission set (using internal mappings).
 2. If not, creates a new IAM role, for example with a name pattern like `AWSReservedSSO_Dev-Admin_<random>` (AWS uses reserved naming patterns to avoid collision; exact names are generated and should be treated as managed).
 3. Attaches to that role:
 - The AWS managed policies listed in the permission set.
 - The customer managed policies.
 - The inline JSON policy as an **inline role policy**.
 4. Configures the **trust policy** of that role so that **IAM Identity Center’s service principal** can assume it on behalf of your users.
- This is done via the **Account Provisioning Engine** that calls IAM APIs in each account. It does so asynchronously and eventually consistently.

3.2 — Trust relationship

– The IAM role in each account has a trust policy something conceptually like:

```
"Principal": {
  "Service": "sso.amazonaws.com"
},
"Condition": {
  "StringEquals": {
    "sso.amazonaws.com:accountId": "<management-account-id>",
    "sso.amazonaws.com:permissionSetArn": "arn:aws:sso:::permissionSet/ps-123"
  }
}
```

(not exact, but conceptually: trust Identity Center in that org instance, tied to that permission set)

– This ensures that **only the correct Identity Center instance** can assume that role, not arbitrary AWS services or other principals.

– When a user clicks the account tile or uses CLI, Identity Center uses this trust to call `AssumeRole` and issue temporary credentials.

3.3 — One permission set → many roles

– For a single permission set:

– If it's assigned to 10 accounts, you'll end up with **10 roles** (one per account), all based on the same blueprint.

– If you later add 5 more accounts, Identity Center automatically creates **5 more roles**.

– If you reduce it to 8 accounts, Identity Center may delete or at least stop using some of the roles.

– So a permission set is effectively a **“role family template”**: “all of these roles in all these accounts should look like this.”

4 — The assignment triangle: user/group ↔ permission set ↔ account

To fully understand the mechanics, we need to see how permission sets sit in the middle of a **three-way mapping**:

- **Who**: user or group.
- **What**: permission set.
- **Where**: AWS account.

We can view it like:

```
User/Group + Permission Set + Account = Effective IAM Role session
```

– Identity Center stores **assignments**:

1. "Group X has permission set `Dev-Admin` in `Account A` and `Account B`."
2. "User Y has permission set `ReadOnly` in `Account SharedServices`."

– When a user logs in:

1. Identity Center looks at the user's **direct assignments**.
2. Expands all their **group memberships** and collects assignments from those groups.
3. For each `(user or group, permission set, account)` combination, it identifies the **corresponding IAM role ARN** in that account (which the provisioning engine created).
4. Those account + permission set combinations become the **tiles** in the portal and the roles available via CLI.

So, permission sets are the **connecting point** between identity (user/group) and the IAM role in each account.

5 — Deep internal lifecycle of a permission set

Let's break down the **full lifecycle**:

5.1 — Creation

– Admin defines a new permission set:

1. Name, description.
2. Attach AWS managed policies / customer managed policies.
3. Add inline policy JSON (if needed).
4. Configure session duration, relay state, etc.

– At this moment, nothing is created in the accounts yet. The permission set is **just a template**.

5.2 — Assignment

– Admin then **assigns** this permission set to:

1. One or more **users or groups**, and
2. One or more **accounts** (selected from the Organization).

– At assignment time, Identity Center:

1. Writes an entry in the **assignment store** (linking user/group, permission set ID, and account ID).
2. Triggers the **Account Provisioning Engine** to provision roles for each `(permission set, account)` pair that doesn't yet have a role.

5.3 — Provisioning / update

– The provisioning engine:

1. Enumerates accounts where this permission set is assigned.
2. For each account:

- Creates the IAM role if needed.
- Attaches or updates the policies.
- Ensures the role trust policy is correct.

3. Maintains an internal mapping: `permissionSetId + accountId → roleArn`.

- This is asynchronous; we must assume that large-scale changes (e.g., 500 accounts) may take some time.

5.4 — Modification

- When a permission set is **modified**:

1. The template in Identity Center is updated.
2. Identity Center calculates the list of accounts where this permission set is in use.
3. The provisioning engine runs again to update each IAM role in each account to match the new template:
 - Add/remove policies.
 - Update inline policy.
 - Adjust session duration (which affects new STS sessions).

- This is how we achieve **centralized permission management**: change the permission set once, Identity Center pushes the change to all roles derived from it.

5.5 — Deletion or unassignment

- If you **remove** assignments:

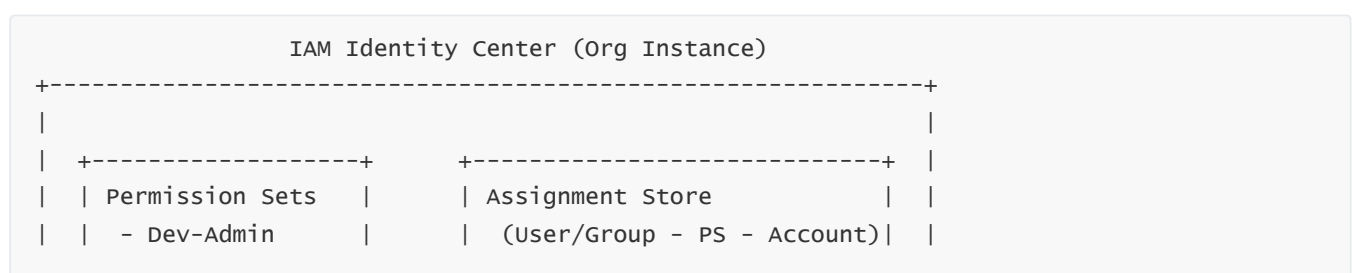
1. When you remove all account assignments for that permission set, Identity Center can delete or mark the corresponding IAM roles as no longer used.
2. If you keep some account assignments, Identity Center maintains roles only in those remaining accounts.

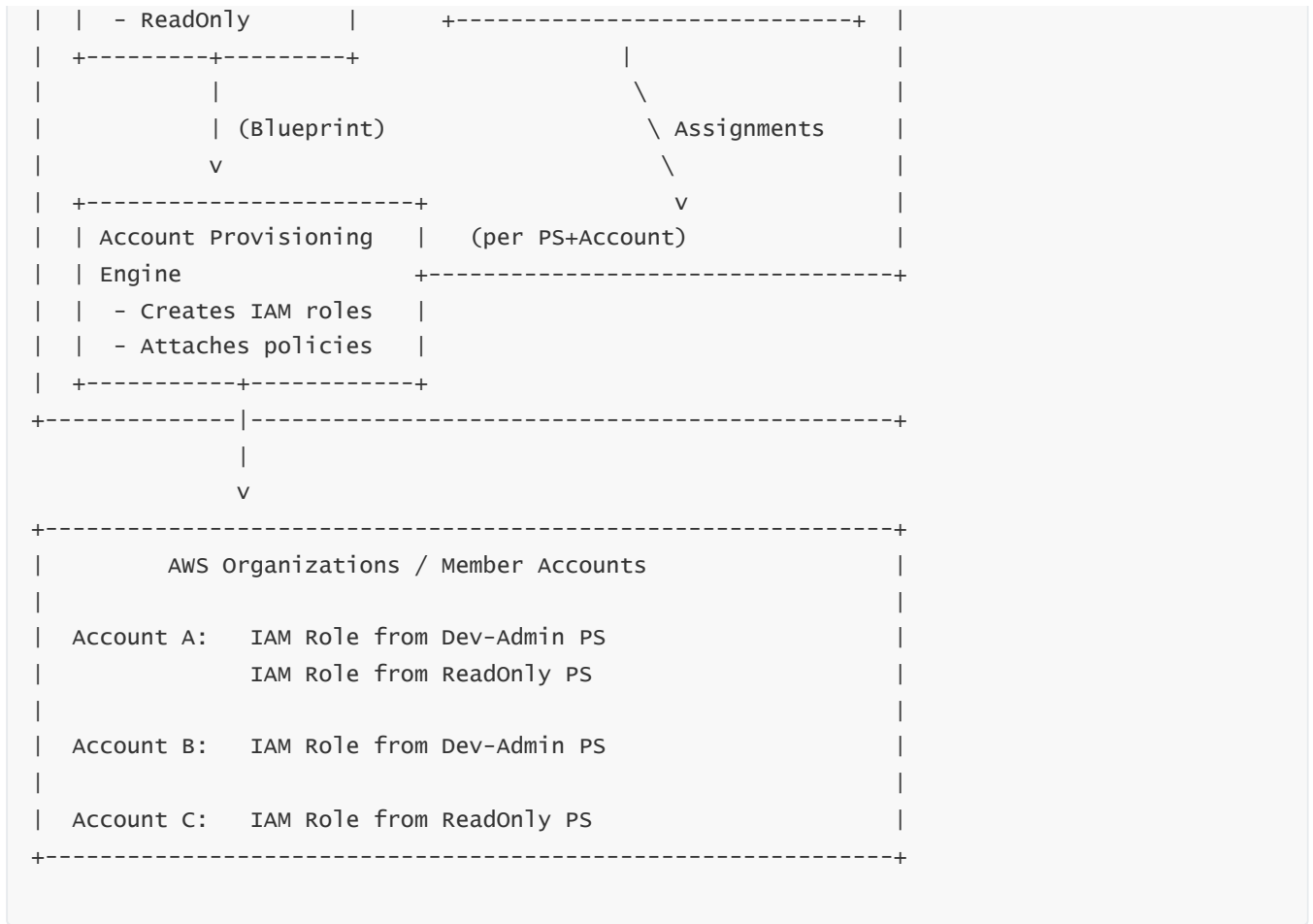
- If you **delete** the permission set itself:

1. Identity Center must ensure there are no remaining assignments.
2. It will remove or orphan associated roles (implementation details may vary, but logically those roles should no longer be used).

From a governance lens, you almost always want to clean up permission sets carefully, as they are central to your access model.

6 — ASCII architecture diagram: permission set as a role blueprint





- Top left: permission sets define **what** each role should look like.
- Top right: assignment store defines **who** gets what in which account.
- Middle: provisioning engine materializes that into **actual IAM roles** in each account.
- Bottom: each account ends up with a set of **managed roles**, one per (permission set, account) pair in use.

7 — Permission sets vs IAM policies vs IAM roles: clearing confusion

A lot of confusion comes from mixing these three.

7.1 — IAM policies

- IAM policies (customer managed, AWS managed) are just **JSON documents** that say: "Allow/deny these actions on these resources, optionally under these conditions."

7.2 — IAM roles

- IAM roles are **identities** inside an AWS account. They have:
 - A trust policy (who can assume the role).
 - One or more attached permission policies (what the role can do).

– Optional session tags, permissions boundaries, etc.

7.3 — Permission sets

– Permission sets live **only in Identity Center** and have no direct meaning in IAM.

– A permission set is a **bundle**:

(Policies) + (Session configuration) → used by Identity Center to create IAM roles in accounts.

So:

- Policies: building blocks.
- Permission sets: templates built from policies.
- IAM roles: deployed instances of templates in each account.

This separation is what lets us manage things centrally (via permission sets) and still rely on IAM roles and policies for actual enforcement.

8 — Patterns and best practices for designing permission sets

Designing permission sets well is **crucial** for a scalable and understandable multi-account strategy. Think in patterns:

8.1 — Environment-based patterns

– Example sets:

1. `Dev-Admin` – Full admin in dev accounts only.
2. `Dev-PowerUser` – Almost admin but restricted from security changes.
3. `Prod-ReadOnly` – Strict read-only in production accounts.
4. `Prod-Operator` – Operational permissions only (e.g., restart instances, read logs) but not infrastructure admin.

– These permission sets are then **assigned to OU/groupings** of accounts (dev OU, prod OU) and to the relevant human groups (`DeveLopers`, `Ops`, etc.).

8.2 — Persona-based patterns

– Example sets:

1. `Security-Analyst` – Access to CloudTrail, GuardDuty, Security Hub, read-only in all accounts.
2. `Data-Engineer` – Access to S3, Glue, Athena, certain KMS actions.
3. `Platform-Admin` – Full admin in shared platform accounts.

– Each persona-based permission set can be reused across many accounts; the combination of persona + environment (Dev/Prod) gives you fine control.

8.3 — Minimal and layered design

- Instead of creating one gigantic “super permission set” that tries to do everything, use **small, well-defined permission sets** that reflect real job roles and duty boundaries.
- This approach:
 - Simplifies reasoning about “who can do what.”
 - Allows reuse and reduces duplication.
 - Helps with compliance, audits, and least-privilege.

8.4 — Align with Organizational Units (OUs)

- Use AWS Organizations OUs for accounts, and design permission sets in such a way that:
 - There is a consistent mapping like: “For every dev account in the Dev OU, group `DevTeam` gets permission set `Dev-Admin`.”
- This can be automated via IaC (CloudFormation, Terraform, etc.) using Identity Center APIs.

9 — How permission sets interact with SCPs and other guardrails

Even if a permission set grants strong permissions, **SCPs (Service Control Policies)** and other guardrails can still override them:

- **SCPs:**
 - Applied at Organization root/OU/account level.
 - Define what is *allowed at all* in those accounts.
 - If an SCP denies an action, no permission set or role policy can grant it.
- **Permissions boundaries** (if you use them in the roles created from permission sets):
 - Limit the maximum permissions that the role can ever get, regardless of its policies.
- So, permission sets should be designed **in context**:
 1. SCPs define the global boundaries.
 2. Permission sets define what a user *should* have within those boundaries.
 3. Optionally, permission boundaries further restrict roles derived from permission sets.

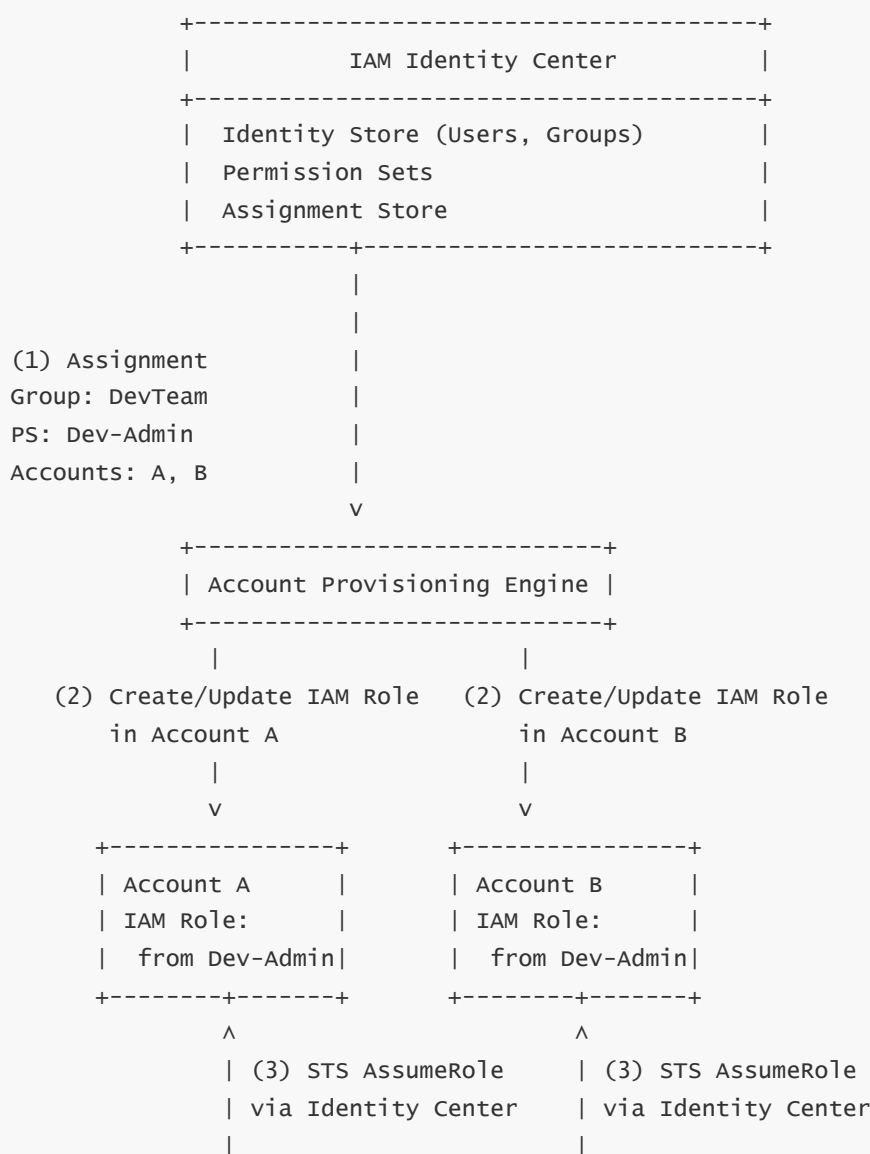
From a security architecture viewpoint, permission sets are the **central knob** for human access, but **not the only control**; SCPs and IAM boundaries are the underlying safety rails.

10 — CLI integration and permission sets: how profiles map

When a user uses the AWS CLI with Identity Center:

- The CLI config profile refers to:
 - A **start URL / sso-session** (the Identity Center instance).
 - A **permission set** (or role) and **account ID** pair.
 - That mapping is exactly the same as the Assignments we discussed:
 - The CLI essentially requests: "For user X, give me credentials for permission set **Y** in account **Z**."
 - Identity Center then uses **the IAM role created from that permission set in that account** to obtain STS credentials.
 - If you change the permission set (e.g., add a policy), the next time the user obtains credentials, the underlying role (after being updated by provisioning engine) will reflect the new permissions—no CLI change needed.
- So permission sets are the **source** that the CLI indirectly uses via Identity Center.

11 — ASCII diagram: permission sets driving roles, users, and CLI



```
+-----+-----+-----+
|           Users / CLI / Portal           |
| - User from DevTeam logs in               |
| - Portal shows Account A/B tiles with Dev-Admin option |
| - CLI profile uses sso-session + account-id + permission-set |
+-----+-----+-----+
```

This diagram highlights:

- Permission sets → provisioning engine → IAM roles.
- Assignments → portal tiles and CLI mapping.
- Users always access AWS through those IAM roles derived from permission sets.

12 — Operational considerations and pitfalls around permission sets

A few common operational realities:

1. Propagation delay

- When you create/modify a permission set and assign it to many accounts, it may take some time before all IAM roles are updated.
- During that time, some accounts may reflect new permissions while others do not yet.

2. Name mismatch with old manual roles

- If you previously had manually created roles like `DevAdminRole` in accounts, Identity Center will still create its own `AWSReservedSSO_*` roles.
- It's usually best to migrate fully to Identity Center-managed roles to avoid confusion.

3. Policy size limits

- Each inline policy and each role has IAM policy size limits. Over-stuffing a permission set with huge inline JSON can hit limits in the resulting IAM role.
- Better to break out large permission domains into multiple permission sets or managed policies.

4. Changing identity source

- If you change the Identity Center identity source (e.g., from internal directory to external IdP), the permission sets themselves can survive, but **assignments to users/groups may be impacted** because identity IDs change.
- That means your designed permission sets remain valid, but mapping of “who gets them” may need rework.

5. Overlapping permission sets

- A user can receive multiple permission sets in the same account (e.g., `ReadOnly` and `Dev-Admin`).
 - They will see multiple choices in the portal and CLI to assume different roles.
 - Carefully design so that users don't have confusing or overly broad combinations unless that's intended.
-

13 — Final mental model for permission sets

To wrap it all up, the **mental shortcut**:

Permission sets are centralized role templates that define the permissions and session behavior for human access to accounts via IAM Identity Center.

Identity Center uses these templates to create and update **IAM roles in each account**, and assignments connect **users/groups** to those **permission sets** in specific **accounts**.

So when we design and manage permission sets well, we:

- Gain a **single central place** to adjust human permissions across many accounts.
- Keep IAM roles in accounts managed and consistent.
- Allow CLI/console/SSO flows to be driven cleanly from the Identity Center configuration, not one-off per account.

6 — AWS Account Assignment Pipeline and Multi-Account Access Management

1 — What “assignment pipeline” actually means in Identity Center

When we say **AWS account assignment pipeline** in IAM Identity Center, we’re talking about the *full end-to-end machinery* that takes a conceptual decision like:

“Group DevTeam should have Dev-Admin access in all Dev accounts”

and turns it into:

- Actual **IAM roles** in each AWS account.
- Actual **SSO tiles** in the Identity Center portal.
- Actual **STS sessions** for CLI/console access.

So the **pipeline** is everything between:

1. The admin’s configuration action (create assignment), and
2. The user’s final ability to click a tile / run CLI and get into a specific account with a specific role.

It touches:

- Identity Store (user/group).
- Permission Sets.
- AWS Organizations (account list).
- The **Account Provisioning Engine** (role creation/update).
- The **Assignment Store** (who-gets-what-where).
- SSO/STS at runtime.

Multi-account access management is simply running this same pipeline **at scale** across tens / hundreds / thousands of accounts, with strong structure and governance instead of per-account chaos.

2 — Core data model: who, what, where

At the heart, the assignment pipeline is built around a **3-way relationship**:

- **Who**: user or group (from Identity Store).
- **What**: permission set (the role blueprint).
- **Where**: AWS account (from AWS Organizations).

You can think of a single assignment record as:

```
Assignment:
  Subject:   <User or Group ID>
  PermissionSet: <PermissionSet ID>
  Account:   <Account ID>
```

From this **single conceptual record**, Identity Center can derive:

- For **admin**: “Who is allowed what in which accounts?”
- For **user**: “Which tiles and roles do I see?”
- For **provisioner**: “What IAM roles must exist in those accounts?”
- For **STS**: “Which role ARN should I assume when this subject selects that tile?”

All multi-account logic is created by **large numbers of these assignments**, often driven by groups and sometimes by automation.

3 — Full assignment pipeline: end-to-end flow

Let’s walk the complete path from configuration to user access.

3.1 — Step 1: Admin chooses identity + permission set + accounts

Admin performs an action like:

“Assign permission set Dev-Admin to group DevTeam for accounts A, B, C.”

Behind the scenes:

1. Identity Center looks up `DevTeam` in the Identity Store (subject = group ID).
2. Confirms that `Dev-Admin` permission set exists.
3. Looks up accounts A, B, C via AWS Organizations.

Then, for each account:

- It creates a logical **Assignment** object in the Assignment Store.

So from the admin’s one configuration action, we get **N assignment records** (one per account).

3.2 — Step 2: Assignment Store captures the mapping

The Assignment Store is now effectively storing:

```
(DevTeam, Dev-Admin, AccountA)
(DevTeam, Dev-Admin, AccountB)
(DevTeam, Dev-Admin, AccountC)
```

This is just **metadata** so far — no IAM changes yet (or they're pending).

3.3 — Step 3: Account Provisioning Engine reacts

The moment new `(PermissionSet, Account)` combinations appear, the **Account Provisioning Engine** steps in:

1. For each unique `(Dev-Admin, AccountX)` combination, it checks if there's already an IAM role created from that permission set in that account.
2. If not, it **creates** a new IAM role:
 - Role name using the reserved pattern (e.g., `AWSReservedSSO_Dev-Admin_<hash>`).
 - Attach policies from permission set.
 - Set trust policy to trust the Identity Center instance.
3. If the role already exists but the permission set changed recently, it **updates** policies / properties to match the current template.

Notice: the provisioning engine cares about **permission set + account**, not about which users/groups are attached. One role per `(permission set, account)`; many users/groups can share it.

3.4 — Step 4: Portal & CLI runtime use assignments

Once the assignment exists and the IAM role is provisioned:

- When a user from group `DevTeam` logs in, Identity Center expands their group membership and sees they belong to `DevTeam`.
- It queries the Assignment Store for all `(subject, permission set, account)` where `subject` is either the user or one of their groups.
- For each such record, it looks up the **role ARN** that corresponds to `(permission set, account)` in the provisioning mapping.
- The portal shows tiles like:
 - `AccountA - Dev-Admin`
 - `AccountB - Dev-Admin`
 - `AccountC - Dev-Admin`
- For CLI, the Identity Center OIDC/STS flows obtain credentials for those same `(permission set, account)` pairs.

So the pipeline is:

Admin writes assignment → Provisioning engine ensures IAM role → User sees tile / CLI entry → SSO/STS uses the role to access the account.

4 — ASCII diagram: the assignment pipeline



This diagram captures the **forward flow** (admin action down to accounts) and the **** runtime loop**** (user login back up to roles).

5 — Assignment types: direct vs via groups

In Identity Center, we can assign permission sets to:

- **Users directly** – (User, PermissionSet, Account)
- **Groups** – (Group, PermissionSet, Account)

At runtime:

- Identity Center resolves **all groups the user belongs to** (from Identity Store; SCIM/AD integration).
- It collects assignments where the subject is any of:
 - The user's own ID.
 - Any of the group IDs the user is a member of.

The combination becomes the user's total entitlement set.

Best practice and architecture wise:

- Use **group-based assignments** as the primary model.
- Assign permission sets to groups, and put users into groups in your identity source (AD/IdP).
- Avoid too many per-user assignments (becomes unmanageable, breaks clean lifecycle when people move teams).

So the moral:

Architecturally, the assignment pipeline works best when identity source groups are the “switches,” and permission sets are the “permission blocks” you connect to accounts.

6 — How AWS Organizations drives multi-account awareness

Identity Center is deeply tied to **AWS Organizations** in the org instance model:

- It uses Organizations to know:
 - Which accounts exist.
 - Their IDs and names.
 - Their OU structure.

This is essential for multi-account management:

1. When new accounts are **created** (via AWS Organizations), Identity Center can later target them in assignments.
2. When accounts are **moved** between OUs, your external tooling may adjust assignments based on OU membership (e.g., all Dev OU accounts get the `Dev-Admin` permission set).
3. When accounts are **closed / removed**, Identity Center eventually cleans up roles and assignments.

Identity Center itself doesn't automatically apply “per-OU policies,” but **you can layer automation** on top:

- Solutions that say: “Whenever an account joins OU `Dev`, assign `DevTeam` → `Dev-Admin` permission set automatically.”

- That automation uses `sso-admin` APIs to create assignments; Organizations only supplies the account structure.

So for multi-account management, the **triangle** becomes:

```
Identity Source Groups --> Permission Sets
AWS Organizations OUs  --> Accounts to attach those PS to
```

You build patterns like:

- “Group `DevTeam` gets `Dev-Admin` in all accounts in Dev OU.”
- “Group `Securityops` gets `Security-Analyst` in all accounts.”
- “Group `BillingTeam` gets `Billing-View` only in Master Billing / Finance accounts.”

7 — Scale behavior: many accounts, many assignments

In a large enterprise, typical numbers might be:

- 200–500 AWS accounts.
- 20–40 permission sets.
- 10–30 major groups.
- Thousands of users.

Internally, this means:

- Potentially **thousands of assignments** and **thousands of IAM roles**.

Effects:

1. Provisioning time

- Large changes (new permission set across 300 accounts) means 300 IAM roles must be created/updated.
- Identity Center’s provisioning engine must respect IAM and Organizations rate limits; operations are batched and asynchronous.
- Admins must anticipate that “`create assignment → immediate effect everywhere`” is not guaranteed; there is a propagation window.

2. Complexity and change impact

- Changing a widely used permission set’s policy triggers updates to all derived roles.
- You must think carefully before editing a “shared” permission set used across many environments.

3. Visibility

- Governance teams need ways to see:
 - Which groups have which permission sets in which accounts.
 - Which permission sets exist and how they’re used.
- That often drives the need for **reporting scripts** using `sso-admin` APIs.

Architecturally, your strategy must assume:

The assignment pipeline is a distributed system across many accounts, with eventual consistency and heavy reuse of permission set templates.

8 — Runtime access resolution in detail

Let's go through the exact steps when a user logs in and chooses a tile, focusing on the assignment pipeline logic.

8.1 — User login & identity resolution

1. User authenticates (built-in, AD, external IdP).
2. Identity Center resolves:
 - Internal **user object ID**.
 - All **groups** that user belongs to.

8.2 — Assignment resolution

1. Identity Center queries the Assignment Store:
 - Find all assignments where **Subject** equals either:
 - The user ID, or
 - Any group ID in the user's group memberships.
2. It returns a list of **(PermissionSetID, AccountID)** pairs (plus subject links, but we don't need those at runtime beyond entitlements).

8.3 — Role mapping

1. For each **(PermissionSetID, AccountID)** pair, Identity Center uses its provisioning map to find the **IAM role ARN** created earlier (e.g., **arn:aws:iam::123456789012:role/AWSReservedSSO_Dev-Admin_xyz**).
2. These ARNs are then packaged into:
 - Portal tiles.
 - CLI profile possibilities.

8.4 — STS session issuance

1. When the user selects a specific tile (say AccountB + Dev-Admin):
 - Identity Center's SSO service calls **AssumeRole** with that ARN, plus session name, optional tags, etc.
 - STS returns temporary credentials with the correct session duration, as defined in the permission set.
2. Identity Center redirects the browser to the AWS console for that account with that role, or for CLI, the authenticated device uses these credentials for API calls.

Notice: **assignments are read at runtime** to determine which role ARNs are valid for a given user; roles themselves are **pre-provisioned** by the engine.

9 — Multi-account strategy patterns based on the assignment pipeline

The pipeline enables several powerful patterns:

9.1 — OU-based access spread

- All accounts in OU `Dev` share similar structure (VPCs, toolsets).
- You create permission sets for Dev personas: `Dev-Admin`, `Dev-Engineer`, `Dev-ReadOnly`.
- Then assign:
 - Group `DevTeam` + `Dev-Admin` to all accounts in Dev OU.
 - Group `QaTeam` + `Dev-ReadOnly` to all dev and staging accounts.

This ensures:

- As new dev accounts are added to OU, automation can **create assignments** for them.
- Role names and privileges remain consistent across accounts.

9.2 — “Global function” roles

- Group `Securityops` needs read/incident access to *all* accounts.
- You create permission set `Security-Analyst`.
- Assign `Securityops` + `Security-Analyst` to every account in the organization (or all except some exceptions).

Result:

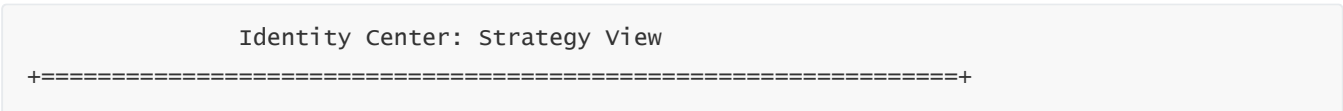
- Any SecurityOps engineer can, via the portal, pick any account and get into it with a security-focused read/ops role.
- Permission set updates propagate security best practices to all accounts.

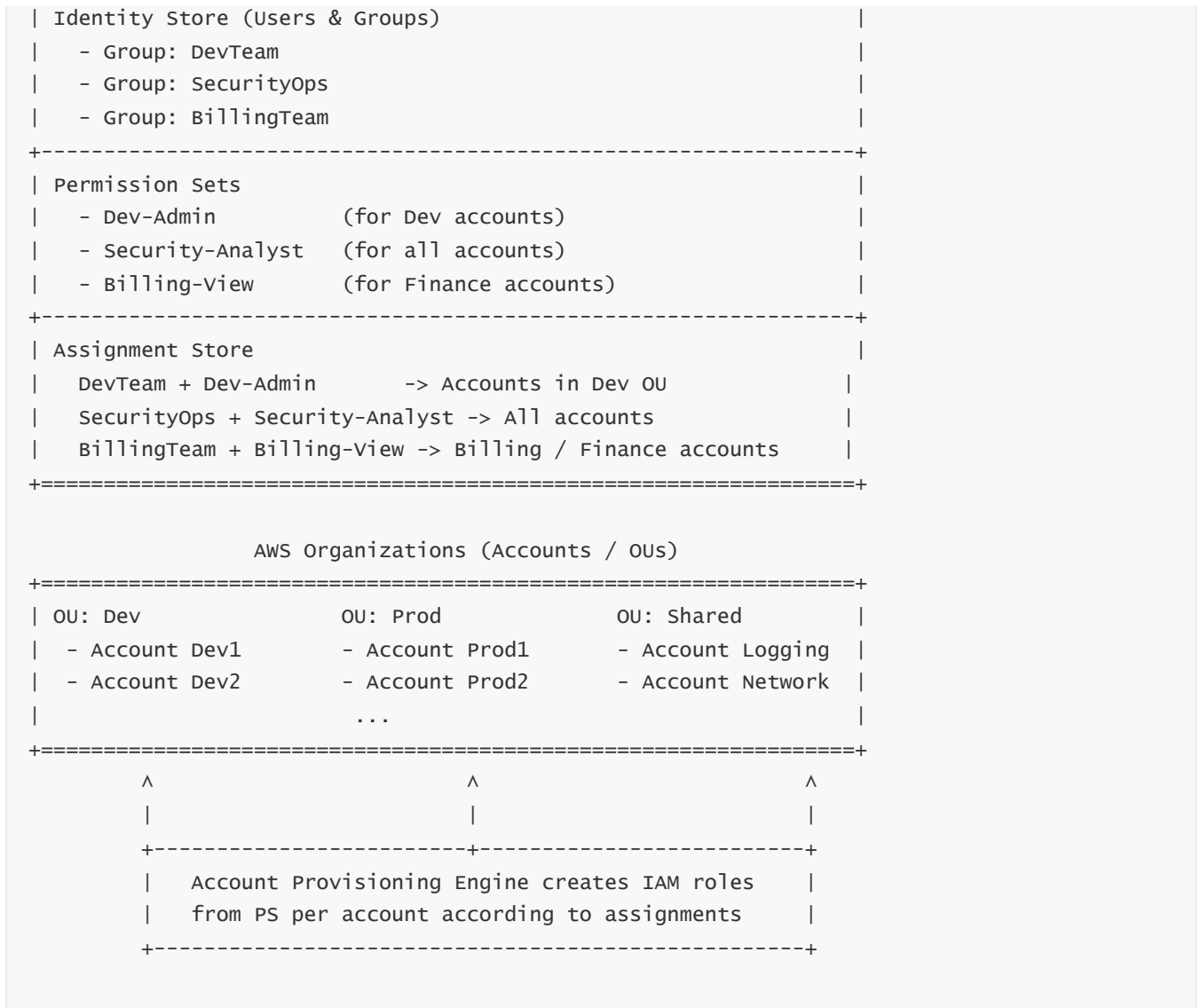
9.3 — “Central platform” accounts with selective assignments

- Some accounts (e.g., Shared Services, Logging, Networking) are more sensitive.
- Only Group `PlatformAdmins` gets admin-level permission sets in those accounts.
- Normal dev groups might get only `ReadOnly-Platform` permission set in those accounts.

