

SETTING UP A MULTI-ACCOUNT AWS ENVIRONMENT USING AWS ORGANIZATIONS, IAM IDENTITY CENTER (FOMERLY SSO) AND TERRAFORM.

Index

1. Overview
2. Prerequisites
3. Terraform Directory Structure
4. Step-by-Step Breakdown
 - Organization Setup
 - Account Creation (Module-Based)
 - AWS SSO and Permission Sets
 - Group and User Setup
 - Account Assignments
 - Service Control Policy (SCP)
5. Cost and Management Considerations
6. Security Best Practices
7. Proof of Concept

1. Overview

This documentation outlines how to set up a multi-account AWS environment using **AWS Organizations**, **SSO**, and **Terraform**. It includes automated account creation, permission set assignments, and security policies. It uses a **modular approach** to provision reusable and clean infrastructure code.

2. Prerequisites

Before deploying:

- Terraform installed
- AWS CLI configured
- Administrative IAM credentials with organizations:*, sso:*, and iam:* permissions

- **NB:** With SSO(IAM Identity Center) you need explicit permission grants attached to the user being used to provision these resources with Terraform even if you have the AdministrativeAccess policy attached to this user.

You can use the AWS managed policy **Policy ARN:**

arn:aws:iam::aws:policy/AWSSSOServiceRolePolicy or search for these in IAM console:

AWSSSOServiceRolePolicy

AWSSSODirectoryAdministrator

AWSIdentityStoreServiceRolePolicy OR

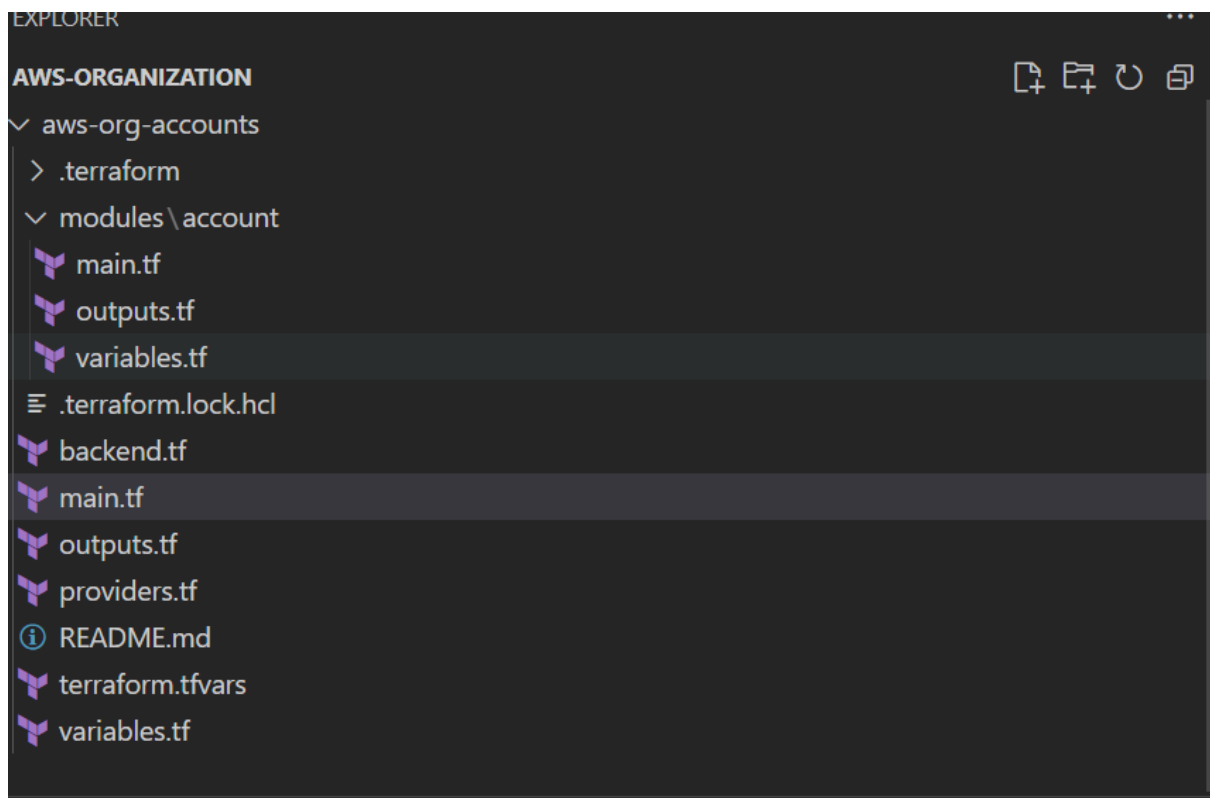
create a custom policy like:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sso:*",
        "sso-admin:*",
        "identitystore:*"
      ],
      "Resource": "*"
    }
  ]
}
```

and attach it to the user.

This is AWS security design to prevent accidental SSO modifications.

3. Terraform Directory Structure

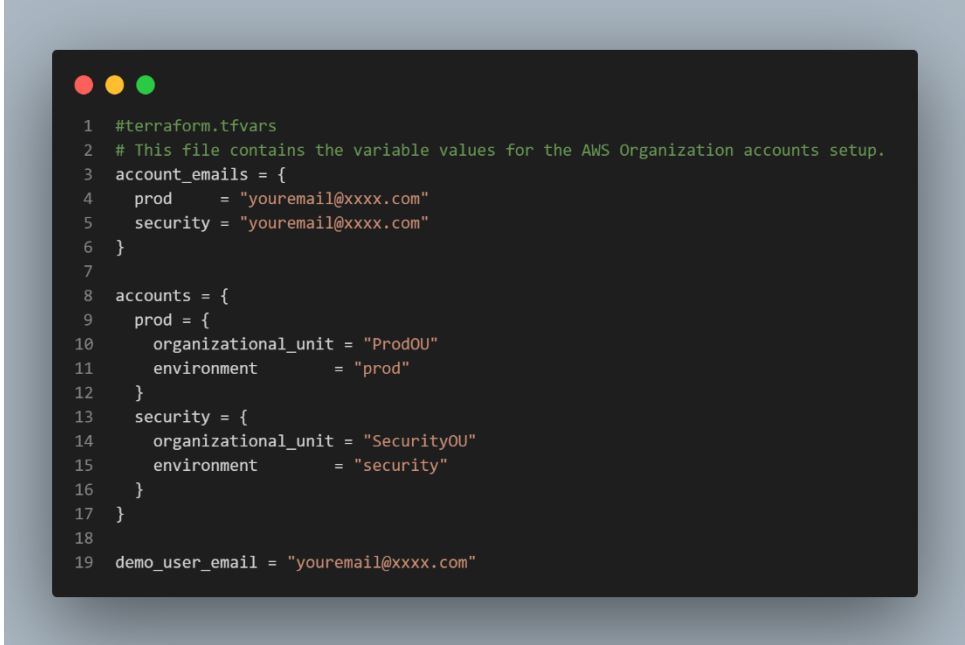


aws-org-project/