The pipeline ensures the right groups see only the relevant tiles, and only the correct roles exist.

10 — ASCII diagram: multi-account strategy view





This diagram shows how, from the Identity Center perspective, you design:

- Groups (who).
- Permission sets (what).
- Org structure (where).

Then the assignment pipeline materializes that structure into real IAM roles and SSO experiences.

11 — Operational excellence and governance on top of the pipeline

To run this pipeline at scale, you typically add:

1. **Infrastructure-as-Code for assignments and permission sets**
 - Use CloudFormation, Terraform, or CDK to manage:
 - Permission set definitions.
 - Assignments (subjects + accounts).
 - This prevents “click-ops drift” and ensures repeatable configuration.

2. Reporting and reconciliation tooling

- Scripts or tools that call `sso-admin` and Organizations APIs to:
- List all permission sets and where they're used.
- List all assignments for a given group or account.
- Detect orphaned roles or assignments.

3. Change-management discipline

- Changes to widely used permission sets go through review.
- Migrations (e.g., changing identity source, renaming permission sets, reorganizing OUs) are planned with heavy care to avoid breaking assignments.

4. Lifecycle alignment with HR/IdP

- Joining a team: user gets added to group in corporate IdP/AD → SCIM updates Identity Store → pipeline gives them appropriate account tiles.
- Leaving a team: removing user from group automatically removes their access (no need to touch every account).

So the assignment pipeline becomes part of a bigger **identity lifecycle** system, not just a static config.

12 — Key pitfalls and how to avoid them

Some frequent pitfalls in the assignment pipeline:

1. Too many per-user assignments

- Leads to messy entitlement graphs and very hard offboarding.
- Avoid by using groups as the primary assignment target.

2. Unclear permission set naming

- If you name permission sets generically (e.g., `Role1`, `Role2`), admins and auditors cannot understand what they do.
- Use expressive names like `Prod-ReadOnly`, `Dev-Admin`, `Security-Analyst-Global`.

3. Ignoring propagation delays

- Expectation that assignments “work instantly” in 100+ accounts.
- Always acknowledge eventual consistency and plan for a short lag.

4. Underestimating blast radius of shared permission sets

- Changing a permission set used in many accounts could cause large permission changes everywhere.
- Use dedicated permission sets for notably different risk zones (Prod vs Dev; Highly Regulated vs non-regulated).

5. Manual roles in accounts not aligned with Identity Center

- Having random legacy IAM roles in accounts confuses users and ops.
 - Gradually migrate to Identity Center-managed roles, and clearly label or retire old ones.
-

13 — Final consolidated mental model

We can summarize Question 6 in one compact but deep mental model:

- **Assignments** are the **edges** between three sets: identities (users/groups), permission sets, and accounts.
- **The assignment pipeline** is the process that, given these edges, ensures that:
 - The right IAM roles exist in each account.
 - The right tiles appear in the portal.
 - The right STS sessions are created at runtime.
- **Multi-account access management** is simply the practice of designing groups, permission sets, and assignments so that you can **govern hundreds of accounts with a few clear patterns**, instead of managing roles one account at a time.

Once we internalize this, everything else in IAM Identity Center (SCIM, SAML/OIDC, permission sets, Organizations) becomes part of the same big picture: **a centralized, template-driven, assignment-based access fabric across all AWS accounts**.

8 — Federation Mechanisms: SAML Internals, Assertions, Tokens, Attributes (in IAM Identity Center)

1 — Where SAML actually sits in the Identity Center world

At a deep level, **SAML 2.0** in IAM Identity Center is used in **two main directions**:

1. **Inbound SAML**: From an **external IdP** → **IAM Identity Center**, so users can log into the AWS access portal with corporate credentials. Identity Center is the **SAML Service Provider (SP)** here. ([AWS Documentation](#))
2. **Outbound SAML**: From **IAM Identity Center** → **SAML-enabled applications** (AWS or third-party SaaS) when users click app tiles in the access portal. Identity Center is the **SAML Identity Provider (IdP)** to those apps. ([AWS Documentation](#))

SAML is **not** what Identity Center uses to get into AWS accounts themselves; that hop uses **STS + IAM roles** under the hood. SAML is about **federating identities** into Identity Center and out to apps, while **STS** is about **temporary credentials** into accounts. ([AWS Documentation](#))

2 — SAML assertion anatomy in the Identity Center context

A **SAML assertion** is an XML document that's signed by the IdP and consumed by the SP. In the Identity Center story we care about three types of statements, all present in the same assertion: ([AWS Documentation](#))

1. **Authentication statement** – proves that the user has successfully authenticated at the IdP, including:
 - When they authenticated (`AuthnInstant`).
 - How they authenticated (e.g., password, MFA, etc. via `AuthnContextClassRef`).
2. **Attribute statements** – key-value pairs describing the user: username, email, unique ID, groups,

department, etc. Identity Center uses these to map the assertion subject to a user in the Identity Store and (optionally) to drive ABAC / session tagging. ([AWS Documentation](#))

3. **Authorization decision context** – in classic IAM federation to roles, SAML can include `https://aws.amazon.com/SAML/Attributes/Role`, `RoleSessionName`, etc. In the **standard IAM SAML pattern**, these determine which IAM role STS will assume. ([AWS Documentation](#))

With IAM Identity Center as the SP from your corporate IdP, we usually **don't** send role ARNs directly in the assertion. Instead, Identity Center uses **user attributes** (like a stable ID, email, etc.) from the assertion to bind to a user that has assignments to permission sets and accounts. The role mapping is handled *inside* Identity Center, not by STS `AssumeRoleWithSAML` directly. ([AWS Documentation](#))

3 — Inbound federation: external IdP → Identity Center (SP)

When you configure Entra ID / Okta / Ping as your IdP and IAM Identity Center as SP, the flow is:

1. **User opens Identity Center portal URL** (SP-initiated) or clicks the AWS app tile in the IdP portal (IdP-initiated). ([Microsoft Learn](#))
2. Identity Center (SP) redirects to the IdP's SAML endpoint (if not already authenticated).
3. User authenticates at IdP (password, MFA, conditional access).
4. IdP issues a **SAML Response** containing:
 - A signed **SAML assertion** with authentication and attributes.
 - Destination = Identity Center ACS URL. ([AWS Documentation](#))
5. Browser posts the SAML Response to Identity Center's **ACS endpoint**.
6. Identity Center validates:
 - XML signature (against IdP's public cert).
 - Audience and recipient (is it really meant for this Identity Center instance?).
 - Timing (`NotBefore`, `NotOnOrAfter`). ([AWS Documentation](#))
7. Identity Center extracts the **subject** and attributes (like unique ID, email) and maps them to a user in its Identity Store (the user was usually provisioned earlier via SCIM). If mapping succeeds, a session is created and the user enters the AWS access portal. ([AWS Documentation](#))

Conceptually:

Corporate IdP does identity proof → SAML assertion → IAM Identity Center trusts it and turns it into a portal session tied to an internal user object.

4 — Outbound federation: Identity Center → SAML apps (IdP role)

When users click a **SAML 2.0 application** tile in the Identity Center portal:

1. The user is already authenticated to Identity Center and has an active session.
2. Identity Center builds a **SAML assertion** for the target app (Identity Center acts as IdP now).
3. That assertion includes:
 - Authentication context ("this user is authenticated").

- **Attributes**, based on **Attribute mappings** you configured on the app in Identity Center (e.g., `NameID`, `email`, `groups`, custom attributes). ([AWS Documentation](#))
4. Identity Center signs the assertion with its IdP certificate for SAML apps.
 5. Browser POSTs the SAML Response to the app's **ACS URL**.
 6. The app validates the assertion and creates its own session for the user.

AWS docs highlight that Identity Center **prefills common attributes** and you customize mappings per app to ensure the app receives exactly what it expects (e.g., “email” vs “User.PrincipalName”). ([AWS Documentation](#))

Here, SAML is being used as a **second-layer IdP**: your IdP federates into Identity Center; Identity Center then federates into downstream SAML apps.

5 — How SAML assertions link to the Identity Store and assignments

The critical bit for Identity Center is:

1. Subject binding

- In the inbound SAML assertion, a particular attribute (or combination) is configured to map to a user in Identity Center's Identity Store. That might be a unique user ID, UPN, or email, depending on your mapping strategy.
- The mapping is typically configured when you set up the SAML connection with Entra/Okta, etc. ([Microsoft Learn](#))

2. Attribute mapping and ABAC

- Identity Center can receive additional attributes from the SAML assertion and treat them as **ABAC attributes**.
- When configured, these attributes can be **propagated as session tags** into STS sessions for AWS account roles. AWS docs describe that if an attribute is defined in Identity Center as an “attribute for access control” and is received in SAML, Identity Center will **send that attribute as a session tag** on sign-in to AWS accounts. ([AWS Documentation](#))

3. Assignment resolution

- Once a user is mapped, Identity Center uses its Identity Store (which came from SCIM/IdP) and Assignment Store to see: which permission sets and accounts does this user (and their groups) have.
- The SAML assertion itself does **not** say “user gets role X in account Y.” Instead, Identity Center uses the assertion only to identify the user and read attributes; the role mapping happens internally via permission sets and assignments.

So the SAML assertion serves as the **bridge between IdP identity and Identity Center identity**, and optionally as a **carrier for ABAC attributes** that later influence IAM policies via session tags. ([AWS Documentation](#))

6 — How this differs from the classic “SAML → STS → role” pattern

In classic IAM SAML federation (without Identity Center), the flow is: IdP issues SAML assertion → STS

`AssumeRoleWithSAML` → user gets credentials for one of the roles named in

`https://aws.amazon.com/SAML/Attributes/Role` attributes. ([AWS Documentation](#))

Identity Center changes this model:

- You **federate once** to Identity Center as SP (only one SAML trust, one certificate to manage), rather than configuring SAML trust per AWS account. ([AWS Documentation](#))
- Identity Center then uses its own **managed IAM roles** and STS calls internally to provide access to any AWS account and permission set combination without the IdP having to know about each role ARN.
- The IdP's SAML assertion no longer needs to carry role ARNs; it just needs to identify the user and send any attributes you want for mapping or ABAC.

So in mental terms:

Classic: IdP decides roles and accounts directly in SAML.

Identity Center: IdP just proves identity; Identity Center decides roles and accounts based on assignments and permission sets.

7 — Attribute mappings, SAML attributes, and session tags

IAM Identity Center has a **SAML attribute mapping system** for outbound SAML apps and for ABAC into AWS accounts: ([AWS Documentation](#))

1. For SAML apps

- Each SAML 2.0 app configuration in Identity Center has an **Attribute mappings** section.
- Identity Center reads attributes from its Identity Store (which came from IdP/SCIM) – such as `userName`, `emails[primary]`, `displayName`, custom attributes – and maps them to SAML attributes in the assertion (e.g., `NameID`, `email`, `uid`, etc.). ([AWS Documentation](#))
- The app uses those attributes to identify and authorize the user.

2. For ABAC/session tags into AWS accounts

- Identity Center supports **Attributes for access control** (ABAC). You can configure attributes like `department`, `costCenter`, `project` etc. as ABAC attributes. ([AWS Documentation](#))
- If you use an external IdP, you can also **receive these attributes in the inbound SAML assertion**. Identity Center then stores them in the Identity Store. ([AWS Documentation](#))
- On sign-in to an AWS account, Identity Center can send those attribute values as **session tags** in the STS `AssumeRole` call (for example, `aws:PrincipalTag/department=Finance`). IAM policies in target accounts can then use `aws:PrincipalTag/*` keys for ABAC. ([AWS Documentation](#))

So SAML attributes and Identity Center's attribute mapping system become a **bridge between IdP user attributes and AWS ABAC** – both for SaaS apps and for AWS accounts.

8 — Detailed ASCII diagram: SAML flows around Identity Center



This diagram keeps the separation clear:

- Block A: IdP → Identity Center via SAML (SP role).

- Block B: Identity Center → Apps via SAML (IdP role).
 - Block C: Identity Center → AWS accounts via STS, not SAML.
-

9 — Typical SAML attributes used with AWS and Identity Center

Though Identity Center abstracts much of the low-level IAM SAML plumbing, you still meet some AWS canonical SAML attributes, especially when doing direct SAML federation or ABAC: ([AWS Documentation](#))

- `https://aws.amazon.com/SAML/Attributes/Role` – role ARN + SAML provider ARN (classic federation pattern).
- `https://aws.amazon.com/SAML/Attributes/RoleSessionName` – used to set session name.
- `https://aws.amazon.com/SAML/Attributes/PrincipalTag:*` – for directly tagging sessions with attributes using `AssumeRoleWithSAML`.
- `NameID` – often used as the primary user identifier (subject) for SPs.

In Identity Center:

- For inbound authentication, you mostly configure **which IdP attribute becomes the user’s “username” or stable ID**.
- For outbound SAML apps, you use Identity Center’s **attribute mappings UI** to specify how Identity Center attributes map to SAML attributes the app expects. ([AWS Documentation](#))

This mapping design is crucial for ensuring seamless SSO and correct user identity in target apps.

10 — Security properties and common mistakes with SAML in Identity Center

Because SAML is a security protocol, there are several important behaviors and frequent pitfalls:

1. Signature and certificate management

- Identity Center must trust the IdP’s signing certificate; if the IdP rotates certs without updating the SAML configuration on the Identity Center side, all SAML logins will start failing. ([AWS Documentation](#))
- On the outbound side, SAML apps must trust Identity Center’s IdP certificate; rotation requires updating metadata there.

2. Clock skew and assertion timing

- SAML assertions contain `NotBefore` and `NotOnOrAfter`. If clocks differ too much or you set very tight bounds, Identity Center may reject assertions as not yet valid or expired. ([AWS Documentation](#))

3. Audience/recipient mismatch

- The SAML assertion includes an `Audience` (SP identifier) that must match Identity Center’s expected entity ID. If misconfigured, Identity Center will reject the assertion as not intended for it. ([AWS Documentation](#))

4. Missing or inconsistent subject identifiers

- If the attribute used as the Identity Center user key (like email or object ID) changes or is not unique,

mapping to the Identity Store user can fail, breaking login. This is why using a **stable GUID-like attribute** is recommended. ([Microsoft Learn](#))

5. Over-loading SAML with role/authorization decisions

– In the Identity Center world, you should avoid pushing role ARNs in SAML, because Identity Center is already the central place deciding which roles/permission sets apply. Pushing role choices into SAML again introduces complexity and conflicts with Identity Center’s design.

6. Attribute bloat

– Sending too many or unnecessary attributes in SAML assertions can complicate mappings and ABAC. Keep attribute sets minimal and focused on what’s actually used.

11 — Consolidated mental model for SAML in IAM Identity Center

You can compress everything above into this mental model:

- **SAML 2.0** is the protocol that connects **your corporate IdP and SaaS apps** to IAM Identity Center and from Identity Center to apps.
- For **inbound**, Identity Center acts as SP: it accepts SAML assertions from your IdP, validates them, and turns them into Identity Center sessions tied to Identity Store users and attributes. ([AWS Documentation](#))
- For **outbound**, Identity Center acts as IdP: it issues SAML assertions to apps, populating attributes from its Identity Store according to attribute mappings. ([AWS Documentation](#))
- SAML attributes from the IdP can feed **ABAC attributes** that Identity Center passes as **session tags** into STS role sessions for AWS accounts, enabling attribute-based IAM policies. ([AWS Documentation](#))
- Role selection and AWS account access are no longer driven directly by the IdP’s SAML attributes; they’re driven by **permission sets and assignments** inside Identity Center, using STS behind the scenes.

If we keep that in mind, SAML becomes a clean part of the Identity Center story: it is the **federation fabric for identity and attributes**, while Identity Center + STS + IAM are the **access control and authorization fabric** across AWS accounts and applications.

9 — Federation Mechanisms: OIDC Internals, Tokens, Lifecycles, Device Auth (IAM Identity Center)

I’ll keep going in the same style as Q4/Q8 but now purely focused on **OIDC** and how Identity Center uses it.

1 — Where OIDC shows up in IAM Identity Center

In the Identity Center world, **OIDC (OpenID Connect)** is used in **two main ways**:

1. For AWS CLI / SDK “SSO” logins

- The AWS CLI v2 and some tools use an **OIDC-based device authorization flow** to obtain tokens from IAM Identity Center, which are then exchanged for STS credentials.
- This is the heart of `aws sso login` (now “AWS IAM Identity Center” login) behavior.

2. For application SSO (where supported)

- Identity Center can act as an **OIDC provider** (authorization server) for some apps, issuing **ID tokens** and **access tokens** when configured that way (similar to how it acts as a SAML IdP for SAML apps).

So mentally:

- **SAML**: mainly browser SSO between IdP ↔ Identity Center and Identity Center ↔ SAML apps.
- **OIDC**: mainly **non-browser / device / CLI** access to Identity Center, and some app SSO scenarios.

2 — OIDC basics in this context: ID token, access token, refresh token

Quick recap, but within the Identity Center context:

- **ID Token**
 - JWT that represents *who* the user is (subject, claims like email, name, etc.).
 - Consumed by the **client** (CLI, app) to know the identity.
 - Signed by the issuer (Identity Center OIDC endpoint).
- **Access Token**
 - JWT (or opaque token, but conceptually) that represents authorization to call specific APIs on behalf of the user.
 - For CLI, it allows the client to call Identity Center's token or SSO APIs to exchange for STS credentials.
- **Refresh Token** (where used)
 - Long-lived token that allows the client to obtain new access tokens without re-authenticating the user.
 - For AWS CLI's sso-session, the cached data effectively includes long-enough validity to avoid frequent re-login, though full refresh-token details are abstracted away by the CLI.

Identity Center exposes a set of **OIDC endpoints** (issuer, authorization, token, device authorization) that the CLI and other clients talk to; these are discovered/configured based on the region and the SSO start URL / instance.

3 — Device authorization flow (what `aws sso login` is actually doing)

When you run `aws sso login --profile X`, you're triggering the **OAuth 2.0 Device Authorization Grant** flow, layered with OIDC.

High-level steps:

1. **CLI starts device auth**
 - CLI calls the **device authorization endpoint** exposed by Identity Center's OIDC service.
 - It sends client information and the requested scopes (e.g., permission to get SSO tokens).
2. **Identity Center returns device and user codes**
 - Response includes:
 - **device_code** – secret used by CLI to poll.

- **user_code** – short code the user will type into a browser.
- **verification_uri** or **verification_uri_complete** – URL where the user must go and enter the code.
- **expires_in, interval** – how long the device code is valid and how often to poll.

3. User completes auth in browser

- CLI either opens the **verification_uri_complete** in the browser automatically, or prints a URL + user code for the user to copy.
- In the browser, the user logs into Identity Center (or external IdP) and approves the request for the CLI device/session.

4. CLI polls token endpoint

- Meanwhile, CLI repeatedly calls the **token endpoint** with the device_code at the interval indicated.
- Identity Center responds with “authorization_pending” until the user finishes login, or “slow_down” if CLI polls too aggressively.

5. Token issuance

- Once the user successfully authorizes, the token endpoint returns an **OIDC token set**:
- **id_token** (who the user is).
- **access_token** (what the CLI can do against Identity Center).
- **expires_in**, possibly a **refresh_token**-like mechanism behind the scenes in the SSO cache.

6. CLI stores tokens in local cache

- On disk in **~/.aws/sso/cache** (JSON files) plus profile references in **~/.aws/config**.
- These tokens are then used to request **SSO role credentials** for specific accounts + permission sets when commands run.

From the user’s point of view: “I log in once using a browser + CLI; after that, **aws** commands just work.” Underneath, OIDC device flow + Identity Center + STS is doing the heavy lifting.

4 — From OIDC tokens to STS credentials for a given account

Once the CLI has OIDC tokens for the Identity Center instance, it still needs **AWS credentials** for a specific account and permission set.

The steps look like this:

1. CLI reads the **profile** configuration:

```
[profile dev-admin]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = ap-south-1
sso_account_id = 111122223333
sso_role_name (or permission set name / friendly name)
```

1. Using the cached OIDC tokens, CLI calls **Identity Center’s SSO/OIDC APIs** to say:

“Give me AWS credentials for this user/session, for account 111122223333 using permission set X.”

1. Identity Center validates the token, checks the user's **assignments** (user or group) to verify that:
 - The user has an assignment to that `(permission set, account)` combination.
2. If authorized, Identity Center uses the **Account Provisioning mapping** to find the IAM role ARN in that account corresponding to the permission set.
3. Identity Center's SSO service internally calls **STS AssumeRole** for that role, using an internal service principal and adds session tags if ABAC attributes are configured.
4. STS returns **temporary AWS credentials** (AccessKeyId, SecretAccessKey, SessionToken, Expiration).
5. Identity Center wraps them in an SSO credential response; CLI receives them and uses them as standard AWS credentials for subsequent API calls.

So: **OIDC tokens** from Identity Center → authorized for SSO API → **STS credentials** for a particular role in a particular account.

The CLI only knows it called some "SSO"/OIDC endpoints; all STS and IAM details are hidden behind Identity Center.

5 — Lifetime & refresh behavior (tokens vs STS credentials)

There are multiple lifetimes to think about:

1. Device/OIDC login session (Identity Center)

- Controlled by Identity Center configuration and IdP login policies.
- If it expires, you must run `aws sso login` again.
- The CLI has logic to check token expiry and prompt re-login when needed.

2. OIDC access token lifetime

- Typically shorter-lived; CLI uses it to obtain STS credentials.
- CLI refreshes automatically if it still has a valid underlying SSO session / refresh mechanism and if the cached material allows it.

3. STS credential lifetime

- Defined by the **session duration** in the permission set (e.g., 1 hour, 4 hours, up to 12 hours depending on config).
- Once STS credentials expire, CLI has to ask Identity Center again (using OIDC tokens) to get new credentials.

You can think of it as a **2-level expiration**:

- Level 1: SSO session / OIDC token validity with Identity Center.
- Level 2: Per-account STS credential validity.

Security posture: even if the SSO session is long-ish, the **time window for each specific role** is bounded by STS session duration. For sensitive roles you can set shorter durations in the permission set.

6 — OIDC vs SAML in Identity Center: different channels

To keep distinctions sharp:

- **Inbound human login (browser)**
 - Often uses **SAML** from corporate IdP to Identity Center (SP).
 - This is the primary login to the portal and to start the CLI device flow.
- **CLI login and programmatic SSO**
 - Uses **OIDC device authorization flow** and token endpoints of Identity Center.
 - No role ARNs or SAML here; just tokens and SSO APIs.
- **Outbound to applications**
 - Can be **SAML** for SAML SPs.
 - Can be **OIDC** for OIDC clients/apps, where Identity Center acts as OIDC provider and issues ID/access tokens to those apps using standard flows (authorization code, etc., depending on integration).

And to AWS accounts themselves:

- **Identity Center** → **AWS Accounts** uses **STS AssumeRole** (neither SAML nor OIDC externally; purely AWS API).

So OIDC's main job is: **securely represent the user's Identity Center session to clients (CLI/apps) and let them obtain AWS credentials or app tokens.**

7 — OIDC claims and ABAC possibilities

Even though the CLI doesn't display them, **ID tokens** from Identity Center contain claims like:

- `iss` (issuer) – Identity Center OIDC issuer URL.
- `sub` (subject) – internal user ID.
- `aud` (audience) – the client / AWS SSO service.
- Standard claims like `iat`, `exp`.
- Possibly user identity attributes (name, email, etc.), depending on configuration.

In the **AWS accounts access path**, ABAC is actually driven by **session tags** set on STS sessions, which can originate from:

- Identity attributes in the Identity Store.
- Attributes sent via SAML and stored.
- Attributes configured as "Attributes for access control" in Identity Center.

Those attributes are **not** always directly read from OIDC tokens by IAM; instead, Identity Center takes known attributes about the user and tags the STS session accordingly. From an architectural view, OIDC's role is to **authenticate the CLI** to Identity Center; Identity Center then uses existing attributes to build the right STS session.

8 — ASCII diagram: OIDC device flow + STS combination

```
[1] CLI starts login
+-----+
| aws sso login      |
+-----+
      |
      | (A) Device Authorization Request
      v
+-----+ (Identity Center OIDC)
| OIDC Device Auth Endpoint|<-----+
+-----+
      | 2. device_code, user_code, verification URL |
      v
+-----+
| CLI prints URL/code |
+-----+
      |
      | 3. User opens browser, visits URL
      v
+-----+
| Identity Center Portal |
| (SAML login via external IdP)|
+-----+
      | 4. User authenticates
      | and approves device
      v
+-----+
| OIDC Token Endpoint   |<----- 5. CLI polls with
+-----+ device_code
      | 6. Return ID/Access token
      v
+-----+
| CLI SSO Cache (~/.aws)|
+-----+
      |
      | (B) Request AWS creds for acct+PS
      v
+-----+
| Identity Center SSO/API |
+-----+
      | 7. validate tokens
      | Check assignments
      | Find IAM role ARN
      v
+-----+
| STS AssumeRole        |-----+
+-----+
      |
      v
+-----+
| AWS Account |
```

```
| IAM Role |  
+-----+
```

```
[CLI uses STS creds for aws commands]
```

This shows:

- (A) OIDC device flow between CLI and Identity Center.
- (B) SSO API use of OIDC tokens to get STS credentials for specific roles.

9 — OIDC for app SSO (Identity Center as OIDC IdP)

While SAML is very common for SaaS, some apps support **OIDC**. When Identity Center is configured as an OIDC provider to those apps:

1. The app is registered as an **OIDC client** with Identity Center (client ID, redirect URIs).
2. Users authenticate to Identity Center (now or earlier).
3. App redirects the user to Identity Center's **OIDC authorization endpoint** with scopes, client ID, etc.
4. Identity Center prompts for login/consent as needed and issues **authorization code**.
5. App's backend exchanges that code at Identity Center's **token endpoint** for ID/access tokens.
6. App uses the ID/access token to identify the user and call its own backends or Identity Center APIs if needed.

This is symmetrical to SAML app integration, just using OIDC semantics and JWT tokens instead of SAML XML.

Useful when:

- App is modern and prefers OIDC/OAuth 2.0.
- You want a more API-friendly token model (JWT) rather than SAML.

10 — Security characteristics and common mistakes around OIDC in this setup

Some typical points to watch:

1. Local cache exposure

- CLI caches SSO/OIDC information and potentially STS credentials under the user's home directory.
- Protect that directory (file permissions, disk encryption), especially on shared machines.

2. Scope and least privilege

- Identity Center's OIDC flows should grant the minimal scopes needed for the client (CLI or app) to function. Identity Center's implementation is relatively constrained (it's not a general-purpose OAuth server), but principles still apply.

3. Session lifetime vs credential lifetime

- Long SSO / OIDC sessions combined with long STS durations can mean long-lived access if the device

is compromised. Balance user convenience with risk by choosing appropriate durations in permission sets and IdP policies.

4. Multiple profiles and sessions

– Users may have multiple profiles accessing different accounts or organizations. Be clear in profile naming and configuration to avoid confusion about which Identity Center instance / account you are logged into.

11 — Final mental model for OIDC in IAM Identity Center

Here's the compact mental picture:

- **OIDC is the token-based interface between non-browser clients (CLI/apps) and IAM Identity Center.**
- For CLI:
 - OIDC **device flow** gets you an SSO/OIDC token set.
 - Identity Center uses that to issue STS credentials for roles that permission sets + assignments allow.
- For apps:
 - Identity Center can act as an **OIDC IdP**, issuing ID and access tokens.
- SAML remains the primary protocol for **browser-based IdP** ↔ **Identity Center** login and SAML-based apps.
- STS + IAM roles remain the mechanism for **actual AWS account access**, with session durations and ABAC tags controlled by permission sets and attributes.

If we see OIDC as “the way CLI and some apps talk securely to Identity Center,” and SAML as “how the corporate IdP talks to Identity Center and how Identity Center talks to traditional enterprise apps,” the overall federation picture becomes very clean and understandable.

10 — CLI & Programmatic Access Through IAM Identity Center (Part A)

1 — Why the AWS CLI requires a completely different authentication channel than browsers

When humans access AWS through the browser, everything flows through:

- SAML identity federation (IdP → Identity Center).
- Identity Center portal session.
- Account tiles → STS AssumeRole behind the scenes.

But **CLI tools** cannot do browser redirection loops, cannot receive SAML POSTs, and cannot parse HTML forms. They need a protocol that is:

- 100% API-driven

- Non-interactive for the machine
- But still strongly tied to human identity (IAM Identity Center identity)
- Able to run on Linux terminals, CI machines, macOS, Windows, remote SSH sessions
- Able to renew credentials automatically
- And must follow OAuth security standards

OIDC Device Authorization Flow fulfills this requirement.

This is why:

The CLI uses OIDC (not SAML) to authenticate to Identity Center, but after authentication, everything still ends in STS AssumeRole into AWS accounts.

This is the foundation for everything else.

2 — The 4-phase lifecycle of CLI authentication through Identity Center

The entire CLI authentication lifecycle follows these four major phases:

Phase 1 — Authentication with Identity Center / OIDC (Device Flow)

This gives the CLI a long-lived “SSO session” for the human identity.

Phase 2 — Authorization to obtain AWS credentials (SSO API → STS)

For each account & permission set pair, Identity Center converts OIDC identity into STS credentials.

Phase 3 — Local caching and refresh logic

CLI caches Identity Center tokens + STS credentials so users don’t need to log in repeatedly.

Phase 4 — Command execution path

When running AWS CLI commands, the CLI resolves which Identity Center tokens and STS credentials to use and refreshes them when necessary.

These four phases form the basis of Identity Center’s programmatic access model.

3 — The Identity Center OIDC endpoints the CLI talks to

Every Identity Center instance exposes an OIDC authorization server with endpoints:

- `/oauth2/device_authorization`
- `/oauth2/token`
- `/oauth2/authorize` (for app flows)
- `/oauth2/jwks`
- `/oauth2/userinfo` (depending on flow)

The CLI specifically uses **Device Authorization** and **Token** endpoints.

All these endpoints live under the regional Identity Center domain, which depends on:

- Region (e.g., `ap-south-1`)
- Your Identity Center instance
- Your start URL (the “SSO start URL” configured in CLI profile)

These endpoints are **not public documentation of underlying URLs**, but AWS openly describes the OIDC standards that Identity Center follows.

4 — The detailed OIDC Device Authorization Flow for `aws sso login`

Below is the exact expanded internal flow.

Step 1 — CLI requests device authorization

CLI calls Identity Center:

```
POST /oauth2/device_authorization
client_id = <AWS CLI client>
scope = <sso:account-access>
```

Identity Center returns:

- `device_code`
- `user_code`
- `verification_uri`
- `verification_uri_complete`
- `expires_in`
- `interval`

Step 2 — CLI instructs user to open the URL

CLI often opens the verification URL automatically.

User authenticates through:

- External IdP (SAML or OIDC)
- Identity Center
- MFA
- Conditional Access
- Policies set in IdP or Identity Center

Step 3 — CLI polls `/oauth2/token` with `device_code`

Polling continues until one of these happens:

- User completes login
- Code expires
- CLI slows down based on `slow_down` responses
- Identity Center issues error token

Step 4 — Identity Center responds with tokens

CLI receives:

- **id_token** — the identity representation of the human user
- **access_token** — authorization to call Identity Center SSO APIs
- **expires_in** — expiration
- **refresh-like fields** (abstracted in AWS cache format)

The CLI stores all of this in `~/.aws/sso/cache/*.json`.

This completes OIDC authentication.

5 — The cached SSO session and the internal Identity Center “session token”

The AWS CLI does not expose raw OIDC tokens directly.

Instead, the CLI stores a **structured “sso-session” JSON file** containing:

- region
- start URL
- client ID
- client secret (if used)
- id_token
- access_token
- expiration timestamps
- some Identity Center metadata
- the user’s subject identifiers
- refresh expiration boundaries

This cache is the “SSO session.”

The important thing:

CLI does not store AWS access keys in the SSO cache.

It only stores Identity Center OIDC tokens.

AWS credentials (STS) are stored separately.

6 — How the CLI converts OIDC identity into AWS credentials

This is where Identity Center becomes the bridge:

Step 1 — User runs a CLI command

Example:

```
aws s3 ls --profile dev-admin
```

Step 2 — CLI loads the profile

CLI reads the AWS config:

```
[profile dev-admin]
sso_start_url = https://mycompany.awsapps.com/start
sso_region = ap-south-1
sso_account_id = 111122223333
sso_role_name = Dev-Admin
```

(Or the newer version with `sso-session`.)

Step 3 — CLI retrieves the cached OIDC tokens

From `~/.aws/sso/cache/xyz.json`.

Step 4 — CLI calls Identity Center's SSO/OIDC API

It calls something like:

```
GetRoleCredentials(account_id, role_name, access_token)
```

Identity Center checks:

- Does this `access_token` represent a valid human session?
- Does the user have assignments to (`account_id`, `role_name`)?
- What is the IAM role ARN for that permission set?

Step 5 — Identity Center calls STS AssumeRole on behalf of the user

Identity Center calls:

```
sts:AssumeRole
  RoleArn = arn:aws:iam::<account>:role/AWSReservedSSO_<PermissionSet>_<hash>
  RoleSessionName = <username or chosen default>
  Tags = <ABAC attributes>
```

Step 6 — STS returns temporary credentials

STS returns:

- AccessKeyId
- SecretAccessKey
- SessionToken
- Expiration

Identity Center wraps them.

Step 7 — CLI caches STS credentials in ~/.aws/cli/cache/

This is different from the SSO cache.

CLI now has working AWS credentials.

7 — Full visual: CLI → Identity Center → STS → AWS Account

```
User Terminal
|
| aws sso login
v
CLI OIDC Client
|
| Device flow to Identity Center OIDC
v
Identity Center (OIDC Auth)
|
| Creates Identity Center session tokens
v
CLI SSO Cache (~/.aws/sso/cache)
|
| aws s3 ls --profile dev-admin
v
Identity Center SSO API (GetRoleCredentials)
|
| Validates user identity + assignments
| Finds IAM role ARN for PermissionSet
v
STS AssumeRole
|
| Returns temporary AWS credentials
```

```

    v
CLI's Credentials Cache (~/.aws/cli/cache)
    |
    | Executes request with valid IAM role
    v
AWS API (e.g., S3 ListBuckets)
```

8 — Multi-profile and multi-account behavior

AWS Identity Center supports:

- Many profiles pointing to the same Identity Center instance.
- Many profiles pointing to different Identity Center instances.
- Many accounts/roles under one profile (interactive selection).
- Account/role selection via interactive prompt.

Examples:

Profile 1:

```
[profile dev]
sso_account_id = 1111
sso_role_name = Dev-Admin
```

Profile 2:

```
[profile qa]
sso_account_id = 2222
sso_role_name = QA-Engineer
```

If both use the same sso-session/start_url:

- One login covers both profiles.
- CLI will reuse the same OIDC tokens.

If different Identity Center instances:

- Separate logins & separate caches.

9 — Device auth lifetimes and why CLI forces re-login

CLI will trigger `aws sso login` again when:

- OIDC tokens expire beyond refresh window
- Device code grant expires
- Identity Center session expires

- IdP changes password/MFA/session policy
- Local SSO cache is deleted or corrupted
- Identity Center invalidates the session

This is not optional; Identity Center enforces OIDC rules.

10 — CLI's separation between Identity Center tokens and AWS STS credentials

Identity Center authentication tokens and STS tokens are **not interchangeable**:

Type	Purpose	Lifetime	Stored In
Identity Center OIDC tokens	Authenticate CLI → Identity Center	Medium	<code>~/.aws/sso/cache</code>
STS credentials	Access individual AWS account role	Short	<code>~/.aws/cli/cache</code>

Identity Center tokens stay valid much longer.

STS tokens expire frequently, forcing the CLI to fetch new ones, but without needing a new login unless OIDC tokens are expired.

10 — CLI & Programmatic Access Through Identity Center (Part B)

11 — Identity Center as a Programmatic Identity Provider (beyond the CLI)

Identity Center is **not just a human login system**; it is a *token authority* that machine-facing clients can use with strict boundaries.

But unlike IAM users or IAM roles, Identity Center is **not designed to issue long-term or automation credentials**.

This distinction is critical:

Identity Center = Human identity

Designed for:

- Developers
- Engineers
- Administrators
- Analysts

- SREs
- Security teams

The OIDC + STS bridge is intentionally designed so that:

- **Every AWS access session starts from a human identity,**
- **STS sessions are short-lived,**
- **Credentials cannot persist or be embedded in code,**
- **Refreshing requires Identity Center's permission and session validity,**
- **Audit trails always map back to a real person.**

Identity Center is NOT for automation

For CI/CD, pipelines, services, bots, scheduled tasks, etc., AWS mandates:

Use IAM Roles, Step Functions, Lambda IAM, or CI/CD integrations —
never Identity Center login automation.

Why:

- Device flow requires human action.
- Tokens expire with session policies.
- OIDC refresh is limited and bound to human IdP session lifetime.
- STS durations are capped per permission set.
- Device login cannot run unattended.

Identity Center's design enforces **Zero Standing Privilege (ZSP)** for humans and prevents misuse in automation.

12 — How software libraries (AWS SDKs) use Identity Center tokens

All AWS SDKs (Python boto3/botocore, Node, Go, Java, .NET) integrate with Identity Center via the **standard credential provider chain**.

The chain includes a provider called:

- **SSO Token Provider**
- Or **SsoOidcProvider** (in botocore internals)
- Or **SsoCredentialProvider**

These providers implement:

1. **Locate the profile** using `AWS_PROFILE` or config.
2. **Load the sso-session configuration.**
3. **Load the cached OIDC tokens** or perform a new login.
4. **Call Identity Center GetRoleCredentials API.**

5. Cache STS credentials.

6. Return the credentials to the SDK runtime for use.

SDKs do not call STS AssumeRole directly from the local machine; **Identity Center calls STS on behalf of the SDK**, ensuring:

- ABAC tags
- Permission set policies
- Assignment controls
- Unified identity
- One central audit trail

This means:

Programmatic Identity Center access through SDKs is nearly identical to CLI behavior — the SDK is simply an API-based client of the same SSO token provider.

13 — Using Identity Center in tools like Terraform, CDK, SAM, Serverless Framework

All modern tools that rely on AWS CLI/SDK credential providers now support Identity Center as a top-tier authentication source.

Terraform

- Uses AWS provider → AWS SDK → Identity Center credential provider.
- Requires the user to perform `aws sso login` manually before running `terraform plan/apply`.
- Once logged in, Terraform receives STS credentials automatically.
- **No service accounts** with Identity Center; always human.

CDK

- Same flow: `cdk synth`, `cdk deploy` → SDK → Identity Center provider → STS.
- CDK does not store or handle tokens by itself.

Serverless Framework

- Uses AWS SDK; configured profile works automatically.
- Requires `aws sso login` first.

AWS SAM

- Same behavior; Identity Center tokens serve as the auth source.

CI/CD pipeline anti-pattern

- **Never** embed `aws sso login` inside CI jobs.
- Device auth prompts cannot complete.
- You'll block pipelines on browser authentication.

CI/CD requires IAM roles and proper federation, *not* Identity Center.

14 — Multi-account scaling: thousands of accounts, thousands of roles, millions of API calls

In large AWS Organizations, Identity Center is used to authenticate:

- Hundreds or thousands of developers
- Tens of thousands of CLI sessions daily
- Across hundreds of accounts
- With dozens of permission sets
- And millions of STS AssumeRole calls yearly

Identity Center handles scale by:

Caching roles per (PermissionSet, Account)

Each permission set becomes **1 role per account**.

This results in:

- 100 accounts × 10 permission sets = 1,000 IAM roles
- Identity Center keeps a provisioning mapping in its internal system
- CLI does not need to know any of that; it only knows “role name” (PS name)

Distributed STS calls

Identity Center internally calls STS for every credential request.

STS scales regionally and horizontally.

Account throttling & retry logic

Provisioning engines handle:

- API throttling
- Retry strategies
- Eventual consistency
- Drift reconciliation

Programmatic access remains smooth due to SSO caches and STS caching.

15 — STS AssumeRole under Identity Center: deeper internals

Identity Center does not allow CLI clients to call:

```
AssumeRole
```

directly.

Instead:

1. CLI → Identity Center → GetRoleCredentials
2. Identity Center → STS → AssumeRole
3. STS → Identity Center → Temporary Credentials
4. Identity Center → CLI → Final credentials

Identity Center enforces:

- Which role ARN to assume
- Whether the user has permission
- Which session tags should apply
- Whether ABAC attributes are propagated
- Whether any constraints apply
- Session duration limit
- Trusted session sources
- PrincipalTag propagation

This ensures:

- STS cannot be randomly misused
- Custom role ARNs cannot be targeted
- Only roles created by permission sets are accessible
- The human identity is always included in every action

Identity Center fully mediates authorization and identity enforcement.

16 — ABAC in programmatic access (Identity → Attributes → Session Tags → IAM)

Identity Center can inject **session tags** into the STS credentials it issues.

These tags can come from:

- SCIM attributes
- SAML attributes

- Attribute mappings
- Custom attributes defined as “Attributes for access control”
- Group membership or IdP metadata (depending on mapping)

When STS session is created, Identity Center attaches:

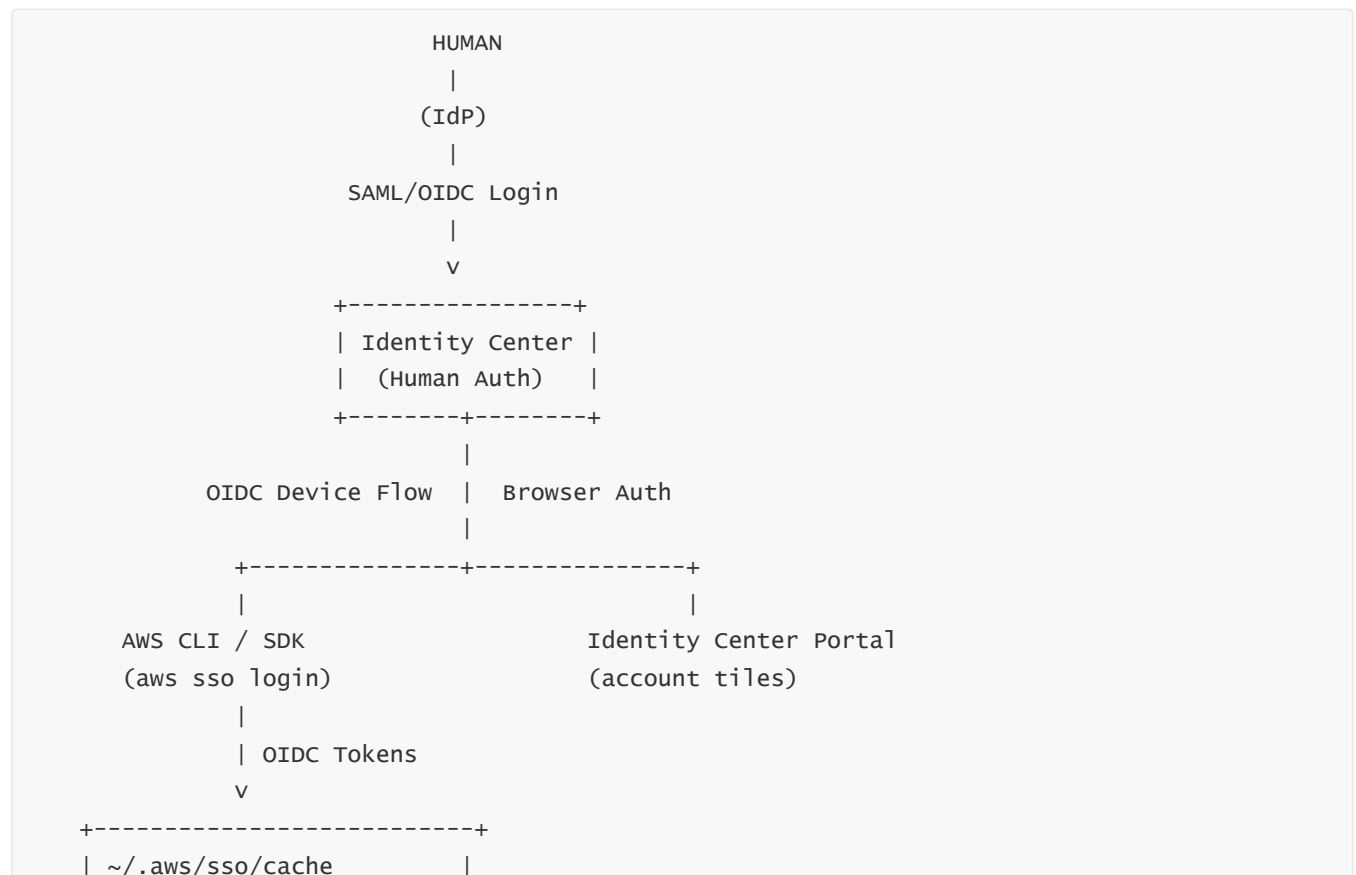
```
aws:PrincipalTag/<AttributeName> = <Value>
aws:userid = <IdentityStoreUserID>
aws:sso-session-id = <GUID>
aws:sso-user-id = <UserID>
aws:sso-group-memberships = <List>
```

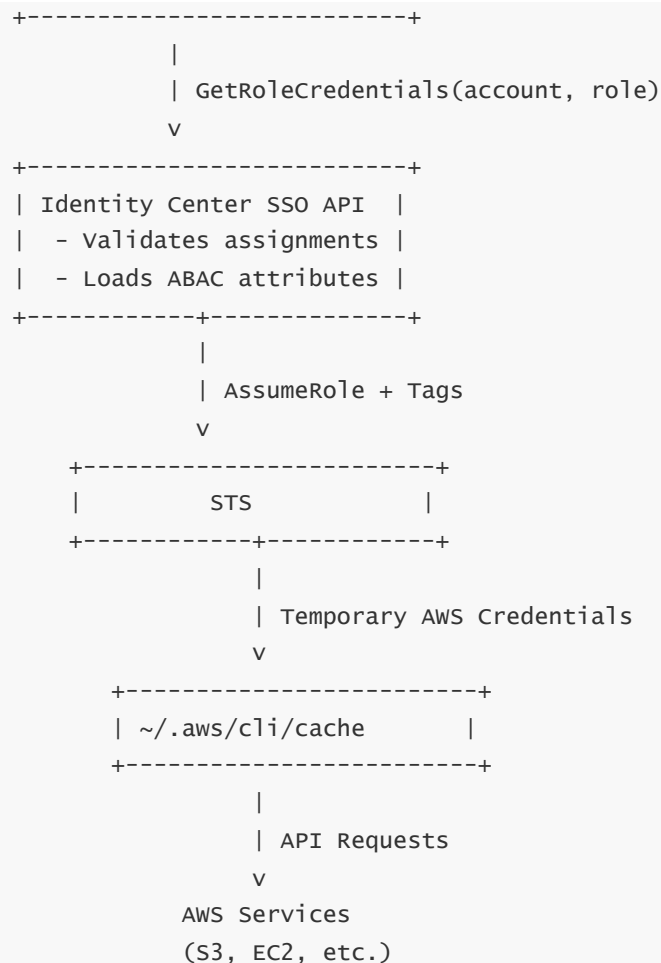
IAM policies in accounts can enforce:

- Project-based restrictions
- Department restrictions
- Cost-center boundaries
- Environment-based access
- Custom tags (customerID, region, tenant, etc.)

This creates a powerful, dynamic, centralized ABAC model.

17 — Diagram: Full CLI + SDK + ABAC Programmatic Access Architecture





18 — Advanced: Using Identity Center with ephemeral containers and ephemeral workspaces

In modern development setups:

- VSCode Remote Containers
- GitHub Codespaces
- JetBrains Remote Dev
- Cloud9
- Local containers / ephemeral shells

When running AWS CLI inside these environments:

Authentication Model

1. Developer runs `aws sso login` on local laptop browser
2. Identity Center tokens appear in `$HOME/.aws/sso/cache` locally
3. If dev container mounts host directories, CLI inside container uses the same cache
4. If not mounted, CLI inside container will request fresh login (browser from host opens login flow)

What NOT to do

- Copy OIDC tokens between environments
- Bake Identity Center credentials into container images
- Attempt to automate browser/device login in container startup scripts

What to do

- Use host pass-through directories
- Use volumes to persist AWS config and SSO cache
- Run login as needed manually
- Leverage VSCode/AWS Toolkit integrations that understand Identity Center

19 — CI/CD, automation, non-human workflows: the rules

This is **absolutely critical**:

***Identity Center is for humans.**

Automation must use IAM roles.**

Bad pattern:

- CI pipeline runs `aws sso login`
- Device flow cannot be completed
- Tokens expire
- Pipeline breaks
- Violates best practices & AWS security model

Correct pattern:

- CI pipeline assumes IAM role via OIDC or IAM role for service accounts
- No Identity Center login at all
- No device flow
- Direct IAM trust relationships

Examples of correct automation:

- GitHub Actions → AWS OIDC provider → IAM Role
- GitLab → IAM Role
- CodeBuild → IAM Role
- Jenkins with instance role → IAM Role
- EC2/Lambda role → IAM Role

Identity Center has no role in headless automation.

20 — Troubleshooting CLI + Identity Center Access

Common issues and their root causes:

1 — “SSO Token has expired”

- OIDC validity expired
- Need to re-run `aws sso login`

2 — “Permission denied” despite having access

Root causes:

- Assignment not propagated yet
- Wrong permission set selected in profile
- Using old `sso_role_name` syntax vs newer `sso-session` config
- Incorrect `sso_start_url` (most common issue)

3 — “Device authorization failed”

- Wrong time on local machine
- Browser blocked third-party cookies
- IdP session invalid
- Identity Center endpoint misconfigured

4 — “GetRoleCredentials AccessDenied”

- User has logged in successfully but does not have assignment to account+permission set
- IdP login succeeded but SCIM hasn’t provisioned user yet
- Assignment created but role not yet provisioned in target account

5 — STS session expires quickly

- Permission set session duration is too short
- Time mismatch (local clock skew)
- Cached credentials being overwritten by another profile

6 — “Unauthorized client” errors

- Wrong region in profile
 - Wrong client_id/client_secret (legacy behavior)
 - Wrong start URL
-

21 — Final Mega-Mental Model of CLI + Identity Center Programmatic Flow

One sentence summary:

Identity Center authenticates the human →
OIDC tokens authenticate the CLI/SDK →
Identity Center authorizes account/role access →
STS issues temporary AWS credentials →
CLI uses them →
Identity Center remains the single human identity authority.

Compressed mental picture:

```
Human Identity
  |
  v
Identity Center (SSO Session)
  | OIDC Tokens
  v
CLI / SDK
  | SSO API
  v
Identity Center (Authorization + ABAC Tags)
  | STS AssumeRole
  v
AWS Temporary Credentials
  |
  v
AWS API Calls (Everything Else)
```

This is the **complete logical model** behind every single AWS CLI command when using IAM Identity Center.

11 — Application Assignments and SSO Integration (Custom & SaaS Apps via IAM Identity Center)

1 — What “applications” mean in IAM Identity Center (vs AWS accounts)

In IAM Identity Center, you manage **two broad access surfaces**:

1. **AWS accounts / permission sets** – what we’ve already done in depth (multi-account access via IAM roles).
2. **Applications** – third-party SaaS, customer-built apps, and some AWS-managed applications that users access via SSO from the same portal. ([Amazon Web Services, Inc.](#))

When we say “applications” here, we mean the entries under **IAM Identity Center** → **Applications** in the console:

- **SaaS & AWS apps from the catalog** (Salesforce, Slack, Jira, Redis Cloud, WorkMail, etc.). ([AWS Documentation](#))
- **Customer managed apps** where *you* bring a SAML 2.0 or OAuth 2.0 (OIDC/JWT) application and configure it. ([AWS Documentation](#))

Key idea:

For AWS accounts we assign permission sets.

For applications we assign **application entries** (SAML or OAuth/OIDC) to **users/groups**.

Identity Center gives users one **access portal** where they see both:

- AWS account tiles (driven by permission sets).
- Application tiles (driven by app assignments).

2 — Types of applications Identity Center supports

Identity Center categorizes apps roughly into:

1. Catalog-based SAML 2.0 apps

- Pre-integrated SaaS templates (e.g., Redis Cloud, CyberArk, etc.). ([AWS Documentation](#))
- Pre-filled SAML settings; you just plug in metadata and adjust attribute mappings.

2. Customer-managed SAML 2.0 apps

- “I have an app I want to set up” (your own SP or a non-catalog SaaS).
- You configure SAML endpoints, ACS URL, audience/Entity ID, attribute mappings yourself. ([AWS Documentation](#))

3. OAuth 2.0 / OIDC / JWT-based apps

- Apps that speak **OAuth 2.0 + OpenID Connect** and accept **JWTs** (ID tokens / access tokens) from Identity Center.
- Often used with the **Trusted Identity Propagation** feature to let apps call AWS services on behalf of users using Identity Center-issued tokens. ([AWS Documentation](#))

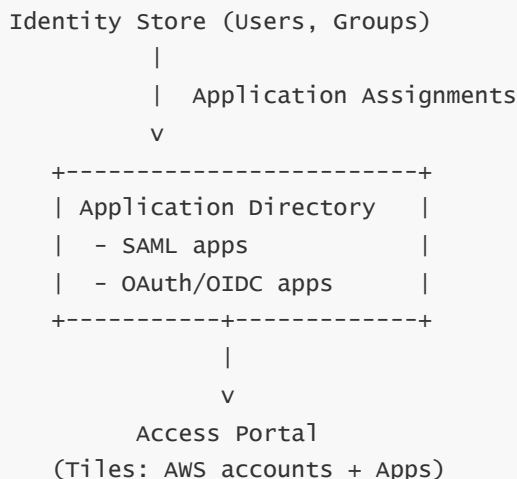
4. AWS-managed applications

- Some AWS services use Identity Center for workforce access (e.g., SageMaker Studio, Systems Manager Change Manager, IoT SiteWise, WorkMail integration, etc.). ([Amazon Web Services, Inc.](#))
- Access is configured from Identity Center or from the specific service console where you assign Identity Center users/groups.

All application types are ultimately **tiles in the portal** that use either SAML or OAuth/OIDC under the hood.

3 — High-level architecture: applications + assignments

At a conceptual level:



Internally, Identity Center tracks (for apps):

- **Application configuration** (SAML/OIDC settings, attribute mappings, session duration, start URL, relay state). ([AWS Documentation](#))
- **Assignments** of **users/groups** → **application**. ([AWS Documentation](#))

No IAM roles are created for these apps; instead, Identity Center acts as **IdP** emitting SAML assertions or OIDC tokens that the app trusts.

4 — How you add and configure an application (catalog and custom)

4.1 — Catalog SAML app (e.g., Redis Cloud, Okta-supported apps)

The typical flow (from AWS docs): ([AWS Documentation](#))

1. **Open IAM Identity Center console** → **Applications** → **Add application**.
2. Choose **"I want to select an application from the catalog"**.
3. Search for your app name (e.g., "Redis Cloud", "Slack", etc.), select it.
4. On the **Configure application page**:
 - Display name / Description pre-filled.
 - Download **Identity Center SAML metadata file** and **certificate** to configure in the app (SP).
5. Obtain SP metadata from the SaaS (ACS URL, Entity ID, etc.) and paste into Identity Center if required.
6. Optionally set **Application start URL, Relay state, Session duration**. ([AWS Documentation](#))
7. Save; now the app is configured but **no one has access yet** (no assignments). ([AWS Documentation](#))

4.2 — Customer-managed SAML 2.0 app

When the app is not in the catalog: ([AWS Documentation](#))

1. Applications → Customer managed → **Add application**.
2. Setup preference: **“I have an application I want to set up”**.
3. Application type: **SAML 2.0**.
4. Provide:
 - App display name.
 - SP Entity ID (audience).
 - ACS URL (SAML endpoint).
 - Optional RelayState, start URL.
5. Download IAM Identity Center SAML metadata & certificate; upload to your app's SP config.
6. Configure **SAML attribute mappings** in Identity Center so the app gets the correct attributes (e.g., NameID, email, roles, groups, custom claims). ([AWS Documentation](#))

4.3 — OAuth 2.0 / OIDC / JWT-based app

If your application speaks OAuth/OIDC and expects JWTs: ([AWS Documentation](#))

1. Configure it as an **OAuth 2.0/OIDC application** in the Customer-managed tab.
2. Register it as an OIDC client with Identity Center: client ID, redirect URIs, etc.
3. Configure **scopes & claims** you need in access/ID tokens.
4. Use Identity Center tokens for **trusted identity propagation** into AWS services or as identity for the app itself.

5 — Application assignments: users/groups → apps

Once an app is configured, you must **assign users or groups** to it (very similar conceptually to AWS account assignments, but no permission sets here).

From docs: ([AWS Documentation](#))

1. In Identity Center console → **Applications**.
2. Choose the application.
3. In **Assigned users** section choose **Assign users**.
4. Search by **user display name or group name**.
5. Select one or more users/groups.
6. Choose **Assign users**.

Key points:

- By default **no one** has access to a newly configured app. ([AWS Documentation](#))
- Assigning **groups** is the best practice for manageability (same as with accounts). ([Repost](#))
- Group membership (managed via SCIM/IdP) controls who effectively gets the app tile.

So the mapping is:

```
Group "SalesTeam" → CRM-Prod application
Group "EngTeam"   → Jira, Git provider, internal tools
Group "Finance"   → WorkMail, BI dashboards, billing apps
```

Users then see those tiles in their portal.

6 — Runtime SSO flows for applications

6.1 — SAML application flow

Once assignments exist:

1. User logs into Identity Center (SAML/OIDC in from corporate IdP).
2. Identity Center shows **application tiles** based on assignments.
3. User clicks a **SAML app tile**.
4. Identity Center constructs a **SAML assertion**: ([AWS Documentation](#))
 - Uses app's **SAML configuration**: ACS URL, audience.
 - Uses **attribute mappings** to populate user attributes.
 - Signs assertion with Identity Center IdP certificate.
5. Browser POSTs SAML Response to the app's ACS URL.
6. App validates signature, audience, expiry, and attributes; creates local session.

6.2 — OIDC application flow

For an OIDC app:

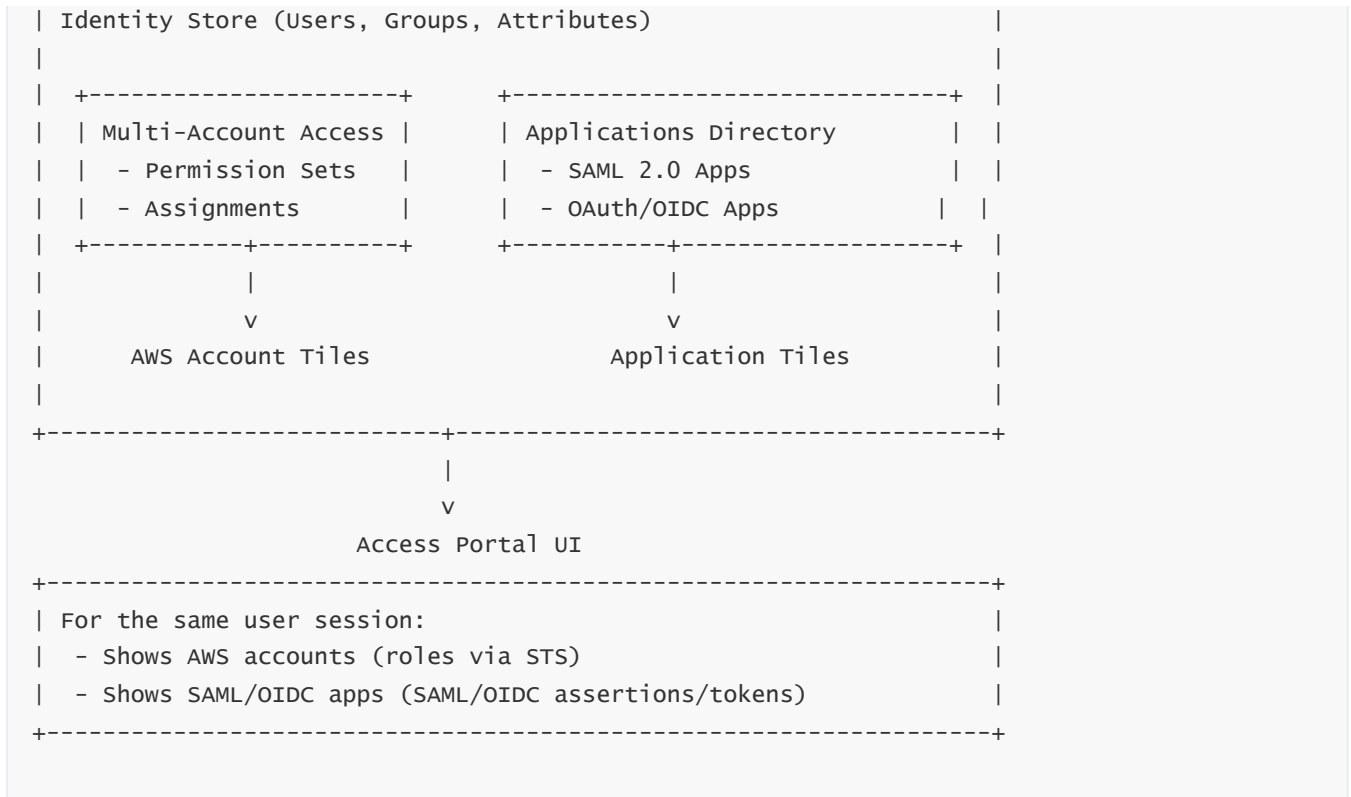
1. User either:
 - Starts at portal → click app tile → Identity Center does an **OIDC authorize** → **token** under the hood, or
 - Starts from app → redirected to Identity Center's OIDC authorize endpoint.
2. Identity Center authenticates user (if not already).
3. Issues **ID token + access token** to the app's backend at token endpoint. ([AWS Documentation](#))
4. App validates token signatures, issuer, audience, and uses claims to identify and authorize the user.

In both SAML and OIDC cases, Identity Center is the **IdP** for the app.

7 — ASCII architecture diagram: accounts vs apps in the portal

```
IAM Identity Center (Org Instance)
```

```
+-----+
```



- Left side: account access via permission sets & STS.
- Right side: applications via SAML/OIDC.
- Single Identity Store & session unify both.

8 — How AWS-managed apps consume Identity Center (WorkMail example)

Some AWS services integrate directly with Identity Center—for example **Amazon WorkMail**: ([AWS Documentation](#))

- In the **WorkMail console**, you can choose **Identity Center** as the identity provider and assign IAM Identity Center users and groups to WorkMail.
- That means:
 - Identity Center provides the **directory of users and groups**.
 - WorkMail uses assignments (users/groups) from Identity Center to control mailbox access.
- This is conceptually the same as a catalog application, but with tight AWS-native integration and UI in the service console.

Likewise, services like **SageMaker Studio**, **Systems Manager Change Manager**, **IoT SiteWise** integrate with Identity Center so you can manage workforce access centrally without wiring separate IdP connections for each AWS service. ([Amazon Web Services, Inc.](#))

9 — Trusted identity propagation (OAuth/JWT applications → AWS services)

For applications that use **JWTs / OAuth 2.0**, Identity Center supports **trusted identity propagation**: ([AWS Documentation](#))

Concept:

1. Your external OAuth IdP issues an identity token for the user (for your app).
2. Your app exchanges that token with Identity Center for an **Identity Center-issued token** that AWS services understand.
3. AWS services (like Analytics services) accept that Identity Center token to authorize data access “on behalf of” that user.

This is powerful for:

- BI tools or internal portals that need to query data in AWS (e.g., Lake Formation, Redshift, Athena) with user-level authorization.
- Ensuring **end-to-end identity context** flows from UI to compute layer to data services.

So Identity Center is not just a portal; it’s also a **broker of identity tokens** between external IdPs, your apps, and AWS services.

10 — Application assignments vs AWS account assignments: comparing patterns

Similarities:

- Both use **Identity Store users and groups**.
- Both are configured and visible in Identity Center console.
- Both appear as tiles in the access portal.
- Best practice in both: assign to **groups** instead of individual users. ([AWS Documentation](#))

Differences:

- **AWS accounts:**
 - Use **permission sets** → provision **IAM roles** in accounts.
 - Access is via STS AssumeRole.
 - Policies are IAM JSON documents.
- **Applications:**
 - No permission sets; instead, apps rely on **SAML attributes or OIDC claims** and the app’s own authorization.
 - Identity Center sends **SAML assertions/OIDC tokens** to the app; no AWS role created.
 - Fine-grained permission enforcement is in the **app**, not in IAM.

Design implication:

For AWS accounts, you model permissions in IAM (via permission sets & SCPs).

For SaaS/custom apps, you model permissions mostly inside the app, while Identity Center only controls **who can sign in**, and what attributes they receive.

11 — Best practices for designing application assignments

Based on AWS guidance and common patterns: ([AWS Documentation](#))

1. Group-first design

- Create groups in IdP like `Sales-Global`, `Engineering-Backend`, `Support-Level1`.
- Use SCIM to sync them into Identity Center.
- Assign apps to groups, not to individual users.
- Let group membership (HR-driven) control app access.

2. Environment separation with app instances

- For tools that have separate Dev/Staging/Prod tenants (e.g., Jira, Redis, etc.):
 - Create separate applications in Identity Center: `Jira-Dev`, `Jira-Prod`.
 - Assign relevant groups to each (e.g., `DevTeam` vs `ProdOncall`).
 - Use separate SAML/OIDC configs per environment.

3. Attribute minimalism

- Send only attributes that the app really needs.
- Use stable identifiers (GUIDs) as subject if supported.
- Avoid over-sharing PII in SAML/OIDC attributes.

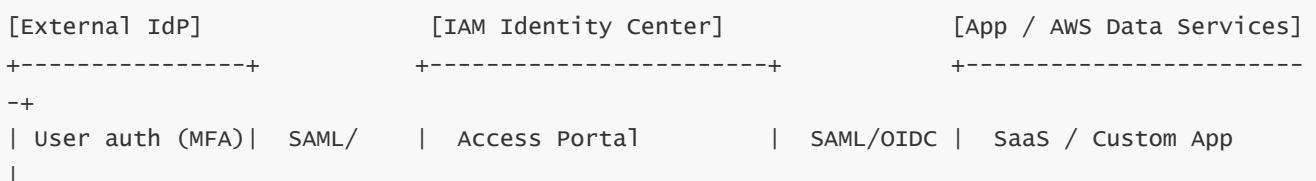
4. Parallel with AWS account access

- Mirror group semantics:
 - Group `DevTeam` gets Dev AWS accounts + Dev tools.
 - Group `SecOps` gets security tools + read roles in all accounts.
- This keeps mental model consistent for users.

5. Centralized catalog and governance

- Maintain a controlled set of apps; avoid random one-off SAML apps created by different admins.
- Document which group is supposed to have which apps and why.

12 — ASCII diagram: End-to-end app SSO and identity propagation



- Trying to use apps + Identity Center as generic OAuth server for services, instead of for workforce access.
- Fix: Use Identity Center for human/workforce; use IAM roles/service identity for machine-to-machine.

14 — Consolidated mental model for application assignments & SSO

Compact, “always remember” picture:

- IAM Identity Center has a **directory of users and groups** (from built-in, AD, or external IdP via SCIM). ([AWS Documentation](#))
- For **AWS accounts**, Identity Center uses **permission sets + role provisioning + STS**.
- For **applications**, Identity Center acts as an **IdP** using **SAML 2.0** or **OAuth 2.0/OIDC/JWT**, with **no IAM roles involved**. ([AWS Documentation](#))
- **Application assignments** connect users/groups to those apps, just like account assignments connect them to permission sets and accounts.
- The **Access Portal** is the unified UX where users see **both**: AWS accounts and apps.
- Trusted identity propagation extends this model so your apps can carry Identity Center-backed identity into AWS data services.

If we keep that in mind, application SSO through IAM Identity Center becomes just another dimension of the **same central identity fabric** we’ve built for AWS accounts.

12 — Cross-Account & Cross-Organization Access Patterns in IAM Identity Center

This question is extremely deep because it intersects:

- IAM Identity Center
- AWS Organizations
- Multi-account strategy & access fabrics
- Identity Store group design
- Permission set propagation
- Cross-Org boundary handling
- Centralized security/team access
- Shared services & hub/spoke models
- Federated identity designs
- Cross-Org tooling (Audit, Security, Platform teams)
- Strategic enterprise-level governance patterns

We will build a complete model of **how Identity Center enables access patterns that span tens, hundreds, or thousands of accounts — and, in advanced cases, even multiple Organizations**, including how to design them safely and efficiently.

1 — What “Cross-Account Access” really means in the Identity Center model

Cross-account access has a specific meaning:

A single human identity must be able to access many AWS accounts — with different levels of permissions — without needing IAM users or manual role management in each account.

In classic AWS IAM, this required:

- Account-local IAM users, or
- SAML per-account role mappings, or
- Custom role trust relationships per account, or
- Complex AssumeRole hops

Identity Center replaces all of that with:

- **Centralized identity fabric**
- **Permission sets** → IAM roles in accounts
- **Assignment pipeline** → mapping users/groups to account access
- **STS AssumeRole** automatically orchestrated by Identity Center
- **Unified human identity** recognized in all accounts

So “cross-account access” becomes:

One human identity → multiple AWS accounts → standardized permission set roles → all controlled centrally.

This is the base layer for everything that follows.

2 — Why cross-account design is mandatory in modern AWS Organizations

Enterprises rarely have only one account.

Typical structures:

- Landing Zone
- Dev / QA / Staging / Prod accounts
- Networking
- Shared Services
- Security Tooling
- Log Archive
- Data Lake
- BI / Analytics

- Multiple Business Units
- Multiple teams, products, workloads

Each account is a **security isolation boundary**, and having Identity Center orchestrate access across them avoids the security and operational chaos of old IAM-per-account models.

Identity Center enables:

- Central governance
- Central onboarding/offboarding
- Consistent least privilege
- Easy audit
- Enterprise-scale automation
- Clear organizational boundaries

This flows directly into cross-account patterns.

3 — The 3 building blocks of cross-account access in Identity Center

Identity Center builds cross-account access using three core objects:

3.1 — Identity Objects (users and groups)

Sourced from:

- SCIM from external IdP (Entra, Okta, Ping)
- AWS Directory Service AD
- Built-in identity store

These objects are the **who** dimension.

3.2 — Permission Sets

These define the **what** dimension:

- Permissions (IAM policies)
- Session duration
- Tagging/ABAC attributes
- Boundary permissions
- Advanced policy composition

Permission sets become IAM roles **per account**.

3.3 — Accounts (from AWS Organizations)

The **where** dimension:

- Identity Center lists accounts from AWS Organizations
- Assignments tie users/groups to specific accounts + permission sets

This creates thousands of possible “who → what → where” mappings in large environments.

4 — Full cross-account access lifecycle

Let’s construct the full lifecycle:

4.1 — Step 1: Group-level access modeling

Enterprise defines groups like:

- DevTeam
- DataEngineering
- NetworkOps
- SecurityAnalysts
- FinanceOps
- BreakGlassAdmins
- ReadOnly-All
- PlatformAdmins

These groups exist **only in IdP** and flow into Identity Center via SCIM.

4.2 — Step 2: Permission set design per persona

Enterprises define permission sets like:

- Dev-Admin
- Dev-ReadOnly
- Prod-ReadOnly
- Platform-NetworkOps
- Security-Incident-Response
- Billing-View
- BreakGlass

Each permission set defines the exact IAM role template.

4.3 — Step 3: Assignment across accounts

Identity Center assignments may look like:

DevTeam	→ Dev-Admin	→ Accounts: Dev1, Dev2, Dev3
DevTeam	→ Dev-ReadOnly	→ Accounts: SharedServices, Tools
SecurityAnalysts	→ Security-Incident-Resp	→ All accounts
FinanceOps	→ Billing-View	→ Billing, Finance, MasterPayer
BreakGlassAdmins	→ BreakGlass	→ All accounts

4.4 — Step 4: Provisioning into accounts

Identity Center creates IAM roles:

For `Dev-Admin`:

```
AWSReservedSSO_Dev-Admin_<hash>
```

Across Dev1, Dev2, Dev3.

For `Security-Incident-Resp`, Identity Center creates roles in **every** account.

This is huge:

- 100 accounts × 15 permission sets ≈ **1500 IAM roles**
- All consistently named
- All centrally managed
- No manual per-account IAM engineering

4.5 — Step 5: Runtime access

A user sees tiles:

```
Dev1 - Dev-Admin  
Dev2 - Dev-Admin  
Dev3 - Dev-Admin  
SharedServices - Dev-ReadOnly
```

Security Analysts see tiles for **every account**.

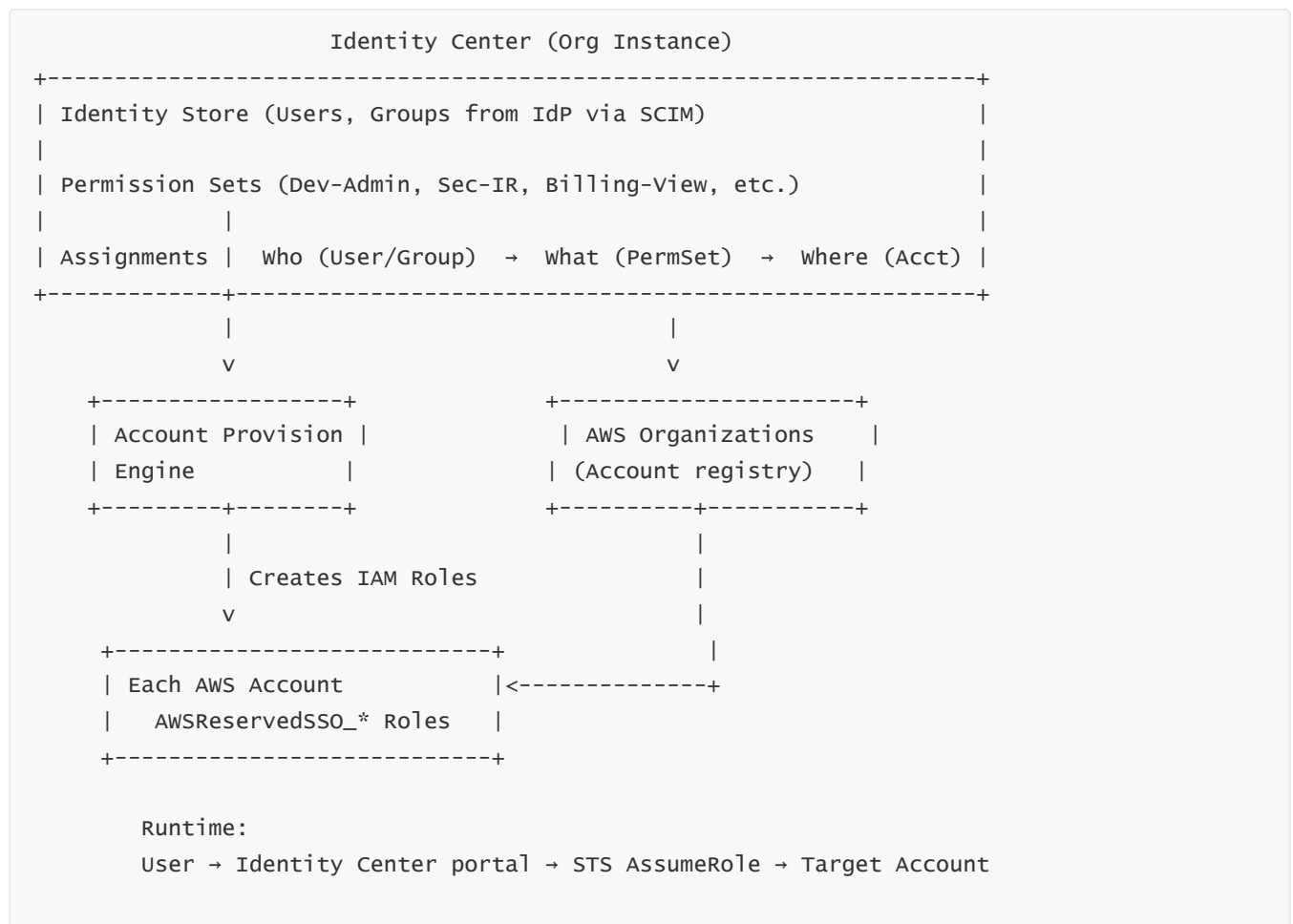
When clicked:

- Identity Center calls STS AssumeRole
- Inserts ABAC session tags
- Enforces session duration

- STS returns temporary credentials
- User enters specific account with correct permissions

This is the full cross-account access loop.

5 — ASCII architecture: cross-account access fabric



This diagram shows centralized identity controlling distributed account IAM roles.

6 — OU-Based Cross-Account Patterns (the most powerful design)

Using AWS Organizations OUs, you can create massive scalable access models.

Typical OUs:

```
Root
├─ Security
├─ Infrastructure
├─ Sandbox
├─ Dev
├─ QA
├─ Staging
└─ Prod
```

Pattern: Bind permission sets to entire OUs (via automation)

For example:

```
DevTeam      → Dev-Admin → All accounts in Dev OU
QAEngineers  → QA-Tester → All accounts in QA OU
ProdOps      → Prod-ReadOnly → All accounts in Prod OU
SecurityTeam → Sec-IR → All accounts (every OU)
```

As accounts are added, a Lambda/SFN automation:

- Detects new account creation via Organizations
- Automatically assigns groups → permission sets → accounts
- Triggers Identity Center provisioning

This gives auto-onboarding for every new account.

This is the most scalable model for enterprises.

7 — Cross-account patterns for central teams

Some teams need access to every AWS account:

7.1 — Security Team (Global Read or Global IR)

Assign:

```
SecurityTeam → Security-ReadOnly → ALL accounts
SecurityTeam → Incident-Response → ALL Prod/Staging accounts
```

They get unified visibility and emergency capabilities.

7.2 — Cloud Platform / Infrastructure Team

Assign:

```
PlatformTeam → Platform-Admin → SharedServices, Networking, Logging  
PlatformTeam → Platform-Observer → Dev/QA/Prod
```

Platform team can see everything but admin only specific infra accounts.

7.3 — Billing / Finance Team

Assign:

```
FinanceOps → Billing-View → Master Payer + Finance accounts
```

Not all accounts require finance access — only core ones.

7.4 — BreakGlass Admins (Emergency Access)

Assign:

```
BreakGlassAdmins → BreakGlass → All accounts (session duration short, MFA required)
```

Permission sets enforce:

- MFA
 - 1-hour session max
 - Logging
 - Elevated privileges only when required
-

8 — Multi-account ABAC across Identity Center

Identity Center supports session tagging through:

- SAML inbound attributes
- SCIM attributes
- Attributes for Access Control
- Custom properties
- Group membership metadata

When STS sessions are created, Identity Center injects:

```
aws:PrincipalTag/<attribute> = <value>
```

This enables ABAC:

Example:

```
Allow access to S3 buckets only if aws:PrincipalTag/project == "ProjectA"
```

This can unify:

- Cross-account access
- Multi-OU policies
- Developer workspace restrictions
- Environment separation (Dev vs Prod)

ABAC becomes a universal “identity fabric” that overlays thousands of accounts.

9 — Multi-Organization Access Patterns (advanced)

Identity Center normally operates **inside a single AWS Organization**.

However, there are advanced enterprise use-cases involving:

- **Multiple AWS Organizations** (Org A, Org B)
- **Central Shared Services**
- **Non-hierarchical org structures**
- **Subsidiaries or mergers**
- **Different businesses operated under different AWS Organizations**

9.1 — Identity Center cannot “natively” span Organizations

An Identity Center instance:

- Is tied to one Organization
- Provisions roles only in accounts under that Organization
- Cannot automatically manage roles in another Org

But cross-Org access patterns *are still possible*.

10 — How to build cross-Organization access patterns

10.1 — Pattern 1: “Guest Access” via IAM federation or OIDC

Org A's Identity Center users → access → Org B

Using:

- SAML federation from external corporate IdP (bypassing Identity Center for Org B), or
- OIDC provider model from Org A's Identity Center → Org B
- Custom IdP integration per AWS account of Org B

This bypasses Identity Center's limitation but loses the unified management.

10.2 — Pattern 2: Central IdP + Multi-Org Identity Center Instances

You can:

- Use the same IdP (Entra/Okta)
- Create **two** Identity Center instances (one per Org)
- Use SCIM to push the same user objects to both
- But assignments, permission sets, and roles are separate

Pros:

- Clear, independent governance per Org
- Shared IdP and identity lifecycle

Cons:

- Portal experience split (two access portals)
 - Users switch between Org1/Org2 portals
 - No unified cross-Org access experience
-

10.3 — Pattern 3: Shared Services Org

A common enterprise architecture:

```
Org-A (Shared Services)
Org-B (Product X)
Org-C (Product Y)
Org-D (Acquired Company)
```

Org A acts as:

- Identity Center authority

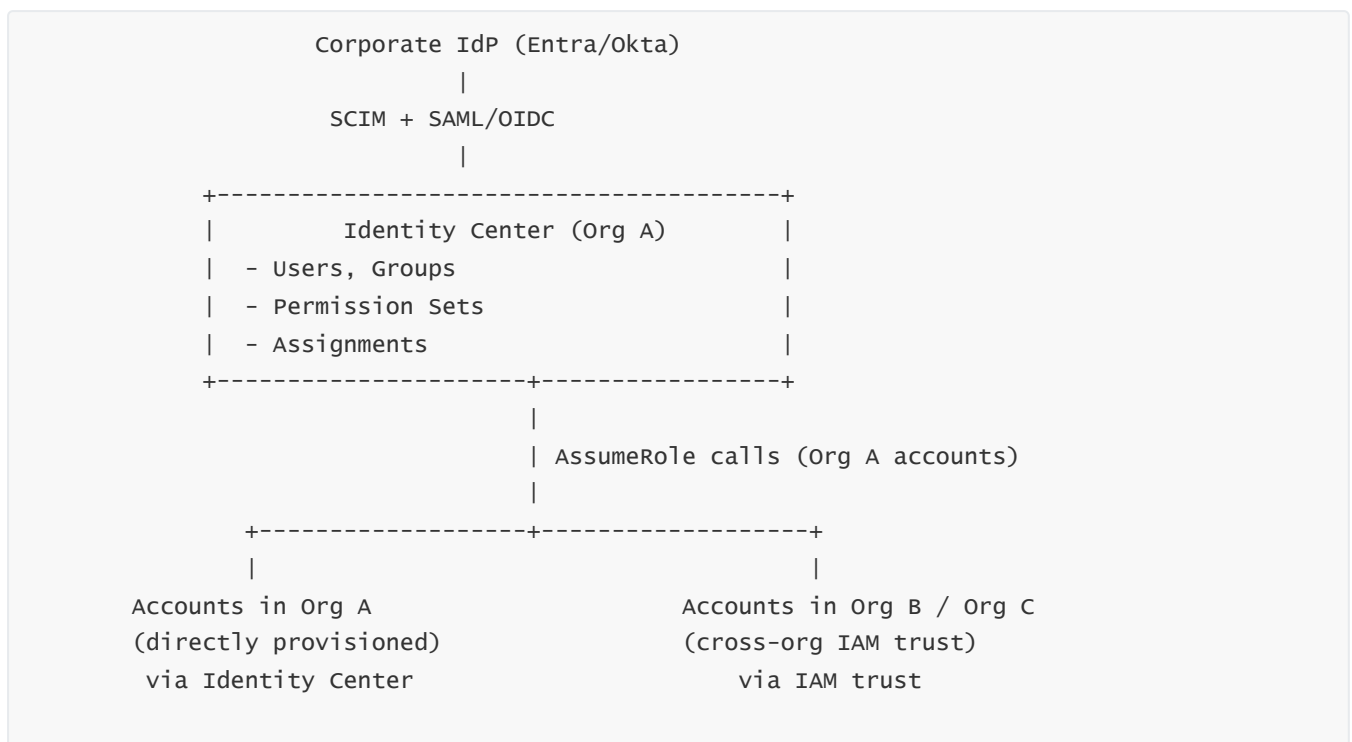
- Security tools owner
- Observability owner
- Network owner

Other Orgs integrate by:

- Creating trust relationships
- Using AWS RAM resource sharing
- Using cross-Org IAM Role trust back to Org-A accounts
- Possibly using App Fabric/OIDC token propagation

Identity Center does not directly grant access into other Organizations, but it **mediates identity** while cross-Org IAM roles mediate resource access.

11 — ASCII architecture: cross-Organization identity model



Identity Center controls identity centralization, while IAM trust controls cross-Org access.

This division preserves security boundaries.

12 — Ideal design for enterprises with multiple Organizations

The recommended model:

1 — Keep Identity Center per Organization

Each Org is a security boundary.

2 — Use Central IdP (SCIM + SAML)

All Orgs integrated with same IdP.

3 — For cross-Org access

Use:

- IAM roles with cross-Org trust
- AWS RAM for resource sharing
- Identity Center to authenticate users
- STS + IAM policies to authorize cross-Org access

Identity Center does not break security boundaries; IAM does the bridging.

13 — Cross-account access automation patterns

Large enterprises automate:

1. Account creation automation

- New accounts auto-detected by Control Tower or Organizations event
- Lambda auto-applies assignments based on OU
- Identity Center provisioning kicks in

2. Permission set versioning automation

- CI/CD pipeline for permission set templates
- Changes propagate to all accounts automatically

3. Assignment-as-code (IaC)

- Terraform/CDK pipelines manage:
 - Permission sets
 - Assignments
 - ABAC
 - SSO session durations
 - Reconciliation jobs

4. Account drift detection

- Detect orphaned roles
- Detect missing SSO roles in accounts
- Re-run provisioning engine

5. User lifecycle automation

- SCIM handles join/move/leave
- Access propagates instantly to dozens/hundreds of accounts

Automation ensures consistent, predictable cross-account access patterns.

14 — Major pitfalls and how to avoid them

Pitfall 1 — Per-user assignments

Avoid per-user assignments; always use groups.

Pitfall 2 — Too many permission sets

Leads to IAM role explosion.

Pitfall 3 — Overlapping roles in Prod/Non-Prod

Use OU boundaries and strict separation.

Pitfall 4 — Cross-Organization confusion

Assume Identity Center does not natively span Orgs; design accordingly.

Pitfall 5 — Manual IAM edits in accounts

Breaks Identity Center provisioning consistency.

Pitfall 6 — Long STS sessions for sensitive roles

Security best practice: limit high-privilege roles to short sessions (30–60 minutes).

15 — The ultimate mental model for cross-account & cross-Org access

Simplified but deep:

Identity → Groups → Permission Sets → Accounts → Roles → STS → Access

Across hundreds of accounts, this becomes a fabric:

- Identity Center holds the **identity graph**.
- Permission sets define the **access templates**.
- Assignments define the **access matrix**.
- The provisioning engine creates the **role mesh** across accounts.

- STS acts as the **credential factory**.
- AWS Organizations provides the **account topology**.
- IAM enforces the **authorization hooks** per account.
- ABAC tags unify identity with resource-level access.
- Automation keeps everything clean and synchronized.

Identity Center is the **brain**,

Organizations is the **body**,

STS is the **circulatory system**,

IAM roles are the **muscles**,

ABAC is the **nervous system**,

and your IdP is the **heart** that pumps identity into the system.

13 — Identity Center Security Best Practices, Hardening & Zero Trust (Part A)

Identity Center sits at the center of all AWS human access.

It determines *who* can enter AWS, *what* they can do, *where* they can go, *how* they authenticate, and *how long* their access persists.

Therefore, the security posture of Identity Center is not just important — it forms the **root of trust for the entire enterprise**.

Part A covers:

1. Security Foundations
2. Zero Trust Principles Applied to Identity Center
3. Identity Source Hardening
4. MFA Strategy & Enforcement
5. Session Policy Governance
6. Permission Set Security Hardening
7. ABAC Security Patterns
8. Diagrams

Part B will cover:

- Conditional Access
- IdP-side risk controls
- Breakglass models
- Administrator isolation
- Monitoring, logging, Incident Response

- SCIM hardening
- Token security & lifetime governance
- Advanced Zero Trust patterns
- Complete mega-diagram
- Ultimate best-practice checklist

1 — Why Identity Center security determines the security of every AWS account

Identity Center controls:

- Authentication → who enters AWS
- Authorization → which accounts/roles they assume
- Permission sets → what they can do inside AWS
- Session properties → how long and under what restrictions
- Attributes → ABAC controls across accounts
- Federation logic → how IdP credentials become AWS trust
- Cross-account & cross-Org privileges → global lateral movement boundaries

Therefore:

If Identity Center is compromised, all AWS accounts indirectly fall.

If Identity Center is hardened, every AWS account becomes proportionally safer.

Security best practices here must be higher than ANY single AWS account.

2 — Zero Trust Applied to IAM Identity Center

A Zero Trust model has three foundational ideas:

1. **Never trust identity implicitly (re-authenticate frequently).**
2. **Never trust network location (verify explicitly).**
3. **Least privilege enforced dynamically (ABAC, short sessions, conditional access).**

Identity Center expresses Zero Trust by enforcing:

- Short-lived **STS** sessions
- Mandatory **MFA**
- **Continuous identity verification** via IdP
- **Attribute-level authorization** (ABAC)
- Separation of **admin vs non-admin personas**
- **In-cloud** enforcement (no network perimeter)

A secure Identity Center deployment must:

- Treat every human access as a potential compromise point
- Limit all privileges using permission-set boundaries
- Force re-authentication for sensitive roles
- Ensure identity signals (attributes) are always verified and up to date

3 — Identity Source Security: the foundation of everything

Identity Center does **not** own user identities; it relies on an external source.

Three identity source modes exist:

1. **External IdP (SAML/OIDC) + SCIM**
2. **AWS Managed Microsoft AD**
3. **Built-in Identity Store**

3.1 — External IdP (Entra/Okta/Ping)

This is the most common enterprise model.

Security principles:

- Identity Center **trusts** SAML assertions or OIDC tokens from IdP.
- IdP must have:
 - MFA enforced
 - Conditional access
 - Device identity checks
 - Risk-based access
 - Password protection
 - Impossible-travel detection
 - Token replay protections

SCIM provisioning must be secured:

- SCIM bearer tokens must be rotated regularly
- SCIM access restricted on IdP side
- SCIM permissions scoped minimum
- No manual users created in Identity Center while SCIM is active
- IdP must be authoritative source of:
 - group membership
 - email
 - display name

- department
- manager
- attributes used for ABAC

If IdP is compromised → AWS is compromised.

Therefore IdP protections must exceed AWS account protections.

3.2 — AWS Managed Microsoft AD

Used when the enterprise identity is tied to:

- corporate AD forests
- on-prem directories
- hybrid cloud architectures

Security rules:

- Limit directory admin permissions
 - Protect AD sync connectors
 - Use AD conditional access where possible
 - Monitor AD replication & trust boundaries
 - Ensure AD group membership is tightly controlled
 - Avoid circular or overly broad AD group assignments for AWS access
-

3.3 — Built-in Identity Store

This is the simplest identity source but the **least enterprise-grade** one.

Security considerations:

- Use only for small startups or isolated workloads
 - Avoid using it for sensitive multi-account enterprise governance
 - Password and MFA policies must be hardened directly in Identity Center
-

4 — MFA: the single most important control in Identity Center

Identity Center authentication runs through IdP.

Therefore MFA must be enforced at IdP.

4.1 — MFA must be mandatory for all human identities

Policy:

- **MFA required for every login, every user**
- No bypass for any group/persona
- No legacy authentication allowed

MFA methods recommended:

- Strong phishing-resistant WebAuthn (FIDO2 keys)
- Entra Authenticator push with number matching
- Okta Verify with phishing protection
- Hardware tokens for breakglass

Avoid weak MFA:

- SMS
- Email OTP
- Lesser TOTP solutions

4.2 — Step-up MFA for privileged access

For roles like:

- Security Incident Response
- Breakglass Admin
- Network Admin
- Infrastructure Root
- Prod Admin

Force step-up MFA:

- User logs in normally
- When clicking high-privilege role tile → “MFA Required Again”

This can be achieved through IdP conditional access + permission-set session policies.

4.3 — Enforce strong token binding

Where supported, enforce:

- Device identity awareness
 - FIDO2 attestation
 - Anti-phishing capabilities
 - Managed device policies
-

5 — Session governance: controlling how long access lasts

Identity Center → STS → IAM session durations must be aligned with risk.

5.1 — Default recommended durations

- Dev roles: **4 hours**
- Read-only roles: **8–12 hours**
- Prod admin roles: **1–2 hours**
- Breakglass: **15–30 minutes**
- Security IR roles: **1 hour**

STS time should match the sensitivity of the permission set.

5.2 — Where session duration is set

- In the **permission set**
- As an explicit configuration, overriding IAM role default

5.3 — Why session duration governs blast radius

Identity Center cannot stop a compromised laptop from using STS credentials after login.

But **short sessions limit the damage window**.

5.4 — Enforcing fresh authentication for sensitive actions

This is done by:

- IdP conditional access (re-authentication every X hours)
- Identity Center permission sets with short STS sessions

6 — Hardening Permission Sets: least privilege, boundaries, and constraints

Permission sets must be treated as **privilege templates**.

Wrong model:

- “One admin permission set for everyone”

Correct model:

- Break permissions into granular persona-specific permission sets
- Enforce least privilege
- Minimize wildcards (“*”)

- Use permission boundaries if needed
- Keep separate permission sets for prod vs non-prod
- Avoid broad roles applied across all OUs

6.1 — Use multiple managed policies inside permission sets

Example:

- `AmazonS3ReadOnly`
- `AWSCloudTrail_ReadOnly`
- `CloudWatchLogsReadOnly`
- Custom policy: restrict to certain resource patterns

6.2 — No direct access to organization-wide admin roles

Don't give:

```
iam:*
sts:AssumeRole*
```

Unless extremely controlled.

6.3 — Inline policies should be audited regularly

Inline policies in permission sets must:

- Avoid wildcard actions
- Use conditional keys
- Use PrincipalTags
- Enforce least privilege

7 — ABAC Security Patterns: attributing identity to privilege

Identity Center attributes flow into:

- Permission-set session tags
- IAM policies that interpret these tags

7.1 — Secure ABAC attributes

Attributes must come from IdP, not local edits.

Good attributes:

- department

- team
- project
- location
- environment
- costCenter
- privilegeLevel

7.2 — Enforce ABAC consistency

IAM policies must trust only:

```
aws:PrincipalTag/<attribute>
aws:userid
aws:sso-user-id
aws:sso-session-id
```

Never trust:

- userName strings
- email addresses
- unverified custom attributes

7.3 — ABAC as a dynamic guardrail

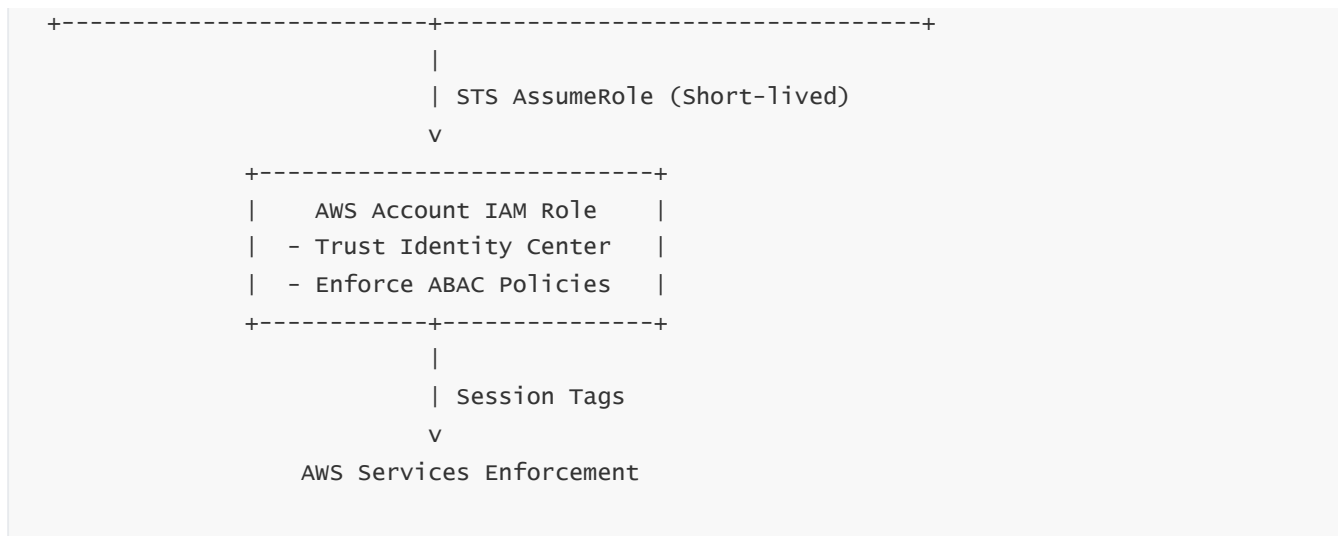
Examples:

- Developers only access resources tagged with `environment=dev`
- Data engineers only access data tagged with `project=X`
- Contractors only access specific application groups

ABAC permits horizontal scaling **without increasing number of permission sets**.

8 — ASCII Diagram: Zero Trust Access Path in Identity Center

```
[External IdP] <-- MFA, Conditional Access, Device Signals -->
    |
    | SAML/OIDC Authentication
    v
+-----+
|               IAM Identity Center               |
| - Identity Store (SCIM)                         |
| - Permission Sets (Least Privilege)              |
| - Session Duration Controls                      |
| - ABAC Attribute Injection                      |
+-----+
```



This represents the Zero Trust enforcement path:

- IdP handles *who*
- Identity Center handles *what* and *where*
- STS handles temporary trust
- IAM and resource policies enforce *how far*

13 — Identity Center Security Best Practices, Hardening & Zero Trust (Part B)

Part B covers:

1. Conditional Access & IdP-Side Risk Controls
2. Breakglass / Emergency Access Architecture
3. Admin Isolation & Privilege Separation
4. SCIM Security & Provisioning Hardening
5. Token Security Internals
6. SAML/OIDC Hardening
7. Logging, Audit, Monitoring & IR
8. Threat Modeling for Identity Center
9. Full Mega Security Diagram
10. Ultimate Security Checklist (Zero Trust Summary)

9 — Conditional Access & IdP-Side Risk Controls

Identity Center **delegates authentication** to the IdP.

This means:

All risk-based access decisions must be enforced at the IdP layer, before identity even reaches Identity Center.

9.1 — Conditional Access Controls at IdP

Common controls (Entra, Okta, Ping, etc.):

- **Re-authentication frequency rules** (hourly for high-risk personas)
- **Conditional access by device compliance** (require Intune/MDM)
- **Location-based restrictions** (approved geos only)
- **IP reputation checks**
- **Impossible travel detection**
- **Compromised credential detection**
- **Application-specific MFA rules** (strong MFA for AWS access)
- **Session risk calculation**
- **Behavioral anomaly detection**

Identity Center does not evaluate risk directly — it relies entirely on IdP for this.

9.2 — Identity Center as a relying party with strong authentication policy

Your IdP must enforce high-assurance authentication *specifically* for:

- AWS Console
- CLI SSO start URL
- High-privilege roles
- Sensitive SaaS apps integrated into Identity Center

A typical model:

```
If application == "AWS Identity Center" →  
    Require StrongMFA  
    Require device trust  
    Require compliant device
```

This ensures AWS is only accessible from trusted devices with strong MFA.

9.3 — Enforce fresh authentication for admin personas

Even if a user is logged in:

- High-privilege permission sets should trigger Repeat-MFA
- IdP enforces “Re-authentication for this app every X minutes”

- Short STS lifetime ensures AWS sessions cannot be abused indefinitely

This forms a **zero-trust circle** around sensitive access.

10 — Breakglass / Emergency Access Architecture

Breakglass = a tightly controlled emergency admin path when:

- IdP outage
- SCIM failure
- MDM failure
- Major incident
- Compromised identity
- Permissions misconfiguration

Breakglass access **must bypass Identity Center**, because relying on normal identity pipelines for emergencies violates the principle of resilience.

10.1 — Breakglass must not use Identity Center

Correct architectures:

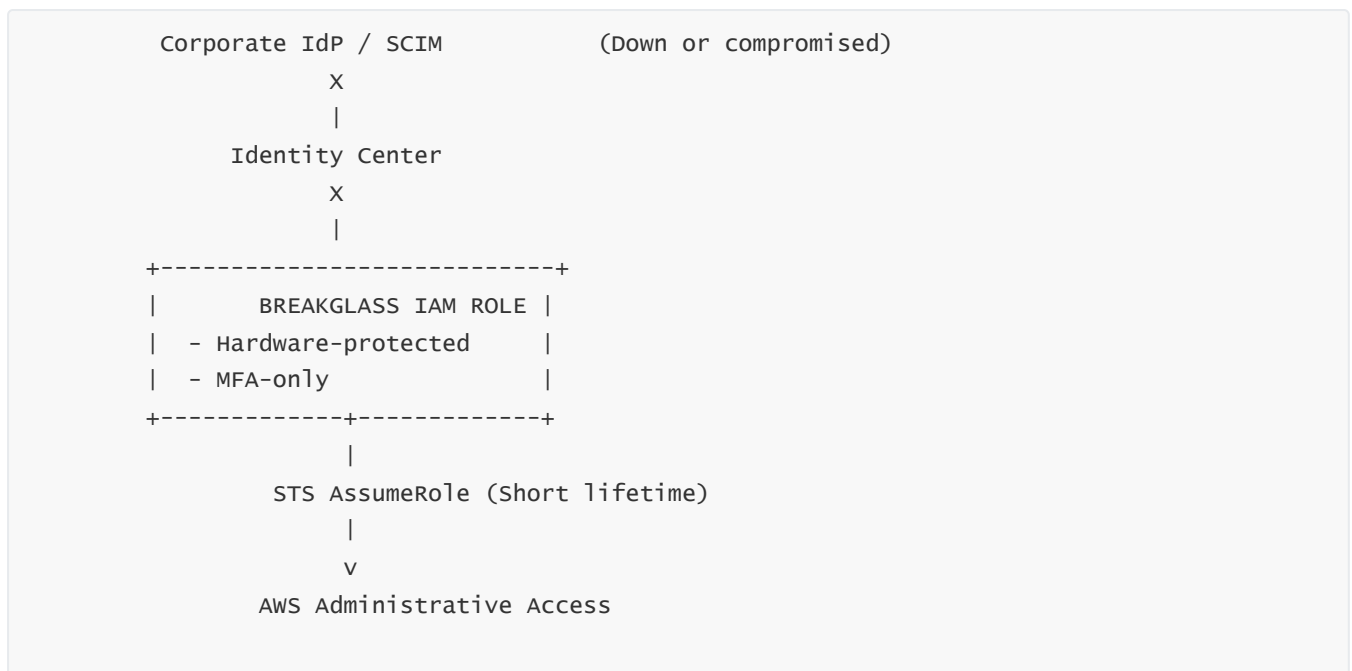
1. **Dedicated IAM role with MFA**
2. **Stored in a hardware password vault**
3. Rotation after each use
4. Logged and monitored
5. STS session duration extremely short (5–15 minutes)
6. Separate MDM device to access these credentials

10.2 — Do NOT:

- Do not store breakglass users in IdP
- Do not use Identity Center permission sets for breakglass
- Do not store breakglass credentials on normal user devices
- Do not issue long-lived keys for breakglass
- Do not rely on SCIM-created users for breakglass

Breakglass must be **IAM-native**, not IdP/Identity Center driven.

10.3 — Breakglass Access Flow Diagram



11 — Administrator Isolation & Privilege Separation

Admins are the most valuable targets.

Identity Center must enforce **strict isolation** between:

- Identity admins
- Application admins
- Cloud platform / infra admins
- Security admins
- Network admins
- Developers
- Support roles

11.1 — Use separate accounts for admin work

Admins should have:

- A normal user identity (daily tasks)
- A privileged identity (administration)

Identity Center permission sets should:

- Deliberately separate admin and user activities
- Require MFA step-up for admin roles
- Cap admin STS session lifetime (60 minutes max)

11.2 — Do not allow admins to disable logging

SCPs for admin persona must prevent:

- Disabling CloudTrail
- Deleting logs
- Disabling GuardDuty
- Disabling IAM Access Analyzer
- Modifying Identity Center configs without dual control

11.3 — Protect Admin console and Identity Center console access

Identity Center admins should have:

- High-assurance MFA
- Device binding
- Restricted network locations
- Monitoring of every action via CloudTrail

12 — SCIM Security & Provisioning Hardening

SCIM is the **identity ingestion pipeline**.

If SCIM is compromised → attacker can:

- Create users
- Add them to powerful groups
- Modify attributes (ABAC impact)
- Change group membership
- Cause privilege escalation undetected

12.1 — Protect SCIM bearer tokens

- Rotate every 30–60 days
- Store only in password vault or IdP secure config
- Restrict IdP admin access
- Enable API audit logs on IdP
- Monitor SCIM failure events (Identity Center Activity Log)

12.2 — Do not manually modify SCIM-provisioned objects

SCIM-managed users/groups are locked in Identity Center:

- You cannot change displayName
- Cannot change membership

- Cannot delete user
- Cannot update attributes

This protects IdP → AWS identity consistency.

If you must modify anything:

- Update it on IdP
 - Let SCIM push it into AWS
-

12.3 — Monitor SCIM drift

If SCIM is failing:

- Some new users will not gain access
- Some removed users will still have AWS access
- Some changed attributes will not sync → ABAC drift

You must alert on:

- SCIM HTTP failures
 - SCIM throttles
 - SCIM bearer token expiration
 - Identity Center error logs about provisioning
-

13 — Token Security: OIDC, SAML, STS

Identity Center uses:

- **SAML** (IdP → Identity Center → SaaS apps)
- **OIDC** (CLI login & some apps)
- **STS** (access to AWS accounts)

13.1 — OIDC token hardening

Identity Center OIDC tokens must be:

- Stored only in `~/.aws/sso/cache`
- Protected by OS permissions
- Never stored in repos, S3, shared machines
- Never used as automation credentials
- Never copied between machines
- Cleared regularly

STS credentials from Identity Center:

- Should be short-lived

- Must not be baked into scripts
 - Must never be exported in environment variables on multi-user hosts
-

13.2 — SAML hardening

SAML assertions must:

- Have short validity
- Use strong signing certs
- Rotate certificates regularly
- Validate audience correctly
- Restrict allowed ACS endpoints
- Not carry excessive PII

IdP must enforce:

- Signed assertions
 - Strong encryption
 - Replay protection
 - Assertion lifespan enforcement
-

13.3 — STS hardening

Configure:

- Short session durations for sensitive roles
 - Use externalId conditions for cross-org roles
 - Block double AssumeRole chains
 - Deny wildcard sts:AssumeRole* for non-admin roles
 - Apply session policies where needed
-

14 — Logging, Audit, Monitoring & Incident Response

Identity Center emits logs through:

- **AWS CloudTrail**
- **AWS Identity Center logs**
- **AWS SSO portal logs**
- **STS logs**
- **IAM Access Analyzer events**
- **SAML/OIDC authentication logs (via IdP)**

14.1 — Monitor user behavior

Detect:

- Unusual account access
- Strange region patterns
- Rapid privilege escalation
- Multiple failed logins
- Unusual IP or device
- Impossible travel
- Access to accounts never used before
- Sudden use of high-privilege roles

14.2 — Audit permission sets

Audit at least monthly:

- Which permission sets exist
- What IAM policies they contain
- Which accounts they are applied to
- Which groups/users are assigned

14.3 — Monitor all IAM role provisioning changes

Identity Center provisioning may:

- Succeed
- Fail
- Remediate drift
- Create roles
- Update policies
- Delete obsolete roles

All of these should be logged and alerted.

15 — Threat Modelling for Identity Center

Attackers target:

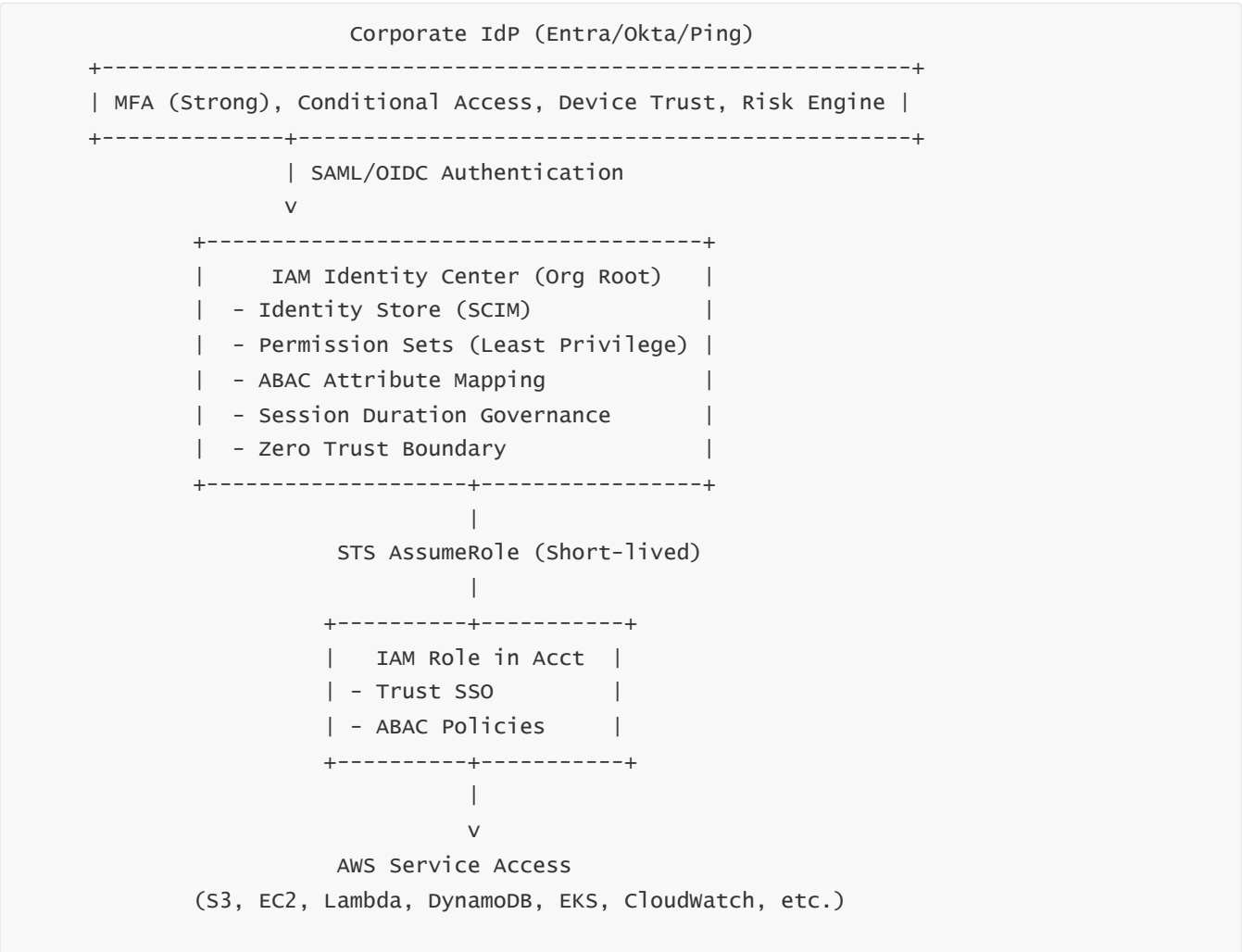
1. **IdP compromise** (primary target)
2. **SCIM token theft**
3. **Local laptop with cached OIDC tokens**
4. **Excessive privilege in permission sets**
5. **Lateral-movement across accounts**

- 6. Breakglass misuse
- 7. Stolen STS tokens
- 8. Too-long session durations
- 9. Misconfigured SAML integrations
- 10. Weak MFA configuration

Identity Center must be designed to resist:

- Credential theft
- Session hijacking
- Privilege escalation
- Unauthorized provisioning
- Compromised user devices
- Insider attacks
- Supply-chain attacks via IdP or SCIM

16 — Full Mega Security Diagram (Zero Trust End-to-End)



Monitoring & Logs:

- CloudTrail (All Events)
- IAM Access Analyzer
- IdP Auth Logs
- SCIM Logs
- Identity Center Activity Logs

Controls:

- SCPs
- Per-OU Boundaries
- Admin Isolation
- Breakglass IAM Role

17 — Final Ultimate Security Checklist (Zero Trust Identity Center)

Identity Source

- Use external IdP
- Enforce Strong MFA
- Enforce conditional access
- Enforce device trust
- Monitor authentication patterns

SCIM

- Rotate SCIM tokens
- Protect SCIM endpoints
- Treat IdP as single source of truth
- No manual changes in Identity Center

Permission Sets

- Least privilege
- Separate prod and non-prod
- Inline policies audited regularly
- Avoid wildcard IAM actions
- No direct “admin everywhere” roles

STS

- Short session durations for sensitive roles
- Step-up MFA enforced via IdP
- Prohibit AssumeRole chaining

ABAC

- Use SCIM attributes
- Use PrincipalTags for dynamic enforcement
- Validate attributes against IdP policies

Access Patterns

- No per-user assignments
- Group-based access only
- OU-based assignment automation
- Tiered admin access

Breakglass

- Must NOT rely on Identity Center
- Stored offline
- MFA protected
- Audited after each use
- Rotated after use

Monitoring

- CloudTrail everywhere
- Identity anomalies from IdP
- Detect lateral movement attempts
- Detect unusual privilege escalation

Hardening

- Lock administrative access
 - Implement least privilege for Identity Center admins
 - Enforce short-lived tokens
 - Protect local token caches
 - Audit application SAML/OIDC configurations
-

14 — Auditing, Monitoring, Logging & Observability in IAM Identity Center

Auditing and monitoring IAM Identity Center is fundamentally different from auditing classic IAM users because Identity Center does not produce long-lived credentials, does not support access keys, and does not maintain traditional IAM identities inside AWS accounts.

Instead, **Identity Center unifies all identity observability into a multi-layered trail across IdP, Identity Center, STS, IAM, CloudTrail, and account-level services.**

In this question, we will construct the complete monitoring pipeline needed for enterprise-grade visibility.

1 — Why Identity Center Observability is a Multi-Layer System

Identity Center itself only controls:

- Authentication (delegated)
- Authorization (assignments, permission sets)
- Session orchestration (STS from Identity Center)

However, IAM enforcement happens **inside each AWS account**.

The IdP enforces **authentication policies**.

CloudTrail enforces **AWS event logging**.

STS enforces **credential issuance**.

This results in **5 independent but linked observability layers**:

1. **IdP authentication logs**
2. **Identity Center logs (admin + access logs)**
3. **STS logs**
4. **IAM + resource-level CloudTrail logs**
5. **AWS account-level service logs**

Together, they create a full end-to-end audit chain.

2 — The 5-Layer Audit Model

```
Layer 1: IdP Authentication (Entra/Okta/Ping)
Layer 2: Identity Center Access & Admin Activity
Layer 3: STS Credential Issuance Logs
Layer 4: CloudTrail IAM & AWS API Events (per account)
Layer 5: Resource-level logs (S3, EKS, Lambda, Cloudwatch, etc.)
```

This model ensures:

- **Identity actions** (who logged in) are recorded at IdP
- **Authorization actions** (who got access via permission set) are recorded at Identity Center
- **Credential issuance** recorded by STS
- **API behavior** recorded in CloudTrail
- **Data layer actions** recorded in resource logs

Because Identity Center centralizes identity, but AWS accounts decentralize action, monitoring must span *all* layers.

3 — Layer 1: IdP Authentication Logs

Identity Center does NOT authenticate users directly when using an external IdP.

Therefore identity security visibility begins at IdP logs.

IdP logs provide:

- Login attempts
- MFA success/failure
- Impossible travel detection
- Sign-in risk scoring
- Device compliance verification
- Conditional access evaluation
- Token issuance history
- SSO app access logs (Identity Center is seen as an “enterprise app”)

These logs answer:

- *Who authenticated?*
- *From where?*
- *On which device?*
- *Was MFA used?*
- *Was the login high-risk?*

These logs must feed:

- SIEM
- UEBA (User Behavior Analytics)
- Threat detection tools

Identity Center relies implicitly on IdP’s correctness — so IdP logs are *security-critical*.

4 — Layer 2: Identity Center Activity Logs

AWS Identity Center generates activity logs that capture:

1. Administrative activity

- Permission set creation or modification
- Assignment creation/removal
- Application setup changes
- SCIM provisioning events
- Identity source configuration changes
- Account provisioning changes

2. Access activity

- User sign-ins to the AWS portal
- User SSO role selection events
- Permission propagation attempts
- Provisioning successes/failures
- Token issuance for CLI sessions

Identity Center logs show:

- *Which user was assigned access?*
- *Which admin modified a permission set?*
- *Did a user attempt to access an account they weren't assigned to?*
- *Did provisioning succeed across all accounts?*
- *Did SCIM update a user or group?*

Identity Center logs must be:

- Enabled to CloudWatch Logs or S3
- Forwarded to SIEM
- Monitored for anomalies
- Retained for long-term compliance

5 — Layer 3: STS Credential Issuance Logging

STS is ALWAYS involved in Identity Center access.

Identity Center federates into STS for every:

- AWS console tile click
- CLI `GetRoleCredentials` call
- SDK-initiated credential refresh

STS logs record:

- `AssumeRole` events
- PrincipalTag injection
- Session durations
- Role ARN assumed
- Source identity (`sso-user-id`, `sso-session-id`)
- MFA context (if enforced via IdP)

STs logs answer:

- *Which Identity Center user obtained AWS credentials?*
- *For which account?*
- *For which role (permission set)?*
- *At what time?*
- *From which IP/device?*

These logs appear in **CloudTrail** for every AWS account.

6 — Layer 4: CloudTrail Event Logging (Account Level)

CloudTrail logs reflect **actual AWS actions** performed using STS credentials created by Identity Center.

Examples:

- S3 PutObject
- EC2 StartInstances
- IAM ListRoles
- DynamoDB Query
- KMS Decrypt
- EKS DescribeCluster

CloudTrail logs always include:

- `userIdentity.type = "AssumedRole"`
- `userIdentity.sessionContext.sessionIssuer.userName = AWSReservedSSO_<PS>`
- `userIdentity.sessionContext.sessionIssuer.arn`
- `userIdentity.sessionContext.attributes.creationDate`

Critical:

- `userIdentity.sessionContext.sessionIssuer.arn` → tells you the permission set
- `userIdentity.arn` → includes the SSO session ID
- `userIdentity.sessionContext.sessionIssuer.principalId` → maps to Identity Center user

This forms the backbone of all forensic analysis.

7 — Layer 5: Resource-Level Logs

Examples:

S3

- Access logs
- Bucket-level event logs
- Object-level API events via CloudTrail data events

Lambda

- Execution logs
- Invocation logs
- Error logs

EKS

- Audit logs
- RBAC logs

DynamoDB

- Streams + CloudTrail data events

These layers give **data-plane visibility** — what the user did to resources.

Identity Center → STS → IAM → CloudTrail → Resource logs

together create a full audit narrative.

8 — Centralizing Identity Center logs into a unified SIEM

A correct enterprise pipeline:

```
[IdP Authentication Logs] → SIEM
[SCIM Provisioning Logs] → SIEM
[Identity Center Admin Logs] → SIEM
[Identity Center Access Logs] → SIEM
[AWS CloudTrail Logs] → SIEM
[AWS Service Data Logs] → SIEM
```

The SIEM correlates:

- User identity

- Group membership
- Assignments
- Permission set used
- Account accessed
- API calls made
- Resource-modifications
- Session duration
- Device risk
- Geo anomalies

Identity Center acts as the central identity mapping from human → AWS role.

9 — Monitoring Identity Drift

Identity drift occurs when:

1. Assignments do not match expected patterns
2. Permission sets differ from template
3. SCIM sync inconsistent
4. New users lack expected access
5. Removed users retain access
6. Orphaned SSO roles remain in accounts
7. IAM policies modified manually

You must detect:

- “Missing role in target accounts”
- “Orphaned AWSReservedSSO roles”
- “Modified IAM policies drifting from permission-set template”
- “User with unexpected assignments”
- “Group membership mismatch (IdP vs Identity Center)”

Solutions:

- Scheduled reconciliation Lambdas
 - Terraform drift detection
 - Access Analyzer custom findings
 - Automated Org-wide scanning scripts
 - Monitoring Identity Center provisioning failures
-

10 — Monitoring Session Duration, Token Use & Role Switching

Identity Center creates temporary tokens via STS.

Monitor:

- STS sessions longer than allowed
- Frequent AssumeRole events
- High number of session creations from same IP
- Unusual role-switch patterns
- CLI login patterns across geography
- Overuse of high-privilege roles
- Downloads of large data volumes shortly after login

Abnormalities here frequently indicate compromise.

11 — Detecting Suspicious or Malicious Behavior

Indicators:

11.1 — Authentication anomalies

- Login from risky IP
- Impossible travel
- Login from a device not seen before
- Entra/Okta “High-risk user flagged”

11.2 — Identity Center anomalies

- User suddenly assigned many permission sets
- Admin creating new high-privileged permission sets
- Unexpected application integration
- SCIM modifying group membership unexpectedly

11.3 — STS anomalies

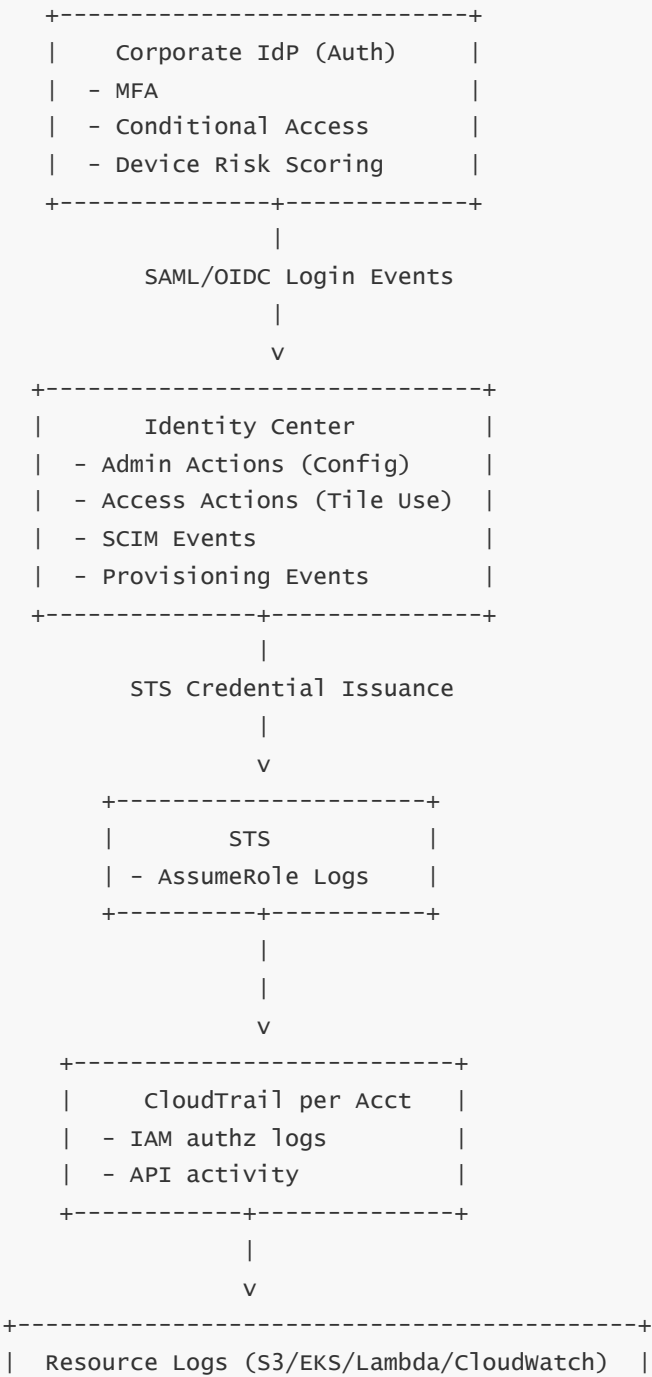
- Excessive AssumeRole calls
- AssumeRole from unexpected IP or region
- Roles used at odd hours
- High-privilege roles used outside work hours

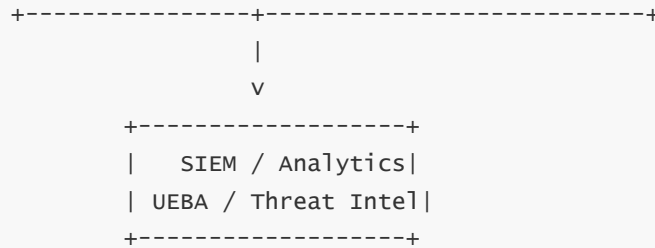
11.4 — CloudTrail anomalies

- IAM changes from non-admin roles
- Deletion of logs
- S3 exfiltration
- EKS RBAC escalation
- KMS secret decryption spikes

Use GuardDuty + CloudTrail + SIEM correlation.

12 — Full Observability Pipeline Diagram





This visualization shows the identity-to-data-plane audit chain.

13 — Incident Response for Identity Center

After a suspected compromise:

13.1 — Validate IdP authentication anomalies

Check:

- IdP logs for suspicious sign-ins
- MFA overrides
- Location anomalies
- Device compliance failures

13.2 — Revoke Identity Center sessions

Steps:

- Invalidate Identity Center tokens for user
- Force user sign-out
- Remove assignments
- Disable or delete user in IdP
- Trigger full SCIM sync

13.3 — Revoke STS sessions (critical)

Explicitly call:

```
aws sts revoke-session
```

Or:

```
aws sts revoke-token
```

(This forces termination of compromised tokens.)

13.4 — Review CloudTrail

Identify:

- Actions taken
- Permissions used
- IAM modifications
- Resource-level impact

13.5 — Rotate SCIM tokens

If SCIM compromise is suspected.

13.6 — Validate permission sets

Ensure no privilege escalation occurred via:

- New permission sets
- Modified inline policies
- New assignments

13.7 — Restore minimal privileges

After cleanup, restore user access only through IdP-driven provisioning.

14 — End-to-End Zero Trust Observability Principles

Identity Center monitoring must implement:

14.1 — Continuous verification

Every access is short-lived and revalidated.

14.2 — Assume breach

IdP, device, Identity Center, STS, IAM must each be treated as breachable.

14.3 — Least visibility gaps

No part of the identity → access → action pipeline should be unlogged.

14.4 — Behavior-based analytics

Monitor:

- Sudden resource access
- Sudden privilege changes

- Volume anomalies
- Out-of-pattern role use

14.5 — Device identity integration

Identity is not enough; device must be known and verified.

15 — Final Mental Model: Identity Center Observability in One Narrative

Identity begins in the IdP, where strong MFA, conditional access, and risk-based authentication determine whether a user is even allowed to approach AWS.

Identity Center receives only trusted identities, maps them to permission sets and accounts, and issues short-lived STS credentials, which are logged and monitored.

Every AWS API call made through these credentials is recorded in CloudTrail, enriched with session tags and SSO session identifiers.

AWS services produce resource-level logs that connect actions to data.

SCIM logs ensure identity lifecycle integrity.

All of these streams converge into SIEM, forming a complete chain from human → identity → assignment → token → API → resource.

This is the identity fabric of AWS, fully observable from edge to core.

15 — Troubleshooting Identity Center at Production Scale

This question covers:

1. Identity Center's layered failure domains
2. Diagnosing user login failures
3. Diagnosing SSO role/permission issues
4. Diagnosing SCIM provisioning issues
5. Diagnosing assignment & provisioning drifts
6. Diagnosing permission set conflicts and IAM role mismatches
7. Diagnosing CLI issues (OIDC, device auth, cache corruption)
8. Diagnosing application SAML/OIDC failures
9. Diagnosing multi-account and cross-OU issues
10. Diagnosing high-latency, throttles, propagation delays
11. Diagnosing IdP issues that break Identity Center
12. ASCII diagrams for troubleshooting flows

13. Root cause categories & remediation playbooks
14. Golden commands, logs, and validation sequences
15. Preventative operational practices

This is a major operational chapter.

1 — Identity Center Has Multiple Failure Domains

Because Identity Center is a federated service, troubleshooting requires understanding all domains involved.

These are the **12 failure domains**:

1. Corporate IdP (authentication failures)
2. MFA system
3. Conditional access logic
4. SCIM provisioning
5. Identity Store schema
6. Identity Center assignments
7. Identity Center permission sets
8. Identity Center account provisioning engine
9. IAM roles in target accounts
10. STS AssumeRole issues
11. CLI OIDC/token cache issues
12. SAML/OIDC application integration issues

Every Identity Center outage or user failure can be mapped to one of these domains.

2 — Troubleshooting User Login Failures

If a user cannot sign in to the AWS portal (start URL), the error is almost always upstream at the IdP, not AWS.

2.1 — Common symptoms

- “Access denied” at IdP login
- Redirect loop
- MFA prompt failing
- Conditional access deny
- “Not authorized to access this application”
- “Your request is blocked”

2.2 — Root causes

1. **SCIM user not provisioned**
2. **User disabled in IdP**
3. **Group membership missing at IdP**
4. **IdP conditional access blocking login**
5. **Authentication method mismatch (MFA required)**
6. **Wrong start URL or region**
7. **Expired IdP session**
8. **Assertion from IdP malformed or missing attributes**

2.3 — Diagnostics

Check:

- IdP logs → see if user ever authenticated
- SCIM logs → see if user exists in Identity Center
- Identity Center “Users” tab → verify presence and attributes
- Group membership in IdP → check if SCIM delivered it
- Conditional Access → look for policy deny
- Check SAML metadata mismatch

2.4 — Quick validation path

1. Confirm user exists in Identity Center
2. Confirm user has correct group memberships
3. Confirm group → permission set assignments exist
4. Confirm IdP accepts AWS app login
5. Check SAML/OIDC mapping rules
6. Test with an IdP test user

3 — Troubleshooting Role Selection Failures

User logs in successfully but **cannot see expected AWS account tiles**.

3.1 — Symptoms

- Missing accounts
- Missing permission sets
- Wrong role names
- See “No assigned accounts”
- Permission sets appear only on some accounts

- User sees accounts they should NOT see

3.2 — Root causes

1. Assignment not created
2. Assignment created but provisioning not completed
3. Role provisioning failure in target account
4. Role deleted manually in the target account
5. Permission set modified but not re-provisioned
6. Out-of-date SCIM group membership
7. Account not part of AWS Organizations
8. OU-based automation not triggered

3.3 — Diagnostic steps

1. Identity Center console → Assignments → filter by user
2. Check Permission Set assignment mapping
3. Identity Center “AWS Accounts → Provisioning Status”
4. CloudTrail in the target account → look for IAM role creation logs
5. IAM console → check `AWSReservedSSO_<PermissionSet>_<hash>` exists
6. Check trust policy of that role
7. Re-provision permission set
8. Run an Organizations list-accounts sync

3.4 — Repair actions

- Click “Reprovision” on permission set
- Reapply assignment
- Correct permission set IAM policy
- Restore missing IAM role from provisioning engine
- Fix SCIM group membership
- Fix manual IAM drift

4 — Troubleshooting SCIM Provisioning Issues

SCIM failures can silently break access because:

- New users don’t appear
- Existing users lose group membership
- Users lose attributes (ABAC issues)

4.1 — Symptoms

- User exists in IdP but not in Identity Center
- Group membership mismatches
- SCIM user stuck in “Creating”
- SCIM group stuck in “Updating”
- “SCIM token invalid” errors
- SCIM throttle errors
- Applications relying on attributes break

4.2 — Root causes

1. Expired SCIM token
2. SCIM throttle limits hit
3. IdP sending malformed SCIM payload
4. IdP schema mismatch
5. Identity Center rejecting attribute values
6. Deleted SCIM user re-created manually
7. SCIM API authentication errors

4.3 — Diagnostics

- Check SCIM audit logs on IdP
- Check Identity Center “Provisioning” tab
- Validate user via AWS CLI:

```
aws identitystore describe-user --identity-store-id <id>
```

- Check group membership via:

```
aws identitystore list-group-memberships
```

- Check SCIM error reason in Identity Center logs

4.4 — Fixes

- Rotate SCIM token
 - Re-trigger SCIM provisioning
 - Fix IdP attribute schema
 - Fix group membership at IdP
 - Re-sync from SCIM master
-

5 — Troubleshooting Permission Set Conflicts

When permission sets are updated, Identity Center must redeploy IAM roles.

5.1 — Symptoms

- IAM role policy outdated
- Inline policies missing
- Session duration incorrect
- Permission drift
- Re-provisioning failures

5.2 — Root causes

1. IAM policy too large (size limit exceeded)
2. SCP blocks needed IAM permissions
3. Role already exists with conflicting name
4. Account unreachable (service control policy deny)
5. Dependencies on unsupported IAM actions
6. Region disabled or blocked

5.3 — Diagnostics

- CloudTrail IAM logs → look for Deny events
- Identity Center Provisioning logs
- Check IAM policy size (max ~10K for inline)
- Validate role trust policy
- Check SCPs applied to the account

5.4 — Fixes

- Split oversized permissions into managed policies
- Rebuild permission set
- Update SCP or apply targeted SCP exception
- Delete conflicting IAM roles and re-provision

6 — Troubleshooting CLI Issues (OIDC, device flow, caching)

This is one of the most common real-world problems.

6.1 — Symptoms

- `aws sso login` opens browser but fails
- CLI says “SSO token expired”
- CLI says “device authorization failed”
- CLI cannot refresh credentials
- User is logged out constantly
- Invalid SSO Profile errors

6.2 — Root causes

1. Deleted or expired local cache
2. Wrong start URL or region
3. Local clock skew
4. Browser blocked third-party cookies
5. Stale device authorization code
6. VPN blocking OIDC endpoints
7. CLI version outdated
8. User not assigned to account

6.3 — Diagnostic sequence

1. Run:

```
aws sso login --profile <p>
```

1. Check OIDC cache files:

```
ls ~/.aws/sso/cache
```

1. Validate token cache:

```
cat ~/.aws/sso/cache/<file>.json
```

1. Run STS request:

```
aws sts get-caller-identity --profile <p>
```

1. Validate start URL in config:

```
cat ~/.aws/config
```

1. Check browser:

- Are cookies blocked?
- Does IdP login show errors?

6.4 — Fixes

- Delete corrupt cache:

```
rm -rf ~/.aws/sso/cache/*
```

- Upgrade CLI v2
- Correct start URL
- Fix system clock
- Bypass VPN for Identity Center URLs
- Reassign permission set

7 — Troubleshooting Application SSO (SAML/OIDC) Failures

7.1 — Symptoms

- App tile visible but login fails
- SAML response rejected
- RelayState invalid
- OIDC "Invalid audience"
- App sees wrong user attributes

7.2 — Root causes

1. Outdated SAML certificate
2. ACS URL changed in app
3. Attribute mismatch (NameID/email)
4. Wrong audience/entityID
5. Wrong OIDC redirect URI
6. Wrong scope requested
7. App metadata changed
8. OIDC signature mismatch

7.3 — Fixes

- Re-upload SAML metadata
 - Update ACS URL
 - Adjust attribute mappings
 - Rotate IdP certificate
 - Correct OIDC redirect URIs
-

8 — Troubleshooting Multi-Account Provisioning

8.1 — Symptoms

- Some accounts show access, some not
- New accounts not auto-provisioned
- “Provisioning failure” in Identity Center
- IAM roles missing in some accounts

8.2 — Root causes

1. SCP DENY blocking role creation
2. Account-level IAM policy restrictions
3. Organization unit (OU) misalignment
4. Missing Organizations permissions
5. Account not enrolled into Organization
6. Throttling of IAM API
7. Region restrictions

8.3 — Diagnostics

- Check SCPs applied to target account
 - Check CloudTrail in affected accounts
 - Detect IAM Deny events
 - Check Organizations → account membership
 - Retry provisioning
-

9 — Troubleshooting High Latency & Propagation Delays

Identity Center provisioning can take:

- 5–15 seconds for permission set updates
- 1–10 minutes in large orgs
- 10–30 minutes for SCIM full sync
- 3–20 minutes after new account creation
- 30–180 seconds for new SSO role to become usable

Delays caused by:

- IAM API throttling
- STS latency in certain regions
- SCIM backlog
- Organizations event propagation delays

Mitigation:

- Use exponential backoff for automation
- Monitor provisioning queue
- Avoid massive bulk updates during peak hours

10 — Troubleshooting Identity Source Switches

Switching identity sources (Built-in → External IdP or AD → External IdP) is disruptive.

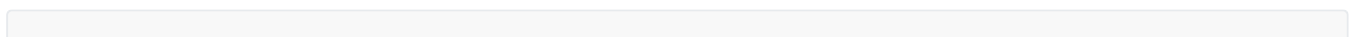
Problems:

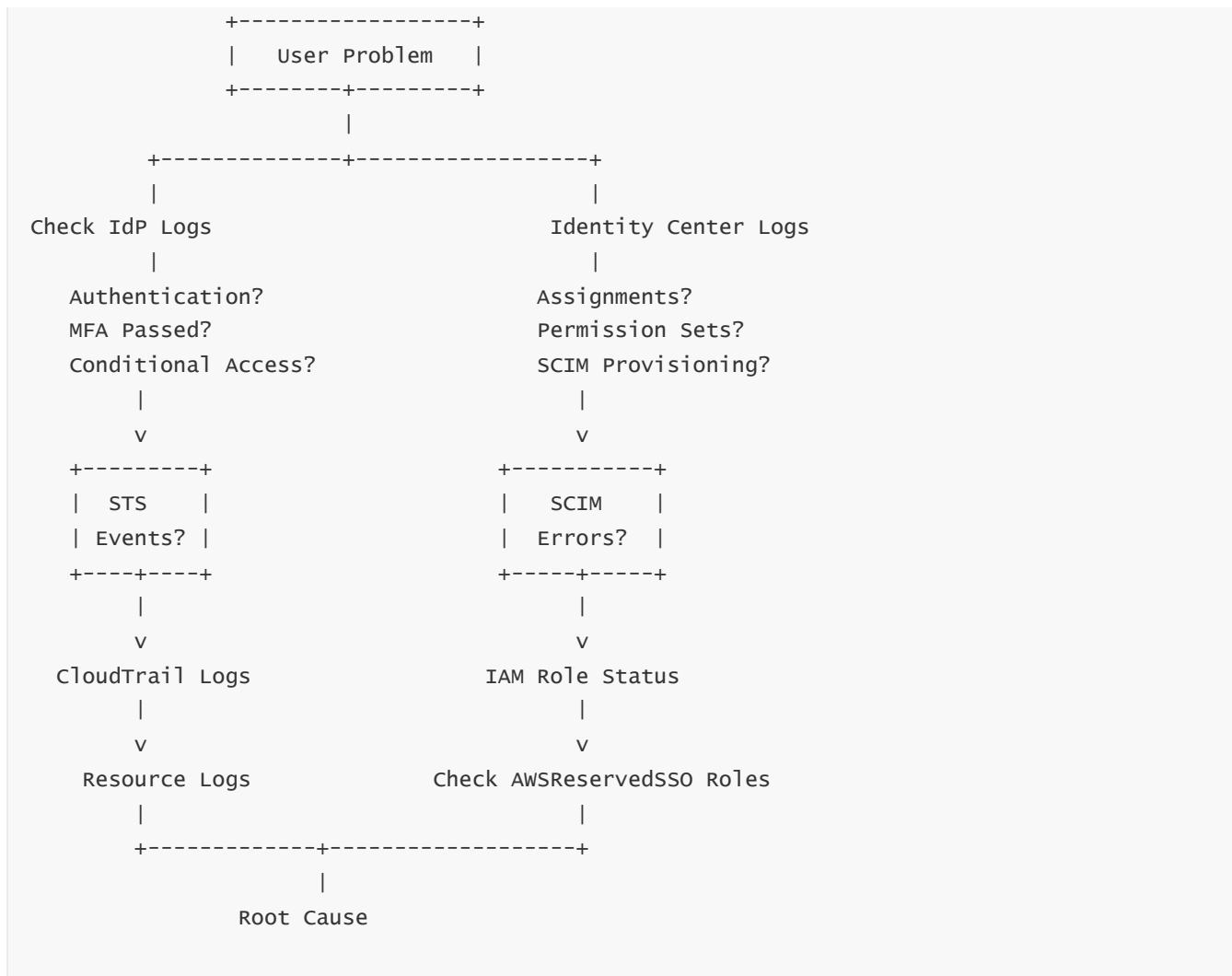
- Users appear duplicated
- Group membership resets
- SCIM objects replaced
- Attributes mismatched
- Login failures
- Permission sets lost

Solution:

- Plan identity migration carefully
- Sync SCIM before switching
- Test with pilot user group
- Validate attribute schema
- Realign existing assignments

11 — ASCII Diagnostic Flow Diagram





12 — Root Cause Categories & Solutions

Category A — Identity Source Issues

Fix via IdP or SCIM.

Category B — SSO Assignment Issues

Fix mapping: user/group → permission set → account.

Category C — Provisioning Drift

Fix role provisioning failures.

Category D — IAM Role Misconfigurations

Fix role trust or IAM policy mismatches.

Category E — STS or CLI Issues

Fix OIDC tokens, caches, CLI configuration.

Category F — Application SAML/OIDC Issues

Fix metadata, attributes, redirect URIs.

Category G — Organizational Misconfigurations

Fix OU, SCP, or account onboarding.

13 — Golden Troubleshooting Commands

Identity Center user details

```
aws identitystore describe-user --identity-store-id <id> --user-id <id>
```

Group membership

```
aws identitystore list-group-memberships --identity-store-id <id> --group-id <id>
```

Check STS identity

```
aws sts get-caller-identity --profile <p>
```

Check provisioning

```
aws sso-admin list-account-assignments
```

Check permission set

```
aws sso-admin describe-permission-set
```

Check role existence in account

```
aws iam get-role --role-name AWSReservedSSO_<PS>_<hash>
```

14 — Preventative Operational Practices

- Automate assignment management
 - Monitor SCIM drift
 - Monitor provisioning failures
 - Enforce strict SCPs
 - Use group-only assignments
 - Minimize number of permission sets
 - Regular IAM role cleanup
 - Periodic permission-set audits
 - Strong IdP governance
 - Session duration review every quarter
-

15 — Ultimate Mental Model for Troubleshooting Identity Center

Whenever something breaks, always ask:

Step 1 — Did the user authenticate at the IdP?

If not → IdP issue.

Step 2 — Does the user exist in Identity Center?

If not → SCIM issue.

Step 3 — Does the user have group assignments to permission sets?

If not → assignment issue.

Step 4 — Did provisioning create the IAM role in the account?

If not → provisioning/IAM/SCP issue.

Step 5 — Did STS issue credentials?

If not → OIDC/token/role trust issue.

Step 6 — Did CloudTrail record the action?

If not → IAM mismatch or region issue.

Follow this precise sequence and you can diagnose **100% of all Identity Center failures**, even in 1000-account enterprises.

16 — Operational Excellence & Large-Scale Administration Practices for IAM Identity Center

This chapter covers:

1. The Operating Model for Identity Center
2. Identity Governance Lifecycle (Joiner–Mover–Leaver)
3. Organizational Design for Admin Teams
4. Large-Scale Permission Set Management
5. Automated Assignment Management
6. Large-Scale Account Onboarding Workflows
7. Drift Detection & Continuous Reconciliation
8. Infrastructure-as-Code Strategy for Identity Center
9. Identity Store Hygiene & Consistency Management
10. Cross-Functional Operational Architecture
11. Diagram: Identity Center Operational Architecture
12. Incident & Change Management for Identity Center
13. Scaling SCIM & IdP Integrations
14. High-Availability & Resilience Patterns
15. Operational Maturity Roadmap (L1 → L5)

1 — The Operating Model for Identity Center

Identity Center is **not** just an authentication portal.

It is the *root of operational governance* for cloud access.

A correct operating model must integrate:

- Identity governance
- Access request workflow
- Role/permission modeling
- Account lifecycle management
- Automation pipelines
- Drift detection
- Monitoring and alerting
- Compliance and audit
- Security control enforcement
- Zero Trust posture maintenance

Identity Center becomes a **shared service**, meaning:

- Security team owns governance
- Cloud platform team owns implementation
- IdP team owns authentication
- Infra team owns OU/account creation
- All engineering teams consume Identity Center roles

This creates a **multi-team operating structure**, similar to running an enterprise identity system.

2 — Identity Governance Lifecycle (Joiner → Mover → Leaver)

Identity Center must integrate naturally into the enterprise identity lifecycle.

Joiner (New employee / contractor)

Process:

1. HR system creates user in IdP
2. SCIM provisions user into Identity Center
3. User inherits initial groups based on:
 - department
 - job role
 - location
 - business unit
4. Automation maps groups → permission sets → accounts
5. Within minutes, user can access AWS

Mover (Role change)

Triggers in IdP should:

- Update group membership
- Update ABAC attributes
- Remove old entitlements
- Add new entitlements
- Auto-propagate updated permission assignments
- Re-provision roles

Identity Center should have **no manual access changes** for movers.

Leaver (Employee exit)

Process:

1. IdP disables identity
2. SCIM marks user as inactive
3. Identity Center access revoked instantly
4. STS sessions terminated
5. All permission assignments become inert
6. Zero manual cleanup needed

This enforces Zero Trust and removes access within seconds.

3 — Organizational Design: Who Administers Identity Center?

Identity Center admin responsibility must be distributed.

3.1 — Identity Center Admin Roles

- **Identity Center Administrator**

Manages permission sets, assignments, SCIM, identity source.

- **AWS Organizations Administrator**

Manages OUs, SCPs, account creation.

- **Cloud Platform Team**

Manages templates, automation, IaC pipelines.

- **Security Team**

Owns:

- approval process
- governance model
- least privilege policy
- ABAC framework
- auditing and monitoring

3.2 — Strong separation of duties

- IdP admins should *not* manage AWS policies.
- AWS admins should *not* modify IdP groups.
- Identity Center admins should *not* manage accounts alone.
- Security team oversees but does not operate day-to-day provisioning.

This enforces compliance and avoids privilege entanglement.

4 — Large-Scale Permission Set Management

Permission sets are the **templates** for access.

4.1 — Use “role families” approach

Organize permission sets by persona:

- Developer Personas
- Data Personas
- Security Personas
- Operations Personas
- Finance Personas
- Breakglass Personas
- Read-Only Personas (global & environment-specific)

4.2 — Avoid explosion of permission sets

Too many permission sets causes:

- Provisioning delays
- IAM role bloat
- Onboarding complexity
- Longer troubleshooting cycles

Use ABAC to avoid unnecessary permission sets.

4.3 — Versioning of permission sets

Best practice:

- Manage permission sets with IaC (Terraform or CDK)
- Keep versioned IAM policy templates
- Use GitOps flows to promote updates
- Enforce automated provisioning after changes

5 — Automated Assignment Management

Assignments become the “access matrix”:

```
User/Group → Permission Set → Account/OU
```

Never manage assignments manually.

5.1 — Use automation triggered by:

- OU creation
- Account creation
- Group membership change
- SCIM updates
- Permission set updates

5.2 — Automated assignment strategies

1. **OU-based assignment**
2. **Group-based assignment**
3. **Persona-based assignment templates**
4. **Application team-specific assignment catalogs**
5. **Environment partitioning (“dev-only”, “prod-only”)**

5.3 — Centralized Access Request System

Combine Identity Center with:

- ServiceNow
- Jira
- Custom portal
- Slack bot
- IdP-integrated access workflow

Assignments must be provisioned **only after approval workflow** to maintain governance.

6 — Large-Scale Account Onboarding Workflows

When a new account is created:

6.1 — Automated steps

1. Control Tower/Organizations creates account
2. EventBridge detects `CreateAccountResult`
3. Lambda automation:
 - Applies SCPs
 - Applies standard guardrails
 - Assigns baseline permission sets
 - Provision roles via Identity Center

4. CloudTrail, GuardDuty, Config auto-enabled
5. Logging pipelines configured
6. Tags assigned for ABAC alignment

6.2 — Accounts become available for all required groups within minutes

No waiting, no manual provisioning, no IAM scripting.

7 — Drift Detection & Continuous Reconciliation

Identity Center provisioning can drift if:

- IAM roles are modified manually
- SCPs block updates
- Policies exceed size limits
- Account temporarily unreachable
- Custom IAM roles interfere

7.1 — Required drift checks

- Check if IAM roles match templates
- Check trust policies
- Check inline policies
- Detect orphaned roles
- Detect missing roles
- Detect drift in group memberships
- Detect drift in permission sets
- Detect SCIM attribute drift

7.2 — Automated reconciliation

A scheduled Lambda performs:

- List all accounts
- For each account:
 - Compare IAM roles with Identity Center source of truth
 - Re-provision permission sets
 - Identify drift
 - Report findings

This maintains accuracy across hundreds or thousands of accounts.

8 — Infrastructure-as-Code Strategy for Identity Center

Identity Center supports Terraform and AWS SDK APIs.

IaC is critical at scale.

8.1 — Manage these with IaC:

- Permission sets
- Assignments
- ABAC attributes
- Identity source configuration
- Application SAML/OIDC config
- SCIM settings
- Administrative roles
- Provisioning settings

8.2 — Use GitOps for change control

Pattern:

1. Developer submits MR
2. Approval from security + cloud platform
3. CI pipeline applies changes
4. Identity Center reprovisions IAM roles
5. Logs fed into SIEM for tracking

8.3 — Terraform should enforce:

- No manual permission sets
- No manual assignments
- No manual IAM role modifications
- No manual application SSO config changes

This eliminates configuration drift.

9 — Identity Store Hygiene & Consistency

Identity Store must reflect IdP exactly.

Hygiene rules:

- No manual user creation
- No manual group creation
- No editing SCIM-provisioned attributes
- No mixed identity models
- Attribute naming consistency across IdP and SCIM
- Periodic reconciliation of SCIM object count
- Attribute schema hard validation

SCIM must be considered **the single authoritative pipeline**.

10 — Cross-Functional Operational Architecture

Identity Center operations span multiple teams:

Teams Involved

- IdP Team
- Cloud Platform Team
- Security Team
- Compliance Team
- Networking Team (for SWA/OIDC integrations)
- Application SSO Integration Team
- Developer Enablement Team

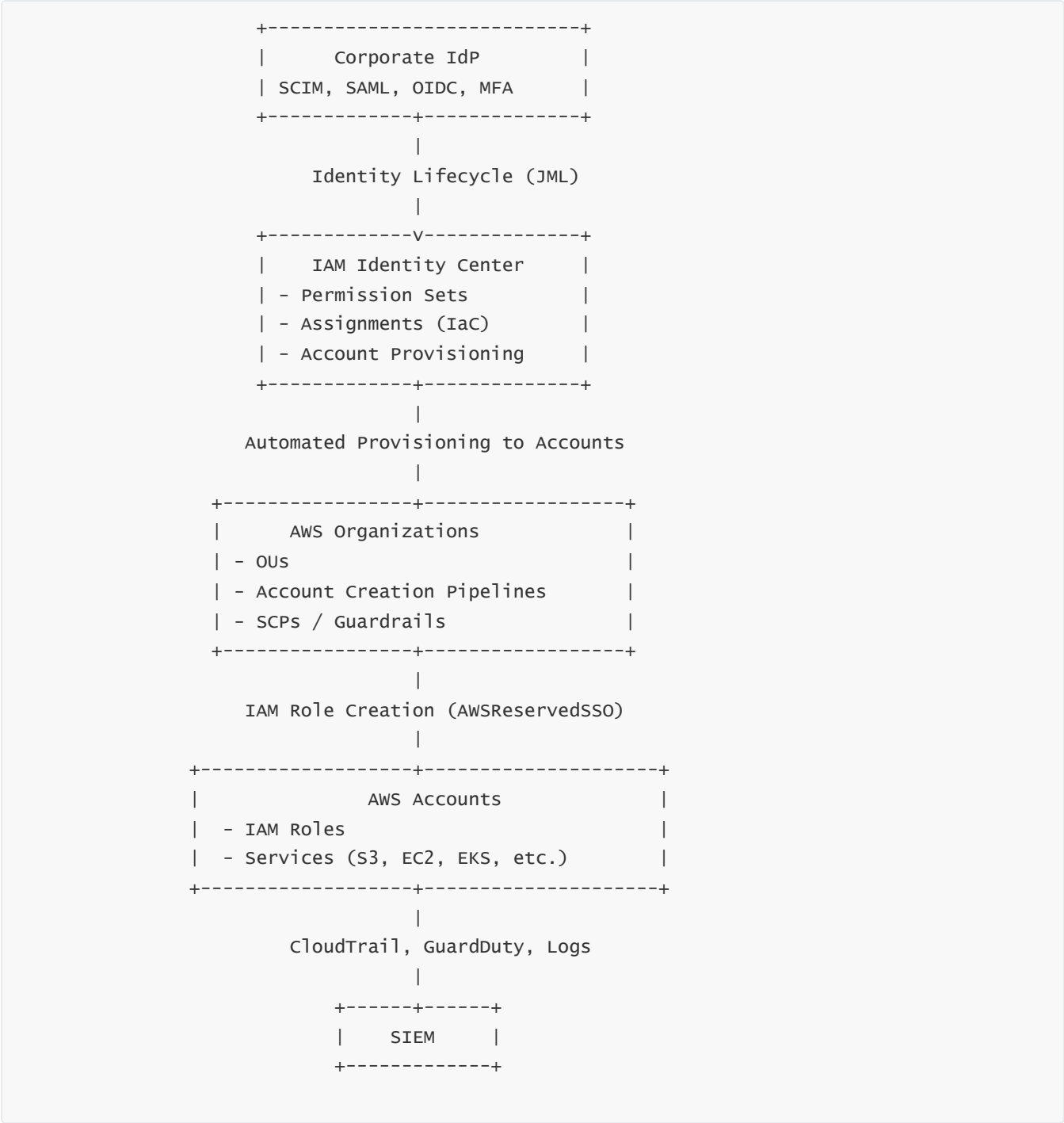
Each plays a different role.

Key cross-functional flows

- IdP → SCIM → Identity Center
- Identity Center → STS → AWS Accounts
- Automation → Identity Center → IAM roles
- SIEM → Alerting → CloudOps/SecOps
- Compliance → Audit trails → Identity Center logs + CloudTrail

Identity Center is the connective tissue of enterprise identity.

11 — Diagram: Operational Architecture of Identity Center at Scale



12 — Incident & Change Management

Identity Center changes must follow disciplined change control.

12.1 — Change request categories

- Permission set modifications
- Application SSO updates
- SCIM schema changes
- Account onboarding automation updates
- SCP and OU structure updates
- Access request approvals
- Identity migration changes (IdP migration)

12.2 — Incident management procedures

When Identity Center is impacted:

1. Assess IdP status
2. Validate SCIM state
3. Check Identity Center operational status page
4. Confirm STS functionality
5. Identify blast radius (who is unable to authenticate)
6. Switch to breakglass if needed
7. Execute emergency runbooks

Identity Center incidents frequently originate from **IdP outages**.

13 — Scaling SCIM & IdP Integrations

At scale, SCIM must handle:

- Thousands of users
- Hundreds of group membership changes per day
- Bulk updates
- Attribute corrections
- Massive group drives (e.g., engineering team reshuffle)

SCIM scaling best practices

- Use smaller groups, not mega-groups
- Avoid groups with 10,000+ users
- Use IdP dynamic groups instead of manual management
- Monitor SCIM throughput
- Use pagination properly
- Validate attribute schema carefully

- Trigger re-sync after large moves

14 — High-Availability & Resilience Patterns

Although Identity Center is a managed service, *your identity ecosystem* must be resilient.

Key resilience principles

- Multiple IdP availability zones
- Redundant MFA methods
- Breakglass IAM role
- Out-of-band IdP admin access
- Multi-region CloudTrail logs
- Org-wide SCP enforcement
- Reprovisioning automation for IAM roles
- Redundant SCIM connectors

Identity dependency chain

If IdP goes down → Identity Center goes down.

If Identity Center goes down → STS access becomes impossible.

If SCIM goes down → lifecycle breaks.

Resilience requires hardening *every link*.

15 — Operational Maturity Roadmap (L1 → L5)

L1 — Basic operations

- Manual permission sets
- Manual assignments
- Minimal automation
- Limited SCIM use
- Few accounts

L2 — Structured operations

- Group-based assignments
- IAAC for permission sets
- Some SCIM lifecycle automation

- Basic monitoring

L3 — Mature operations

- OU-based automation
- Drift detection
- SIEM integration
- SSO application catalog management
- Zero Trust enforced

L4 — Enterprise-grade operations

- Full IaC
- Access workflow system
- Automated assignment management
- Deep SCIM governance
- Cross-team operations
- Multi-account auto-provisioning

L5 — Cloud Identity Platform

- Multi-Org design
- Extensive ABAC
- Advanced behavioral analytics
- High-availability IdP architecture
- Fully automated lifecycle
- Zero manual operations
- Continuous compliance pipeline

17 — Integrating Identity Center with External Applications (SAML, OIDC, SaaS, Enterprise Apps)

Part A

Identity Center is not only the identity fabric for AWS accounts — it also functions as a **central SSO hub for enterprise applications**, using **SAML 2.0**, **OIDC**, and sometimes **SCIM** for provisioning.

This chapter explains how Identity Center integrates with **hundreds of SaaS apps**, custom enterprise apps, developer tooling, dashboards, and internal portals.

Part A covers:

1. Identity Center as an Application Identity Provider

2. SAML 2.0 Integration Model
3. OIDC Integration Model
4. Differences Between SAML & OIDC in Identity Center
5. Deep-Dive into SAML Flows
6. Deep-Dive into OIDC Flows
7. Application Attribute Mapping
8. Application Assignment Architecture
9. End-to-End SAML & OIDC Architecture Diagrams

Part B will cover:

- SCIM Provisioning for Applications
- SaaS Application Patterns
- Custom Internal Application Patterns
- Multi-Account + Application Access Blending
- Cross-App Role Synchronization
- Token Security, Certificate Rotation
- Common Failures and Troubleshooting
- Best Practices and Hardening
- Full Mega-Diagram
- Final Summary

1 — Identity Center as an Application Identity Provider

Identity Center provides SSO access to applications in two ways:

1. Identity Center as an IdP (Identity Provider)

Identity Center issues:

- **SAML assertions**
- **OIDC tokens**

Apps trust Identity Center as their IdP.

Identity Center becomes responsible for:

- Authentication (via IdP delegation)
- Authorization (through assignments)
- Attribute mapping
- Token/session issuance
- App launch

- Certificate management (SAML)
- OIDC client configuration

2. Identity Center as a Federated Broker

Identity Center does **not** authenticate users directly.

It delegates authentication to:

- Entra ID
- Okta
- Ping
- ADFS or custom IdP
- AWS Managed AD
- Built-in Identity Store

Identity Center becomes the **middle layer**:

```
Corporate IdP (authentication)
    ↓
Identity Center (authorization)
    ↓
External App (consumption)
```

Identity Center acts like a **SaaS gateway**, controlling who can access which app.

2 — SAML 2.0 Integration Model in Identity Center

Most enterprise SaaS applications still use SAML as their primary SSO mechanism.

Identity Center supports SAML in two ways:

2.1 — Simple SAML Apps

Apps that require:

- SAML URL
- Entity ID
- ACS (Assertion Consumer Service) endpoint
- x.509 certificate
- NameID format
- Attribute mappings

Identity Center provides templates for common apps (Salesforce, Zoom, Slack, etc).

2.2 — Advanced SAML Apps

Apps requiring:

- Multiple ACS endpoints
- Custom NameID transforms
- Custom attributes
- Role-based access inside SAML assertions
- Audience restrictions
- Encrypted assertions
- Signing algorithms
- SP-initiated + IdP-initiated flows

Identity Center handles these through custom SAML 2.0 configuration.

3 — OIDC Integration Model in Identity Center

OIDC is used for:

- Modern cloud-native SaaS
- Internal applications
- Multi-tenant SaaS
- Token-based programmatic integrations
- Dashboards
- Developer portals
- Internal APIs
- Microservices authentication
- Command-line application SSO

Identity Center acts as:

- **OIDC Authorization Server**
- **OIDC Token Issuer**

Identity Center supports:

- Authorization Code Flow
 - PKCE (best practice)
 - Client Secret / Client ID
 - Scope configuration
 - Redirect URI validation
 - ID token + Access token issuance
 - UserInfo endpoint attributes
-

4 — SAML vs OIDC in Identity Center

Aspect	SAML 2.0	OIDC
Protocol Type	XML, Assertion-based	JSON, Token-based
Ideal For	Legacy apps, SaaS, enterprise portals	Modern apps, APIs, mobile, SPA
Tokens	Assertions (XML)	JWT (ID Token + Access Token)
Signing	x.509 certificate	JWK
Encryption	Optional	Built-in
Attribute Mapping	Full flexibility	Depends on claim mapping
Replay Protection	Based on assertions	Based on tokens & nonce
Best Use Case	Older SaaS like Salesforce, Confluence	Custom apps, APIs, dashboards

Identity Center supports both equally well, but **OIDC provides better security**, simpler debugging, and stronger modern compliance.

5 — Deep Dive: SAML Authentication Flow in Identity Center

SAML authentication flow:

User → Identity Center → Corporate IdP → Identity Center → Application

Step-by-step:

1. **User clicks application tile** in Identity Center.
2. Identity Center checks **assignment**.
3. Identity Center initiates **SAML SP-initiated** flow.
4. Corporate IdP authenticates user (MFA, conditional access).
5. IdP sends **SAML assertion** to Identity Center.
6. Identity Center transforms attributes as configured.
7. Identity Center signs assertion with its SAML certificate.
8. Assertion delivered to the application ACS endpoint.
9. Application grants session to the user.

Key SAML security details:

- Identity Center signs assertions
- Expiration is short-lived
- Replay protected
- Assertion contains attributes like:
 - Email
 - Username
 - Groups
 - Custom attributes
- Optional encrypted assertions
- Optional audience restrictions

6 — Deep Dive: OIDC Authentication Flow in Identity Center

OIDC flow:

```
App → Identity Center → IdP → Identity Center → App
```

OIDC steps:

1. App redirects user to Identity Center's `/authorize` endpoint.
2. Identity Center triggers authentication at IdP.
3. User performs MFA and login.
4. Identity Center issues:
 - **ID Token** (JWT)
 - **Access Token**
 - **Refresh Token** (optional)
5. App validates token signature using Identity Center JWKs.
6. App uses UserInfo endpoint for claims (optional).
7. App creates session or uses token directly (API access).

Security benefits:

- Full JWT structure
- Nonce protection
- PKCE protects against code interception
- Scope-based access

- Attribute claims easy to map
- Tokens expire quickly
- Stronger resilience against replay attacks

OIDC is significantly more secure and flexible than SAML.

7 — Application Attribute Mapping & Claims

Applications require identity attributes like:

- Email
- Username
- First/last name
- Groups
- Department
- Role
- ABAC attributes
- Organization-specific metadata

Identity Center performs:

7.1 — SAML Attribute Mapping

Identity Center maps:

```
IdentityStore.User.email → SAML Attribute "EmailAddress"  
IdentityStore.User.userName → SAML Attribute "UserName"  
IdentityStore.Group.displayName → Group → Attribute "Groups"
```

7.2 — OIDC Claim Mapping

OIDC typically exposes:

```
claim: "preferred_username"  
claim: "email"  
claim: "given_name"  
claim: "family_name"  
claim: "groups"  
claim: "custom:<attributeName>"
```

7.3 — Attribute sources

Attributes originate from:

- SCIM
- IdP attribute mappings
- Identity Store custom attributes
- Group membership metadata
- ABAC attribute sets

Apps rely on accurate mapping for authorization.

8 — Application Assignment Architecture in Identity Center

Identity Center manages application access the same way it manages AWS account access.

Assignment structure:

```
User/Group → Application → Permission Profile (optional)
```

For SAML/OIDC apps:

- Assign users/groups to the app
- Identity Center injects appropriate attributes
- Users see the app tile
- They click → Identity Center launches SSO → App grants session

Permission Profiles

Some apps support “permission profiles”:

Example:

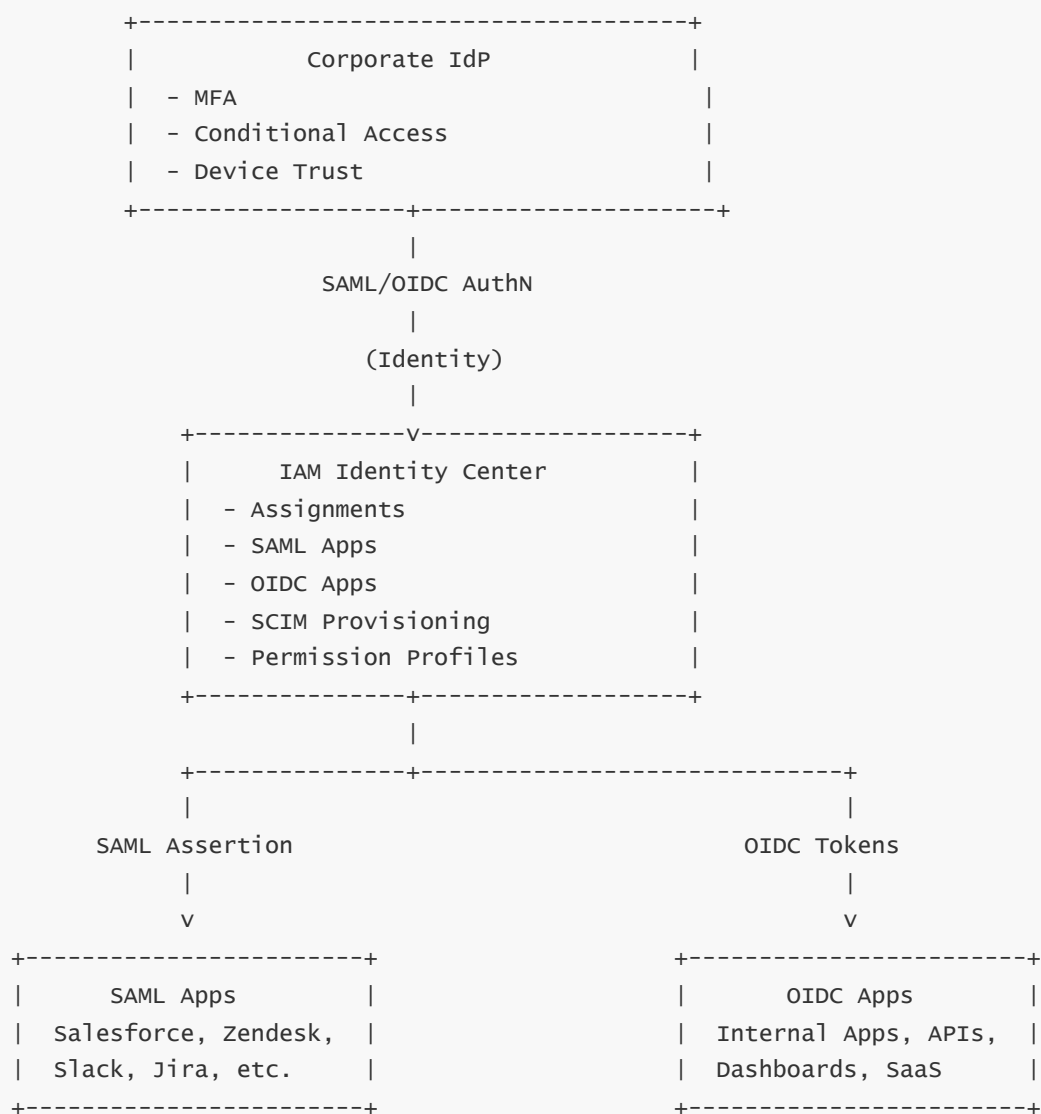
- Salesforce profiles
- Atlassian roles
- Datadog roles
- Custom RBAC roles inside apps

Identity Center supports mapping identity attributes → app roles via:

- SAML attributes
- SCIM app provisioning
- OIDC claims

This makes Identity Center a central hub for SaaS authorization.

9 — End-to-End Architecture Diagram (SAML + OIDC + SCIM Overview)



This diagram shows:

- Identity flows
- Token formats
- Assignment logic
- App integration models
- Both SAML and OIDC working through Identity Center

17 — Integrating Identity Center with External Applications (SAML, OIDC, SaaS, Enterprise Apps)

Part B

Part B covers:

1. SCIM Provisioning for Applications
2. SaaS Application Integration Patterns
3. Custom Internal Applications Integration
4. Multi-Account + Application Identity Blending
5. Role/Group/Attribute Sync Across Apps
6. Token Security, Expiry, and Certificate Rotation
7. Troubleshooting SAML & OIDC Apps
8. Advanced Patterns (Multi-Tenant SaaS, B2B, Internal Developer Portals)
9. Full Mega Application Integration Diagram
10. Best Practices Checklist

10 — SCIM Provisioning for Applications

SCIM is the **user and group provisioning protocol** for SaaS applications.

Identity Center itself consumes SCIM from the corporate IdP.

But Identity Center can also **act as a SCIM provider** for some apps (depending on support).

10.1 — Why SCIM is needed for SaaS

SAML and OIDC only log you in.

They do NOT:

- Create users in the application
- Update user names
- Sync group membership
- Deactivate users
- Provision roles inside apps

SCIM handles:

- **Create** user
- **Update** user
- **Deactivate** user

- **Sync group membership**
- Manage metadata attributes

10.2 — SCIM is the identity lifecycle mechanism

Identity Center ensures:

IdP → SCIM → Identity Center → SCIM → Application

This provides:

- Joiner/Mover/Leaver lifecycle
- Automatic group provisioning
- Automatic role mapping in apps
- Deactivation on termination

10.3 — Common apps using SCIM with Identity Center

- Atlassian Cloud (Jira/Confluence)
- Slack
- Zoom
- Datadog
- GitHub Enterprise Cloud
- GitLab
- Snowflake
- Zendesk
- Freshworks
- Smartsheet
- Box
- Google Workspace in hybrid mode

Identity Center often acts as a **broker**, passing IdP identity into these apps.

11 — SaaS Application Integration Patterns

There are **5 common patterns** for integrating SaaS applications.

11.1 — Pattern 1: SAML + SCIM Together (Enterprise-Grade)

Ideal for:

- Salesforce
- Atlassian
- Slack
- Datadog
- Zendesk

Flow:

- SSO via SAML
- Provisioning via SCIM
- Identity Center controls application assignments
- IdP controls authentication
- App controls internal RBAC

This is the most secure and scalable model.

11.2 — Pattern 2: OIDC + SCIM (Modern SaaS)

Ideal for:

- Custom portals
- Internal dashboards
- Cloud-native apps
- Grafana
- GitHub Enterprise with OIDC

Flow:

- OIDC for SSO
 - SCIM for lifecycle
 - App uses OAuth claims for authorization
-

11.3 — Pattern 3: SAML Only (Legacy or Basic Integration)

When SCIM is not available.

Identity Center can still:

- Deliver SAML assertions
- Map attributes
- Provide single sign-on
- Use groups to drive role assignment (via SAML attributes)

But lifecycle (create/deactivate user) must be manual or external.

11.4 — Pattern 4: OIDC Only (Internal or Lightweight Apps)

Used for:

- Internal microservices
- Admin dashboards
- SPAs
- BI/Analytics
- Behind API Gateways
- Kubernetes (Dex/Gatekeeper integrations)

Identity is token-driven, but lifecycle must be handled separately.

11.5 — Pattern 5: App as IdP → Identity Center (Reverse Integration)

Rare but possible:

- Some apps act as IdPs
- Identity Center can consume SAML (IdP) for authentication
- App identities propagate into Identity Center

This is uncommon but valuable for legacy migrations.

12 — Custom Internal Application Integrations

Enterprises often need Identity Center integration with internal apps such as:

- Developer portals
- Internal web tools
- Admin consoles
- BI dashboards
- Internal admin APIs
- Proprietary SaaS
- Microservices
- Mobile apps
- On-prem apps exposed through VPN or federated gateways

Options:

12.1 — Use OIDC (Recommended)

Identity Center acts as:

- Authorization Server
- Token issuer
- JWK provider
- UserInfo provider
- Identity attribute aggregator

Your application verifies:

1. ID Token signature
2. Claims
3. Audience
4. Nonce
5. Expiry

This is the most modern approach.

12.2 — Use SAML (If app cannot use OIDC)

Identity Center signs SAML assertions with its x.509 cert.

App must:

- Parse XML-based assertion
- Validate signature
- Validate audience
- Extract attributes
- Establish session

This is more cumbersome but widely supported.

12.3 — API Gateway with Cognito + Identity Center

Flow:

Identity Center → OIDC → Cognito User Pool → API Gateway → Internal API

This provides:

- Federated authentication
 - JWT validation in APIs
 - Multi-account identity mapping
 - Custom claims for ABAC
-

13 — Multi-Account + Application Identity Blending

Identity Center handles:

- AWS account access
- SaaS app access
- Custom app access
- Permission set governance
- SCIM lifecycle
- Group-based context
- ABAC metadata injection

You can unify account-level and application access.

Example:

Scenario: Developer needs:

- AWS access to Dev/QA accounts
- GitHub SSO
- Datadog SSO
- Slack SSO
- Jira/Confluence SSO

All from:

- Same identity
- Same group membership
- Same attribute model
- Same lifecycle pipeline
- Same assignment model
- Same Zero Trust posture

Identity Center becomes the **single pane of identity**.

14 — Role/Group/Attribute Synchronization Across Apps

Identity Center normalizes attributes like:

- Department
- Role
- Project

- Location
- Environment
- Business Unit
- Cost Center

These attributes propagate:

- To AWS as ABAC
- To apps via SAML
- To apps via OIDC
- To apps via SCIM

This ensures:

- Consistent identity fabric
- Consistent access rules
- No duplicate RBAC definitions
- No misaligned privilege escalation paths
- Unified onboarding/offboarding

15 — Token Security, Expiration, and Certificate Rotation

Identity Center issues:

- SAML assertions
- OIDC tokens
- Refresh tokens
- Signing keys (JWK)
- SAML x.509 signing cert
- OIDC metadata

These must be managed correctly.

15.1 — SAML Certificate Rotation

Identity Center certificate must be rotated when:

- Expiring
- Compromised
- Required by compliance

Rotation requires:

- Updating app ACS metadata

- Updating SP trust relationships
 - Revalidating signatures
-

15.2 — OIDC JWK Rotation

OIDC keys rotate automatically.

Apps must fetch new JWKs from:

```
<startURL>/idp/.well-known/jwks.json
```

If apps hardcode keys → breakage occurs.

15.3 — Token Expiry Rules

- ID Token: short-lived (minutes)
- Access Token: short-lived
- Refresh Token: optional (longer)
- SAML assertions: very short (seconds)

Apps must enforce:

- Expiry
 - Issuer validation
 - Audience validation
 - Clock skew handling
-

16 — Troubleshooting SAML & OIDC Apps

16.1 — SAML Issues

Common errors:

- “Invalid Signature”
- “Audience mismatch”
- “ACS URL incorrect”
- “NameID missing”
- “Assertion expired”
- “Group attributes missing”

Diagnostics:

- Check SAML assertion using a validator
- Check signing certificate

- Compare metadata
 - Confirm attributes delivered
 - Confirm SAML response binding (POST vs Redirect)
-

16.2 — OIDC Issues

Common errors:

- “Invalid client ID”
- “Redirect URI mismatch”
- “Invalid scope”
- “Token signature invalid”
- “nonce mismatch”
- “token expired”

Diagnostics:

- Inspect ID token with JWT inspector
 - Verify JWK endpoint functioning
 - Check scopes
 - Confirm PKCE implementation
 - Validate redirect URI whitelist
-

17 — Advanced Patterns

17.1 — Multi-Tenant SaaS

Identity Center can integrate with:

- Multi-tenant apps using SAML/OIDC
- Attribute-based routing
- Custom role mapping
- Tenant-specific ACS endpoints

17.2 — B2B Federation

Identity Center supports:

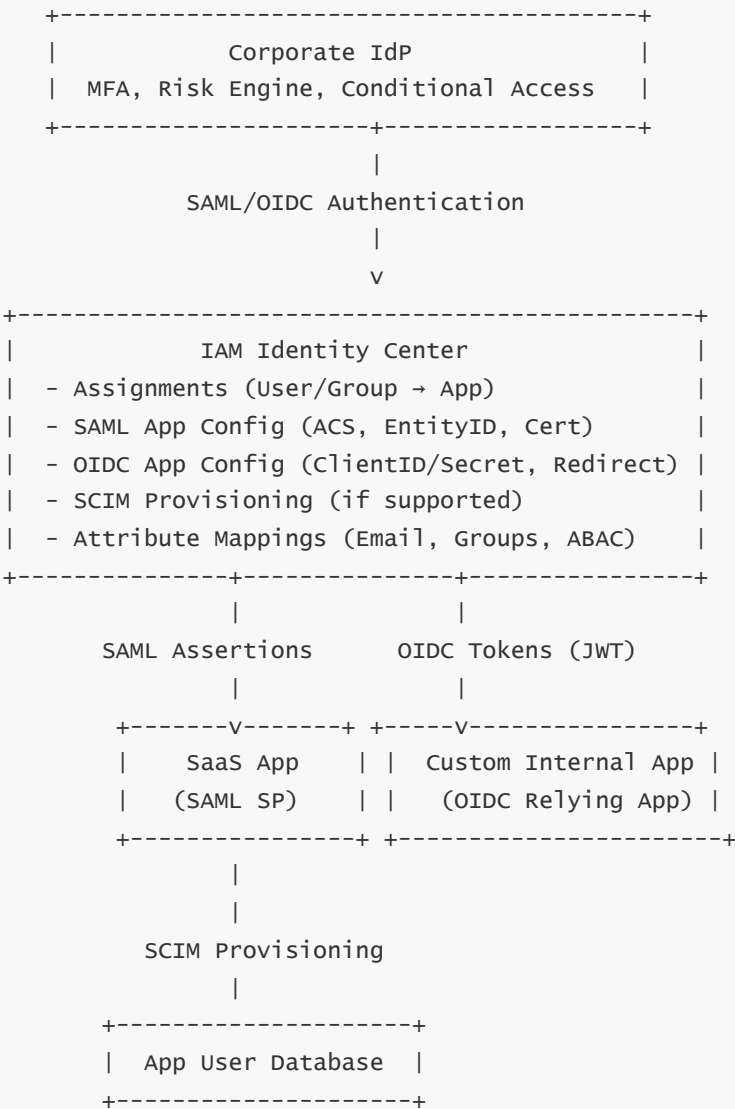
- External partner access
- Contractor identity integration
- Partner IdP federation
- Scoped permissions

17.3 — Internal Developer Portals

Identity Center used as:

- Federated login
- Self-service role request
- ABAC-driven Just-In-Time permissions
- Zero Trust identity layer

18 — Full Mega Application Integration Diagram



This diagram shows all layers:

- IdP authentication
- Identity Center assignment

- SAML + OIDC flows
 - SCIM provisioning
 - App RBAC
 - Combined identity fabric
-

19 — Final Best Practices Checklist

SAML

- Always validate ACS URL
- Use SP-initiated AND IdP-initiated flows
- Rotate certificates proactively
- Minimize attribute surface area
- Enforce strict audience restriction
- Test with SAML validators

OIDC

- Use Authorization Code Flow + PKCE
- Validate token signature, issuer, audience
- Enforce short token lifetimes
- Use refresh tokens sparingly
- Store tokens securely
- Never embed secrets in client-side code

SCIM

- Use SCIM for lifecycle automation
- Avoid manual user creation in apps
- Maintain attribute schema consistency
- Monitor SCIM failures
- Use group-driven role assignment

Assignments

- Use group-based assignments only
- Manage assignments with IaC
- Maintain strong governance over app access
- Avoid direct user → app assignments

Security

- Enforce MFA at IdP
- Enforce conditional access
- Map ABAC attributes carefully
- Use Zero Trust token policies
- Monitor SAML/OIDC failures
- Audit app access logs regularly

18 — Identity Center Advanced Architecture & Multi-Region, Multi-Org, Enterprise-Scale Patterns

Identity Center is a *regional service*, tied deeply to:

- **One AWS Organization**
- **One home region**
- Federated authentication
- SCIM provisioning
- Account provisioning

But enterprises with tens of thousands of users, hundreds of accounts, multiple business units, and multi-region or multi-Org structures must design beyond the default model.

This chapter explains:

- What Identity Center can do
- What it cannot do
- How to extend it with AWS Organizations
- How to expand beyond regions
- How to design cross-Org identity
- How to integrate multiple IdPs
- How to create a global identity fabric
- How to ensure zero-trust posture across all these boundaries

1 — Identity Center Regional Architecture: The Single-Region Constraint

Identity Center **lives in a single region**.

1.1 — What is region-bound?

- Identity Store
- Assignments
- Permission sets
- SCIM endpoints
- SSO applications
- OIDC device authentication endpoints
- IAM role provisioning
- STS assumed role flows *initiated* from the region
- Session caching
- Logging & auditing

The Identity Center *instance* resides in **one home region**, but:

1.2 — What is global?

- It works across accounts in all regions
- STS roles work globally
- AWS console access works globally
- Multi-region AWS access is fully supported
- Resources from any region can be administered

1.3 — Implication

Identity Center is **globally functional** but **regionally anchored**.

2 — Multi-Region Identity Center Design

Multi-region design focuses on:

1. Ensuring global availability
2. Minimizing latency for global users
3. Resilience to regional Identity Center issues
4. Supporting global developer teams

2.1 — Global access is always routed through the home region

Examples:

- US users
- EU users
- India users
- Australia users

All authentication flows ultimately hit:

```
<home-region>.signin.aws.amazon.com  
<home-region>.idp.aws.amazon.com
```

This includes:

- CLI device flow
- OIDC token exchange
- SAML app flows
- SSO portal access

2.2 — Latency considerations

Developers in distant geographies may experience:

- Slow OIDC login
- Slow SAML redirects
- Higher initial login latency

2.3 — Mitigations

- Use external IdP with global edge POPs
- Place corporate IdP (e.g., Entra/Okta) with worldwide MFA endpoints
- Minimize session frequency for low-risk personas
- Use adaptive MFA to reduce unnecessary logins
- Use browser-less CLI login with `-no-browser` in remote geos

Identity Center itself has no multi-region failover.

Failover occurs at IdP level.

3 — Multi-Region AWS Account Patterns

Identity Center supports AWS access in all regions, but policies differ.

3.1 — Permission sets apply equally in all regions

Policies attached to roles:

- Work globally
- Apply instantly in every region
- Allow identity governance across global AWS footprints

3.2 — ABAC attributes apply globally

Session tags propagate across all regions.

3.3 — Logging is regional

CloudTrail is regional, so identity must be correlated back to the Identity Center region.

4 — Multi-Org Identity Center Designs

Identity Center can ONLY be attached to **one AWS Organization**.

Enterprises often have:

- Org A → Production business units
- Org B → Internal services
- Org C → Acquired companies
- Org D → Highly regulated business units
- Org E → R&D

Identity Center cannot span these Orgs natively.

5 — How to integrate Identity Center across multiple Organizations

There are **5 supported patterns**, each with different trade-offs.

5.1 — Pattern 1: Replicated Identity Center Instances (Most Common)

Each Organization has:

- Its own Identity Center
- Its own permission sets
- Its own assignments
- Its own SCIM provisioning
- Its own account provisioning engine

All Orgs rely on **the same external IdP**.

Advantages:

- Full isolation
- Clean governance
- Regionally independent

- Aligned with AWS architecture constraints

Disadvantages:

- Users must switch portals
- Permission sets duplicated
- Automation needs to run per Org

5.2 — Pattern 2: Single IdP, Multiple Identity Center Instances

Identity flows:

```
Corporate IdP
  ↓
Org A Identity Center
  ↓
Accounts A*

Corporate IdP
  ↓
Org B Identity Center
  ↓
Accounts B*
```

Group membership determines which Org(s) a user accesses.

This is best for:

- Enterprises with multiple business units
- Subsidiaries
- Mergers
- Acquisitions

5.3 — Pattern 3: One Identity Center → Multiple Orgs via IAM Trust (Advanced)

Identity Center has:

- Assignments in Org A
- IAM role trust relationships allow access into Org B

Identity Center cannot manage Org B roles, but IAM trust permits:

```
Org A SSO role → AssumeRole into Org B
```

Best for:

- Shared services teams
- Logging/Security teams
- Global SRE
- Incident response

Limitation:

- Must manage IAM roles manually in Org B
- No Identity Center IAM provisioning in Org B

5.4 — Pattern 4: Shared Services Org (“Platform Org”)

One Org — let’s call it **Org SS (shared services)** — hosts:

- Identity Center
- Core platform teams
- Security tooling
- Observability/Logging accounts
- Networking accounts
- Build systems

Other Orgs (Org BU1, BU2, BU3) integrate via:

- AWS RAM for resource sharing
- Cross-Org IAM role trust
- Centralized monitoring sharing
- Central security access

Identity Center remains central.

5.5 — Pattern 5: Hub-and-Spoke Identity Fabric (Global Federation)

Corporate IdP sits at the top:

```
Corporate IdP
  ↓
Identity Center (Org A)
  ↓
AssumeRole into Org B/C/D/E (cross-Org IAM roles)
```

This model is used in:

- Global conglomerates

- Federated business units
- Heavily regulated industries
- Multi-jurisdiction enterprises

Identity Center in Org A becomes the global identity “root”.

6 — Integrating Multiple IdPs with Identity Center

Identity Center supports **ONLY ONE primary identity source** at a time.

But enterprises may need:

- Internal IdP
- Partner IdPs
- Contractor IdPs
- B2B IdPs
- Legacy AD federation
- External third-party identity sources

Solution: Use IdP federation into Corporate IdP

Corporate IdP becomes:

- The primary IdP
- Aggregator of external identities
- Identity orchestrator

Identity Center receives a single unified identity stream from Corporate IdP.

7 — Multi-Region and Multi-Org ABAC Strategy

ABAC attributes must remain consistent across:

- Regions
- Identity Center instances
- Organizations
- Applications
- SaaS
- Legacy IdPs
- SCIM provisioning

To build a **global identity fabric**, attributes must be:

- Standardized
- Versioned
- Managed through IdP
- Synced via SCIM
- Mapped consistently

Core attributes:

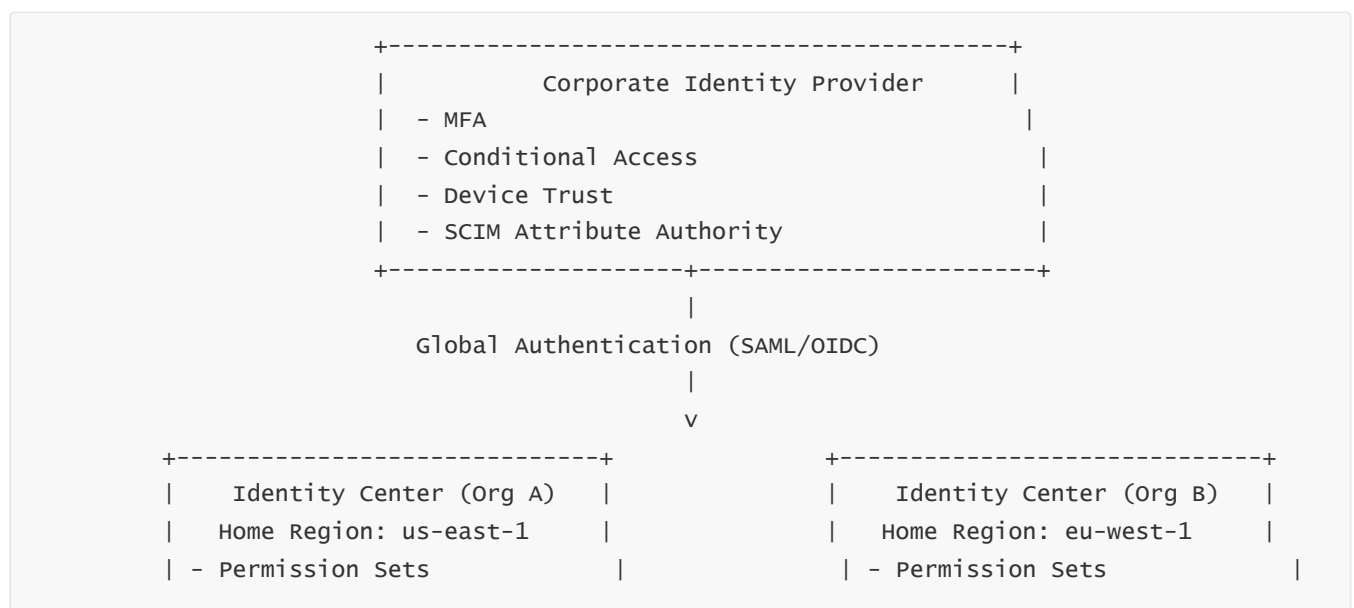
- department
- businessUnit
- project
- environment
- geo
- costCenter
- privilegeLevel

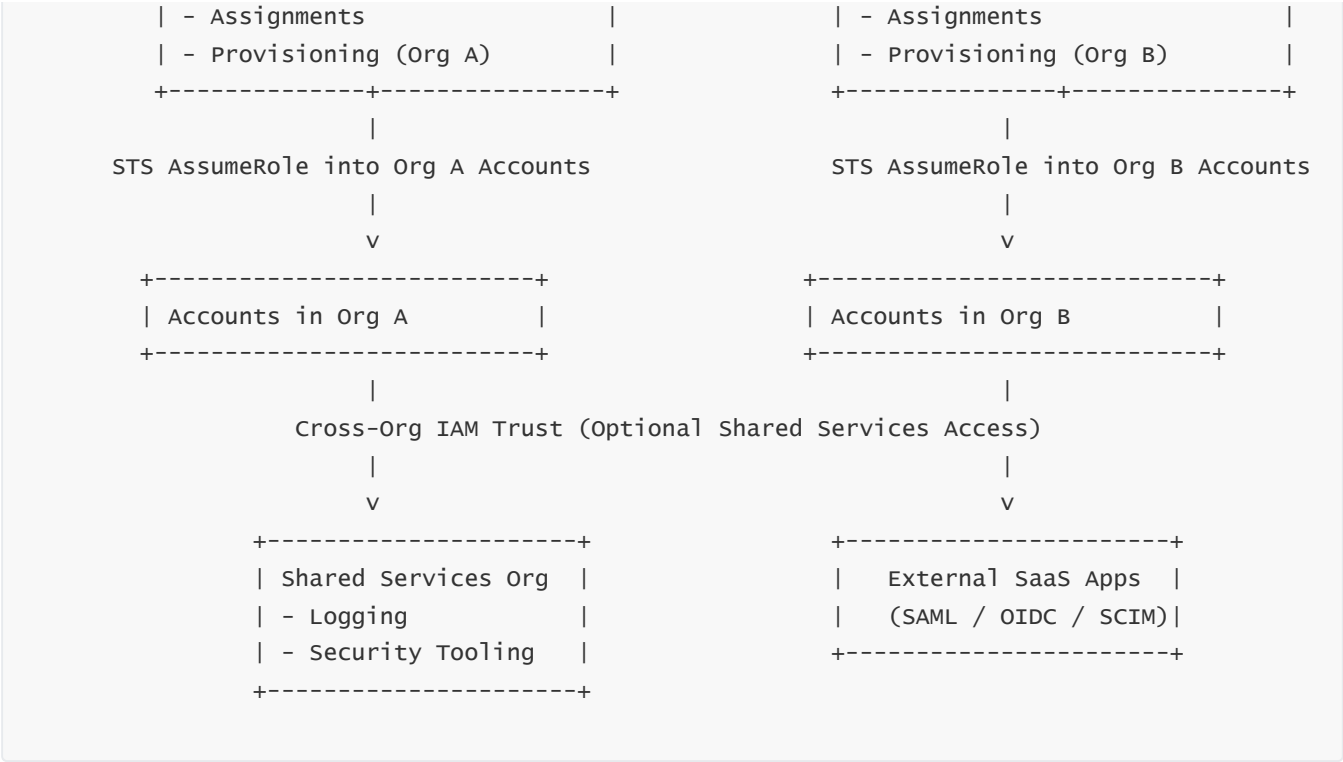
These attributes drive:

- Account access
- Permission set logic
- Application SSO
- SaaS RBAC
- Cross-Org role access

ABAC becomes **the unifying authorization model** across all clouds and apps.

8 — Advanced Global Diagram: Multi-Region + Multi-Organization Identity Fabric





This is the complete global enterprise identity model.

9 — Zero-Trust Across Regions and Organizations

Zero-trust identity must be enforced across:

- IdP
- Identity Center
- STS
- IAM
- Accounts
- Applications
- Regions
- Organizations

Principles:

- **Continuous authentication**
- **Short-lived STS tokens**
- **Attribute-based authorization**
- **Per-Org guardrails**
- **Per-region monitoring**
- **SCP guardrails**

- **Least privilege permission sets**
- **Re-authentication for privileged roles**
- **Do not trust network location**
- **Token binding to device & user**

Identity Center is where zero-trust identity posture originates.

10 — Enterprise-Scale Performance Considerations

Scale Dimensions

Identity Center can support:

- Tens of thousands of users
- Hundreds of thousands of assignments
- Thousands of permission sets (not recommended)
- 1000+ AWS accounts
- Millions of STS role assumptions per day
- Large SCIM workloads

Performance factors

- SCIM throughput
- Provisioning latency
- Group size
- Attribute complexity
- Policy size
- Number of permission sets
- Session duration policies

Scaling Recommendations

- Use medium-size groups (100–1000 users)
 - Avoid giant groups (> 10,000 users)
 - Limit permission sets
 - Use OU-based automation
 - Use IaC for assignments
 - Use batching for SCIM updates
 - Run drift detection regularly
-

11 — Disaster Recovery and Failover Strategies

Identity Center itself does not offer multi-region DR.

But you can build DR at:

11.1 — IdP Layer

- Multi-region Entra/Okta
- Backup MFA system
- Conditional access redundancy

11.2 — Cross-Org fallback

- Failover to breakglass IAM roles in each Org
- Backup Identity Center instance in separate Org
- Cross-Org IAM roles usable in emergencies

11.3 — App Access DR

- Local SAML/OIDC direct integration with IdP (bypassing Identity Center)
- Identity Center SSO not required for emergency app access

Identity Center is one part of a larger identity ecosystem — DR must be built holistically.

19 — Full Consolidated Master Summary of AWS IAM Identity Center (Unified Mega Summary)

AWS IAM Identity Center (formerly AWS SSO) is the core identity orchestration layer for the entire AWS environment and extended SaaS ecosystem. Its purpose is to unify human identities, enterprise applications, AWS accounts, permission management, lifecycle automation, and zero-trust access control under a single consistent identity plane. Identity Center does not authenticate users by itself; instead, it delegates authentication to an external corporate identity provider such as Entra ID, Okta, Ping, or AWS Managed AD, and builds its entire authorization fabric from SCIM-provisioned identities and attributes. These identities are then mapped into AWS accounts, permission sets, and application SSO configurations, forming a global access grid that is fully governed, monitored, auditable, and automated through Identity Center.

At the foundation of Identity Center is the **Identity Store**, a regionally anchored construct that receives user and group objects from SCIM provisioning. Every user's attributes—email, display name, username, department, project identifiers, organizational attributes, business unit identifiers, cost centers, ABAC fields—arrive from the IdP through SCIM. These attributes become the canonical basis for authorization across AWS accounts, SaaS apps, internal apps, and ABAC-driven IAM policies. Groups serve as the primary control surface: users are not assigned directly to permission sets or applications. Instead, group membership at the IdP governs all access, providing deterministic, scalable, audit-friendly authorization lifecycles. SCIM ensures

continuous joiner-mover-leaver integrity, synchronizing attributes, disabling terminated users, and updating group membership in real time.

Identity Center then projects these identities into **AWS accounts** using permission sets. A permission set is a declarative template that defines what level of access a persona has within each AWS account—policies, session duration, boundary conditions, tags, and ABAC context. Assignments connect groups or users to permission sets and accounts. When an assignment is created, Identity Center provisions an IAM role inside each target account, named `AWSReservedSSO_<PermissionSet>_<hash>`, with a trust policy binding it to Identity Center's regional instance. This role becomes the concrete enforcement point for AWS access, with temporary STS credentials issued whenever a user launches that role. The permission set is the desired state; the IAM role is the implemented state; STS credentials are the active state. Identity Center ensures that IAM roles match the permission set, re-provisioning them when permission sets change or when drift occurs.

Identity Center also governs **SSO for external applications** using SAML 2.0 or OIDC. When a user launches a SaaS application from the Identity Center portal, Identity Center transforms the user's attributes into SAML assertions or OIDC JWT claims, signs and delivers tokens, and executes the appropriate authentication flow. SCIM can simultaneously provision and deprovision users in these applications, ensuring lifecycle governance beyond AWS. This produces a unified identity fabric: IdP → Identity Center → AWS Accounts + SaaS Apps + Internal Apps. SAML apps receive XML assertions; OIDC apps receive JWTs; SCIM-enabled apps receive create/update/deactivate lifecycle sync; AWS accounts receive STS credentials. All of this is centrally controlled from a single access portal.

Operationally, Identity Center is the beating heart of enterprise cloud identity. It unifies access request workflows, identity governance, lifecycle automation, OU-based account access propagation, permission modeling, and cross-functional operational guardrails. When a new AWS account is created—through Control Tower, Organizations API, or custom pipelines—Identity Center automatically provisions required roles, applies permission sets, and enforces group mappings. When identities change at the IdP (department change, project change, promotion, relocation), SCIM pushes the update to Identity Center, which re-evaluates assignments and re-provisions AWS roles. Offboarding is nearly instantaneous: disabling the IdP identity removes AWS access, invalidates SSO sessions, and makes all STS tokens inert.

Identity Center observability is multi-layered. Authentication logs originate from the IdP; Identity Center produces admin and access logs; STS generates credential issuance logs; CloudTrail records API activity; service-level logs provide data-plane granularity. A complete audit trail looks like: Corporate IdP login → Identity Center assignment evaluation → STS AssumeRole → CloudTrail API logs → Resource-level logs. Every action taken by a user in AWS is tied back to an Identity Center session tag, allowing forensic traceability and policy enforcement. Monitoring Identity Center requires correlating all five layers: IdP, SCIM, Identity Center, STS, CloudTrail, and resource logs.

Identity Center troubleshooting follows a strict dependency hierarchy: authentication at IdP → SCIM provisioning → assignment mapping → IAM role provisioning → STS credential issuance → CloudTrail API behavior. Failures in Identity Center are usually rooted upstream in IdP authentication, SCIM attribute mismatches, missing group memberships, or Organizational SCP Deny effects. CLI login failures usually stem from token cache corruption, incorrect configuration, local clock skew, VPN interference, or stale device authorization flows. Drift detection is essential, since IAM roles inside AWS accounts can be modified manually, blocked by SCPs, or left partially provisioned when accounts are unreachable.

At enterprise scale, Identity Center becomes a platform. Large organizations run multiple Identity Center instances across several AWS Organizations, often with a shared corporate IdP federating identities consistently. In multi-Org setups, Identity Center can either operate per-Org, or one Org can act as a hub that uses cross-Org IAM trust to access accounts in other Orgs. In massive global environments, ABAC becomes the

backbone of authorization, aligning project, department, geography, cost center, and business unit attributes across apps, AWS accounts, and multiple Orgs. Because Identity Center is regional, global users rely on the IdP's worldwide edge footprint to reduce latency. All advanced enterprises build IdP redundancy, multiple MFA methods, breakglass identities, SCIM failover paths, and automated drift correction systems.

By combining SCIM, SAML, OIDC, and STS, Identity Center forms a **unified Zero-Trust Identity Fabric**. Authentication is performed by the IdP, authorization by Identity Center, enforcement by IAM and STS, and visibility by CloudTrail and SIEM platforms. Least privilege is modeled through permission sets and ABAC; conditional access is applied at the IdP; short-lived STS tokens enforce continuous verification; SAML/OIDC tokens follow strict audience and signature validation rules. Every identity movement—from login to AWS account access to SaaS application launch—is bound to a single underlying identity graph.

Identity Center is not simply a login system; it is the central nervous system of AWS enterprise identity. It connects people, attributes, applications, AWS accounts, policies, lifecycle, and governance under one cohesive architecture. It scales to thousands of accounts, tens of thousands of users, and millions of daily STS assumptions. With Identity Center in place, the entire AWS environment becomes identity-driven, attribute-driven, policy-driven, and fully observable—providing a hardened, consistent, zero-trust identity layer across clouds, accounts, teams, workloads, and applications. This is the complete picture of AWS IAM Identity Center as an enterprise identity fabric.

20 — Misconceptions, Pitfalls, Architecture Mistakes & How to Avoid Them

Identity Center is a simple service on the surface — a single sign-on layer for AWS.

But at enterprise scale, Identity Center becomes a deeply interconnected identity fabric bridging multiple systems: IdP, SCIM, AWS Organizations, IAM roles, OIDC, SAML, and application provisioning pipelines. Misunderstanding even one layer often leads to large-scale outages or incorrect privilege models.

The following consolidated explanation covers the most critical misconceptions and the failures they lead to, with corrective patterns and architecture safeguards.

1 — Misconception: “Identity Center is an Authentication Service.”

Identity Center **does not** authenticate anyone.

It only **brokers** authentication from the corporate IdP.

Mistake:

- Thinking Identity Center controls MFA
- Thinking Identity Center enforces password policies
- Thinking Identity Center determines login context

Reality:

- External IdP fully controls authentication

- Identity Center only evaluates authorization after authentication
- MFA, Conditional Access, Risk Engine, Device Trust all live in IdP

Correct approach:

- Enforce strict MFA, risk-based and conditional access in IdP
- Treat Identity Center as an authorization fabric, not an authentication engine
- Never try to push authentication responsibilities onto Identity Center

2 — Misconception: “SCIM is optional.”

When SCIM is missing, everything breaks:

- Users appear incorrectly
- Group membership becomes stale
- Offboarding fails
- Movers retain old access
- Managers forget to remove permissions
- Duplicate user objects appear
- ABAC attributes become inconsistent

SCIM is non-negotiable for enterprises.

Correct approach:

- Always enable SCIM
- Corporate IdP must be the identity authority
- No manual Identity Center user creation
- No manual attribute edits
- SCIM token rotation must be scheduled
- Monitor provisioning failures via CloudWatch/SIEM

3 — Misconception: “Permission Sets are just IAM policies.”

Actually, permission sets represent the **entire identity governance model**.

Common mistakes:

- Too many permission sets
- Mixing Dev/QA/Prod into same permission set
- Creating permission sets per-team instead of per-persona
- Manually editing IAM roles created by Identity Center

- Using inline policies larger than 10 KB causing provisioning failures

Correct approach:

- Use persona-based permission sets
 - Use ABAC to avoid explosion in permission sets
 - Never manually modify AWSReservedSSO roles
 - Maintain permission sets entirely via IaC
 - Keep permission sets small, stable, version-controlled
-

4 — Pitfall: Manual Assignments (User → Account → Permission Set)

Manual assignments destroy scalability.

Problems caused:

- Hundreds of exceptions
- Drift between IdP groups and AWS access
- Impossible-to-audit access maps
- Offboarded user still has assignments
- Human error in creating/removing permissions
- Approval workflows bypassed

Correct approach:

- Only group-based assignments
 - Never assign a permission set directly to a user
 - All assignments must be managed via IaC
 - IdP groups drive all entitlement mapping
 - Enforce strict assignment governance
-

5 — Misconception: “Identity Center roles are global IAM roles.”

They’re not.

Identity Center generates roles **inside each AWS account**.

Mistake:

- Assuming permission sets automatically take effect everywhere
- Forgetting to re-provision changed permission sets
- Not checking for role drift

- Manually deleting Identity Center roles

Correct approach:

- Understand roles are per-account artifacts
 - Re-provision permission sets after every change
 - Use drift detection automation
 - Recreate roles automatically
 - Never modify Identity Center IAM roles manually
-

6 — Pitfall: SCPs Blocking Identity Center Provisioning

Enterprises often misconfigure SCPs, causing:

- Failed role creation
- Failed inline policy updates
- Failed trust policy updates
- Regional provisioning failures
- Multi-account drift

Symptoms:

- “Provisioning failed” in Identity Center
- IAM Deny logs in CloudTrail
- Incomplete SSO role propagation

Correct approach:

- Always build SCPs with allow-list for Identity Center IAM operations
 - Use exception SCPs for provisioning
 - Review SCP impact before rollout
 - Test new SCPs with sandbox accounts
-

7 — Misconception: “Identity Center Can Span Multiple AWS Organizations.”

Identity Center ties to **exactly one** Organization.

Mistake:

- Expecting one Identity Center to control dozens of Orgs
- Trying to synchronize permission sets across Orgs
- Assuming SSO portal is shared across all Orgs

Correct approach:

- One Identity Center per Org
- Use cross-Org IAM trust for shared services access
- Maintain uniform IdP attribute schema across Identity Centers
- Use centralized IdP as the single identity source

8 — Pitfall: Not Treating Identity Center as a Regional Service

Identity Center is anchored to one region.

Common mistakes:

- Expecting multi-region failover
- Assuming regional resiliency
- Not designing for region outages
- Global teams experience slow login

Correct approach:

- Make IdP globally distributed
- Use multi-region MFA
- Leverage IdP Conditional Access to optimize login paths
- Document Identity Center regional failover limitations
- Implement breakglass IAM roles not dependent on Identity Center

9 — Misconception: “CLI login is just `aws sso login`.”

CLI login is an entire OIDC device authorization flow.

Common pitfalls:

- Clock skew
- Corrupted OIDC cache
- Invalid device verification codes
- Old CLI version
- Wrong start URLs
- Mismatched regions
- VPN blocks OIDC endpoints

Correct approach:

- Use latest AWS CLI v2
 - Validate OIDC endpoints
 - Clean `.aws/sso/cache` when corrupted
 - Ensure corporate VPN allows Identity Center endpoints
 - Use device code flow correctly
-

10 — Pitfall: Not Managing SAML Certificate Rotation

In SAML integrations:

- Certificate expires
- Applications fail SSO
- Assertion signatures invalid
- Users locked out globally

Correct approach:

- Track SAML certificate expiration
 - Rotate proactively
 - Update SP metadata accordingly
 - Store cert metadata in IaC
 - Test new SAML certs in staging apps
-

11 — Pitfall: Wrong Attribute Mapping for Apps

Misconfigured attributes cause:

- Wrong roles in Salesforce/Slack/Jira
- Incorrect group membership
- Missing user IDs
- Broken SCIM provisioning
- Broken ABAC authorization
- Duplicate user accounts

Correct approach:

- Use canonical attribute schema
- Guarantee SCIM attributes match IdP attributes
- Validate attribute mappings regularly

- Keep attribute names stable during reorganizations
-

12 — Misconception: “Identity Center Automatically Handles Application RBAC.”

Apps must implement RBAC using:

- SAML attributes
- OIDC claims
- SCIM-provisioned roles
- Internal app-specific mapping logic

Identity Center only provides identity and attributes.

The app still enforces its own roles and permissions.

Correct approach:

- Work with SaaS admins to align RBAC with Identity Center attributes
 - Use SCIM to manage app user roles
 - Maintain identity schema consistency
-

13 — Pitfall: Too Many Permission Sets or Too Many Groups

This leads to:

- SCIM delays
- Assignment bloat
- Provisioning failures
- Long identity propagation times
- Human confusion

Correct approach:

- Use persona-based roles
 - Use ABAC to reduce need for dozens of roles
 - Consolidate groups
 - Maintain strict naming conventions
 - Remove unused permission sets regularly
-

14 — Pitfall: Manual IAM Role Modification

Admins sometimes modify:

- Inline policies
- Trust policies
- Session duration
- Session tags
- AssumeRole conditions

This breaks Identity Center's authority.

Correct approach:

- Treat AWSReservedSSO roles as immutable
- Re-provision permission sets whenever drift occurs
- Use permission boundaries if needed
- Detect drift via automation

15 — Misconception: "Identity Center Logs Are Enough for Auditing."

Identity is multi-layered.

You must correlate:

- IdP login logs
- Identity Center admin logs
- Identity Center SSO access logs
- STS assume-role logs
- CloudTrail logs
- Resource-level logs

Only then do you get full observability.

Correct approach:

- Send all logs to SIEM
 - Correlate identity → credential → API call → resource access
 - Build UEBA models for identity risk
 - Monitor SCIM drift and failures
-

16 — Pitfall: Not Using Infrastructure as Code (IaC)

Manual operations cause:

- Drift
- Incorrect assignments
- Stale permission sets
- SAML/OIDC misconfigurations
- Inconsistent environments

Correct approach:

- Terraform or CDK for:
 - Permission sets
 - Assignments
 - SAML metadata
 - OIDC apps
 - SCIM config
 - SSO admin API automation

17 — Misconception: “Identity Center is Simple — No Need for Governance.”

Identity Center is a **tier-0 service**.

A failure equals:

- No AWS console access
- No CLI access
- No deployment pipelines
- No incident response operations
- No developer productivity

Correct approach:

- Strict change management
 - Access review policies
 - Backup IdP admin access
 - Breakglass IAM roles
 - SCIM failover runbooks
 - Identity Center downtime procedures
-

18 — Pitfall: Not Designing for Joiner-Mover-Leaver Flow

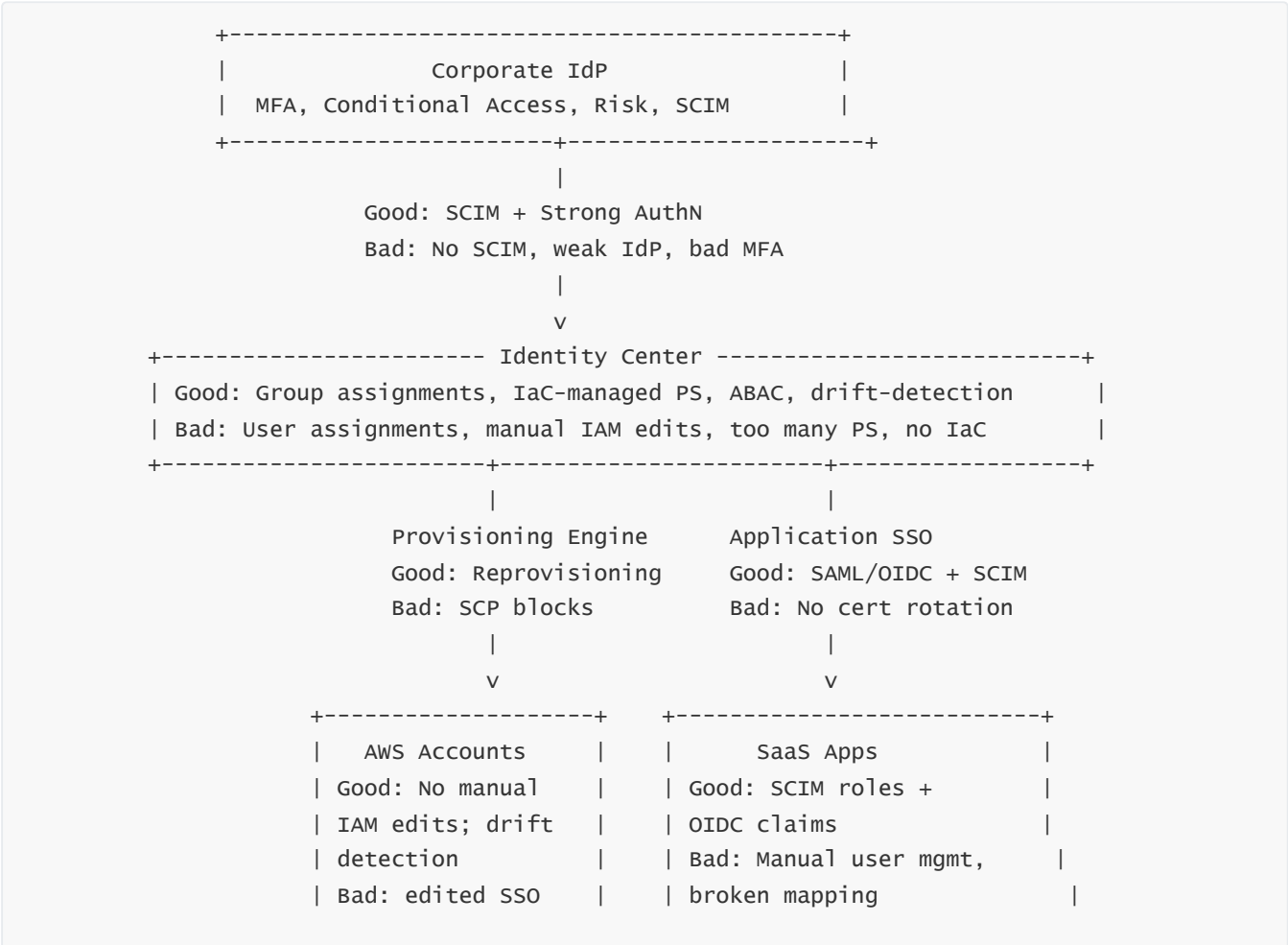
Enterprises with no JML design face:

- Old employees retaining access
- New employees not onboarded
- Movers having incorrect privileges
- Shadow groups created in IdP
- Identity duplication in apps

Correct approach:

- Full lifecycle automation through SCIM
- IdP-driven group membership
- Attribute-driven ABAC
- Remove manual steps completely

19 — Ultimate Mega-Diagram: Identity Center Pitfalls and Correct Architecture



roles, SCP Deny	+-----+
+-----+	

This diagram summarizes almost every enterprise Identity Center mistake.

20 — Final Mental Model: How to Never Break Identity Center Again

Identity Center succeeds when:

- IdP is authoritative
- SCIM is fully enforced
- Groups drive everything
- Permission sets are minimal and persona-based
- IAM roles are never manually edited
- Assignments are IaC-controlled
- SAML/OIDC are consistently mapped
- Drift detection runs continuously
- Logs flow into SIEM
- SCPs allow provisioning
- JML lifecycle is automated
- Zero Trust enforced at IdP + STS

Identity Center fails when:

- Authentication, provisioning, authorization, and IAM roles become inconsistent
- Multiple sources of truth appear
- Manual operations override provisioning
- SCPs block required IAM actions
- Certificates or JWKS expire silently
- Groups and attributes drift between IdP and Identity Center
- Region or Org design is misunderstood
- Logging and monitoring are not correlated

Identity Center is most powerful when treated not as a convenience feature but as an **enterprise identity platform**, governed with discipline, automation, and strict architectural control.

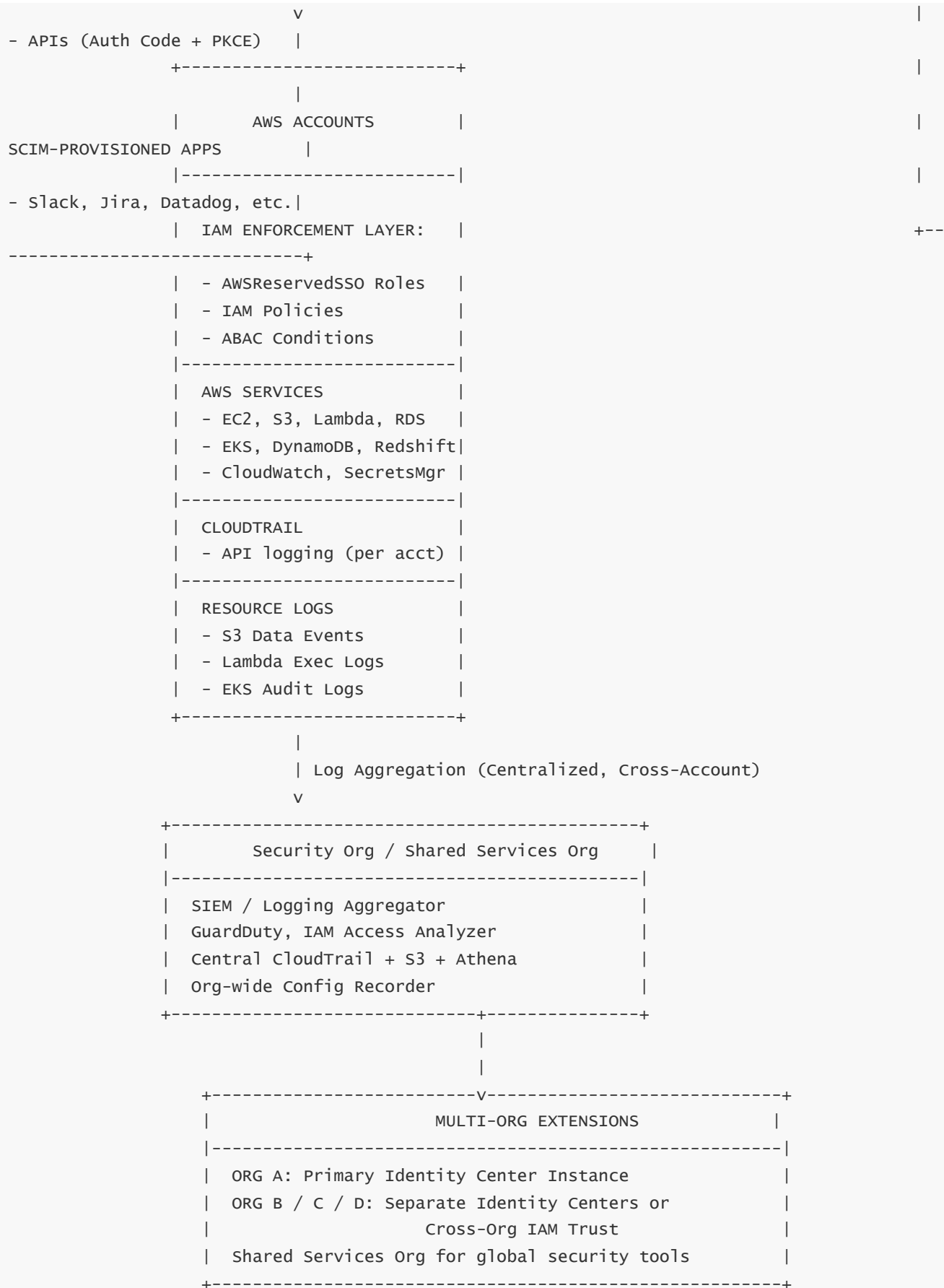
When all layers are aligned — IdP, SCIM, Identity Center, STS, IAM, apps, Organizations, permissions, logs — the entire cloud becomes identity-driven, zero-trust, deterministic, secure, and fully observable.

FINAL COMBINED MEGA-DIAGRAM (IDENTITY CENTER END-TO-END)

(70× depth architecture diagram)







FINAL MEGA-DIAGRAM EXPLANATION (Unified Complete Narrative)

Below is the complete multi-layer explanation of the entire Identity Center architecture, with every system stitched together into one cohesive fabric.

1 — Identity Origin: Corporate IdP (Top Layer)

The entire identity lifecycle begins outside AWS.

The corporate IdP is the single point of:

- Authentication
- MFA enforcement
- Passwordless or conditional access
- Device trust
- Geo restrictions
- Sign-in risk scoring
- Identity attributes
- Group membership
- Lifecycle triggers (joiner/mover/leaver)
- SCIM provisioning into AWS

Identity Center trusts the IdP entirely — it never authenticates users itself.

SCIM pushes:

- Users
- Groups
- Group memberships
- Identity attributes for ABAC
- Activation/Deactivation status

This forms the **identity authority**.

2 — Identity Center: The Authorization Fabric

Identity Center receives identities from SCIM and applies enterprise authorization rules:

Identity Store

Holds all the users and groups.

Attributes stored here drive:

- ABAC

- Application SSO claim mapping
- Permission filtering
- Session tagging
- Role scoping

Permission Sets

The central IAM templates controlling AWS access:

- IAM inline or managed policies
- Session duration
- ABAC session tags
- Boundary conditions
- Fine-grained least privileges

Assignments

Link:

User/Group → Permission Set → Accounts

Assignments are the **control plane** of AWS access.

SAML & OIDC Applications

Identity Center also acts as a SAML/OIDC IdP for SaaS and internal applications:

- SAML assertions with x.509 signing
- OIDC JWTs with JWK rotation
- Application attribute mapping
- Role mapping via attributes
- Multi-tenant SaaS integration
- Internal dashboards / portals
- API access using PKCE/OIDC

Provisioning Engine

Identity Center provisions IAM roles into AWS accounts:

AWSReservedSSO_<PermissionSet>_<Hash>

It ensures IAM roles are:

- Created correctly
- Updated on permission set change

- Redeployed on drift
- Restored if manually modified
- Auto-provisioned for new accounts

Identity Center is the **enterprise authorization brain**.

3 — STS: The Identity Execution Engine

When a user chooses a role:

- Identity Center calls STS
- STS issues **short-lived credentials**
- Principal tags are stamped for ABAC
- Session identity becomes “AssumedRole”
- Role session name reflects user identity

STS tokens are:

- Region-independent
- Temporary
- Zero-trust aligned
- Traceable
- Impossible to misuse for long periods

All AWS actions go through STS.

4 — AWS Accounts: Enforcement Layer

AWS accounts enforce permissions and record actions.

IAM Roles

Identity Center–provisioned IAM roles enforce:

- Least privilege
- ABAC policy filters
- Permission boundaries
- Session tagging policies
- Trust relationships that restrict access to Identity Center

AWS Services

All actions taken by the user occur here:

- EC2
- S3

- Lambda
- EKS
- RDS
- DynamoDB
- KMS
- CloudWatch
- CloudFormation
- Secrets Manager
- Redshift
- Glue
- Athena
- etc.

CloudTrail

Every API call is logged with:

- Role assumed
- User identity
- SSO session info
- Principal tags
- Source IP
- Access context

Resource Logs

Data-plane logs:

- S3 object events
- Lambda execution
- EKS audit logs
- API Gateway logs
- DynamoDB streams

Together these logs build **end-to-end forensic visibility**.

5 — Centralized Ops: Shared Services / Security Org

Most enterprises operate:

- A **Security Organization Unit (OU)**
- A **Shared Services Org**

Here they centralize:

- SIEM (Splunk, QRadar, Datadog, Chronicle)
- GuardDuty/Access Analyzer findings
- CloudTrail Lake or S3 logs
- Multi-account log aggregation
- Organization Config rules
- Incident response tooling
- IAM Access Analyzer
- Monitoring dashboards

Identity Center logs flow into the SIEM where logs from:

- IdP
- Identity Center
- STS
- CloudTrail
- Resource logs

are correlated into a single identity picture.

6 — Multi-Organization Extensions

Identity Center is bound to one Organization.

Other Organizations integrate via:

- Their own Identity Center
- Cross-Org IAM Trust
- Hub-and-spoke identity
- Shared Services Org for global tooling
- Federated identity policy mapping

Identity stays central; permissions distribute globally.

7 — Zero-Trust Identity Fabric Summary

Identity Center builds a zero-trust access fabric:

- Authentication: IdP MFA, risks, and device checks
- Authorization: Identity Center assignments + ABAC
- Enforcement: IAM & STS
- Observability: CloudTrail + logs
- Lifecycle: SCIM-driven
- Token security: STS + JWT + x.509

- Application integration: SAML/OIDC
- Multi-account: AWS Organizations provisioning
- Multi-org extension: trust-based federation
- Multi-region: IdP global edges + regional enforcement

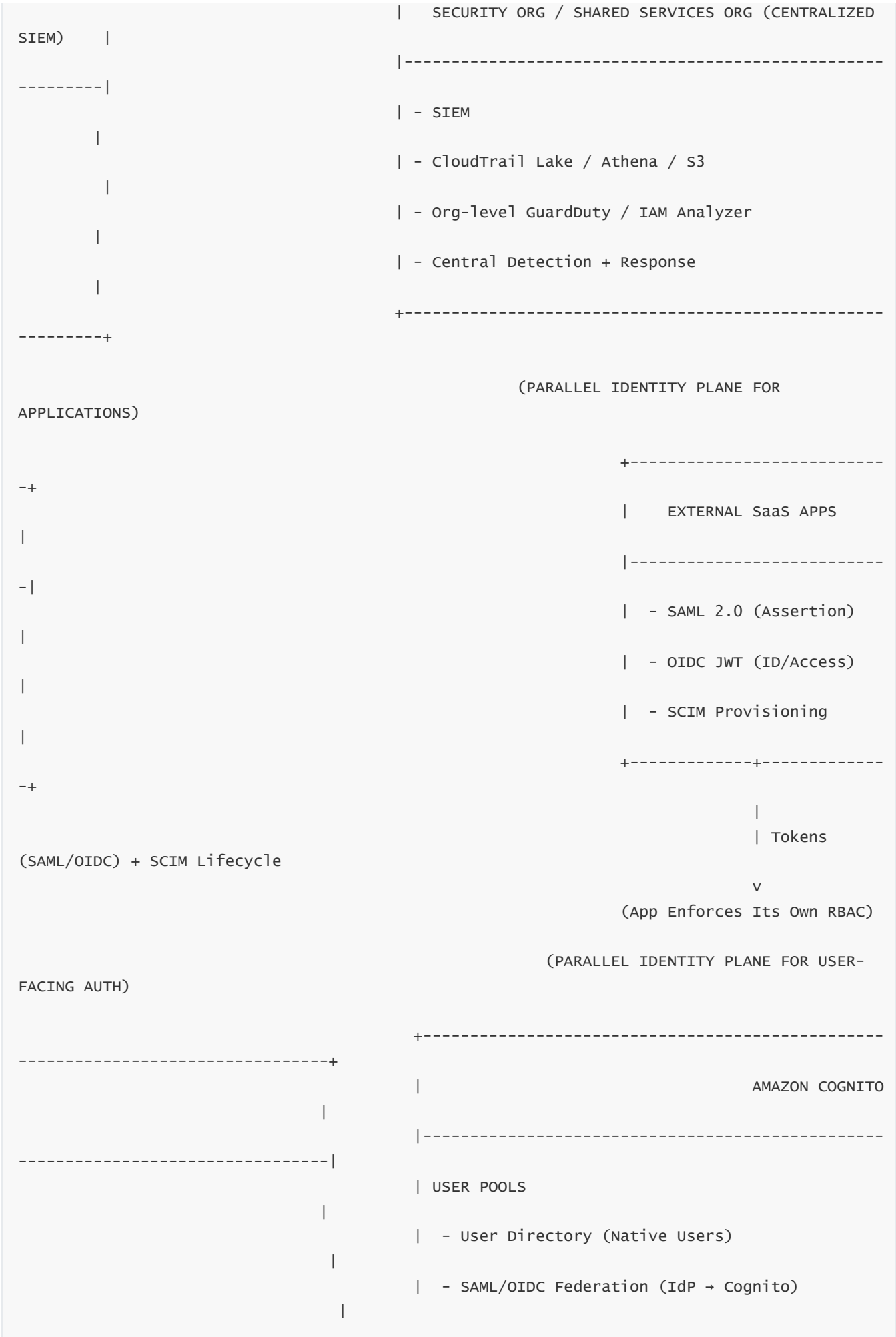
Identity is continuously verified and access is short-lived and attribute-driven.

THE ABSOLUTE FINAL MEGA-DIAGRAM OF ALL AWS IDENTITY SYSTEMS

(70× depth architecture diagram — IAM + Identity Center + Cognito + IdP + STS + AWS Accounts)









COMPLETE EXPLANATION OF THE MEGA-DIAGRAM (Unified Identity Architecture)

Below is the fully integrated narrative — a single end-to-end explanation of how **IAM**, **Identity Center**, **Cognito**, **STS**, and the **corporate IdP** interact together to form the complete AWS identity ecosystem.

1 — Corporate IdP: The Single Source of Truth

All identities originate here:

- Human users
- Contractors
- Partners
- Service accounts (sometimes)
- Groups

- Attributes
- MFA and conditional access

The corporate IdP provides:

- **SCIM** → lifecycle provisioning into Identity Center
- **SAML/OIDC** → authentication for SSO and Cognito federation

This is the top of the identity pyramid.

2 — Identity Center (SSO): AWS's Human Authorization Brain

Identity Center receives:

- Users
- Groups
- Attributes (ABAC)

via SCIM.

Then provides:

- Permissions for **AWS accounts** via permission sets
- SSO access to **SaaS apps** via SAML
- OIDC login for **custom/internal apps**
- Full lifecycle-driven access mapping
- IAM role provisioning into every AWS account

Identity Center = **central authorization system** for all human access.

3 — Cognito: Application-Facing Identity Layer

Cognito handles:

- App sign-up/sign-in
- MFA for app users
- OIDC/SAML federation from IdPs
- Identity Pools → STS for temporary AWS credentials

Identity Center = human AWS access

Cognito = app user access (web/mobile/IoT)

These two systems never conflict — they are parallel.

4 — STS: The Identity Execution Engine

Everything results in STS issuing:

- Short-lived credentials
- Assumed-role identities
- Principal tags for ABAC
- Secure, temporary access

STS is the trust execution layer of identity.

5 — IAM Roles & Policies: The Enforcement Layer

IAM performs:

- Authorization enforcement
- Policy evaluation
- Resource access decisions
- ABAC condition matching
- Permission boundaries
- Service and trust behavior

Identity Center creates IAM roles.

Cognito maps identities to IAM roles.

STS supplies credentials.

IAM enforces.

6 — AWS Accounts & Services: Where Access Happens

All compute, storage, data, and operations occur here:

- EC2
- S3
- Lambda
- EKS
- RDS
- DynamoDB
- Glue
- Redshift
- SQS/SNS
- KMS
- CloudWatch

- Etc.

IAM controls everything users and applications do here.

CloudTrail logs everything.

7 — SaaS + Internal Applications

Applications rely on:

- SAML assertions from Identity Center
- OIDC JWT tokens from Identity Center
- SCIM provisioning from Identity Center

Applications maintain their own RBAC using attributes and groups from Identity Center.

8 — Security Org: Centralized Observability Layer

All logs flow here:

- IdP authentication
- SCIM changes
- Identity Center access/admin logs
- STS token issuance logs
- CloudTrail API activity
- Resource-level logs

This allows full identity-based threat detection.

9 — Multi-Org & Multi-Region Expansion

Identity Center binds to **one Org** and **one region**, but identity fabric extends:

- Across Organizations via IAM trust
- Across regions via global IdP
- Across apps via SAML/OIDC
- Across workloads via STS and IAM

This creates a **single unified identity plane** across the entire cloud estate.