```
├─ main.tf      # Entry point for creating the org, accounts, policies
├─ variables.tf
├─ outputs.tf
├─ modules/
│   └─ account/
│       ├── main.tf  # Creates OUs and accounts
│       └─ variables.tf
```

VARIABLES

The snippet of terraform configuration below shows a terraform.tfvars file which will help you get a better understanding of this documentation. The terraform.tfvars variables contain input variables which was used in this demo.



```
1 #terraform.tfvars
2 # This file contains the variable values for the AWS Organization accounts setup.
3 account_emails = {
4   prod     = "youremail@xxxx.com"
5   security = "youremail@xxxx.com"
6 }
7
8 accounts = {
9   prod = {
10    organizational_unit = "ProdOU"
11    environment         = "prod"
12  }
13   security = {
14    organizational_unit = "SecurityOU"
15    environment         = "security"
16  }
17 }
18
19 demo_user_email = "youremail@xxxx.com"
```

These variables are:

account_emails:

- **Purpose:** Defines the email addresses for each AWS account.
- **Usage:** Each AWS account requires a unique email address.
- **Impact:** These emails receive AWS account notifications and billing alerts.

accounts:

- **Purpose:** Defines the structure and metadata for each account.
- **organizational_unit:** Name of the OU container for each account.
- **environment:** Tag applied to the account for identification.
- **Usage:** Used by the module to create accounts and place them in OU.

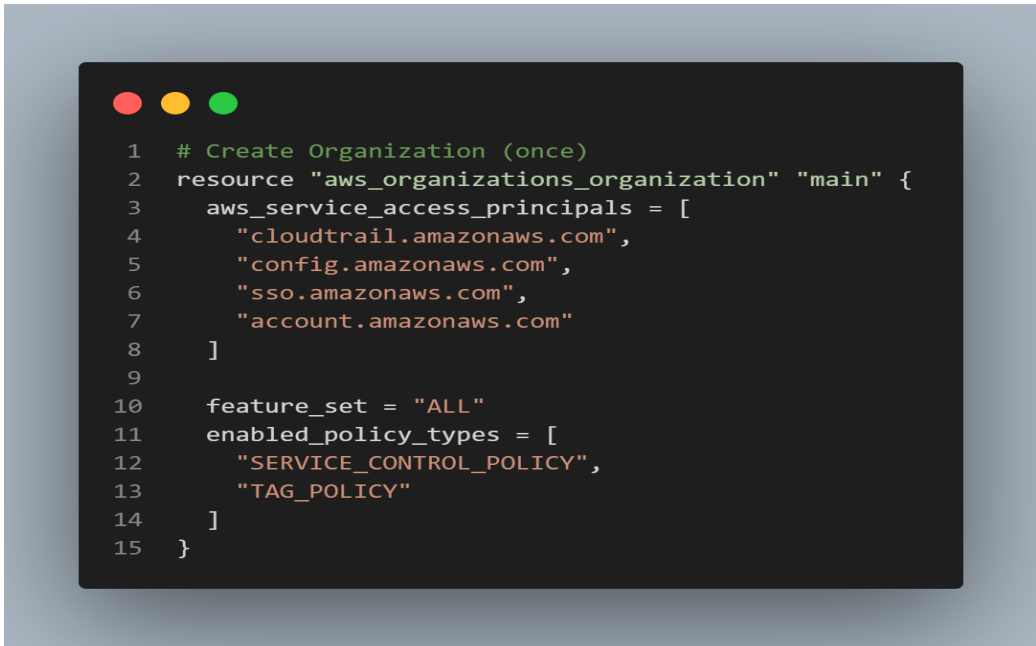
demo_user_email:

- **Purpose:** Email address for the SSO demo user.
- **Usage:** Used to create a user in IAM Identity Center (SSO).
- **Impact:** This email receives SSO login invitations.
- **Note:** Different from account emails - this is for human user access.

4. Step-by-Step Breakdown (Entrypoint's main.tf)

◆ 4.1 Organization Setup

In main.tf, we initialize AWS Organizations and enable integration with AWS services:



```
1 # Create Organization (once)
2 resource "aws_organizations_organization" "main" {
3   aws_service_access_principals = [
4     "cloudtrail.amazonaws.com",
5     "config.amazonaws.com",
6     "sso.amazonaws.com",
7     "account.amazonaws.com"
8   ]
9
10  feature_set = "ALL"
11  enabled_policy_types = [
12    "SERVICE_CONTROL_POLICY",
13    "TAG_POLICY"
14  ]
15 }
```

This creates the organization root and enables necessary service access and policy types.

Service Access Principals Enable Organization-Wide AWS Services:

CloudTrail (cloudtrail.amazonaws.com):

Organization trail - Single CloudTrail that logs API calls across ALL accounts in your organization.

Config (config.amazonaws.com):

Organization Config rules - Deploy compliance rules to ALL accounts at once.

sso.amazonaws.com - AWS Single Sign-On for centralized user access.

account.amazonaws.com - Account management service integration.

feature_set = "ALL" enables centralized account management, but you should add the service principals for complete centralization.

Service Control Policies (SCPs) - Restrict what accounts can do.

Tag Policies - Enforce consistent tagging across accounts.

◆ 4.2 Account Creation (Modular)

We use a module called `account` to create new AWS accounts under their respective Organizational Units (OUs). With Terraform modules since configuration is already setup in the `main.tf` of the module, we use the **source** attribute to call the module and **for_each** loop to create accounts using the variables set in the `.tfvars` file shown earlier.

```
1  # Create Accounts (2 accounts for demo)
2  module "accounts" {
3    for_each = var.accounts
4
5    source          = "./modules/account"
6    account_name    = title(each.key)
7    account_email   = var.account_emails[each.key]
8    organizational_unit = each.value.organizational_unit
9    environment     = each.value.environment
10   organization_root_id = aws_organizations_organization.main.roots[0].id
11 }
12
```

Inside `modules/account/main.tf`, `variables.tf` and `outputs.tf`

```
1  resource "aws_organizations_organizational_unit" "ou" {
2    name          = var.organizational_unit
3    parent_id    = var.organization_root_id
4  }
5
6  resource "aws_organizations_account" "account" {
7    name          = var.account_name
8    email         = var.account_email
9    role_name     = "OrganizationAccountAccessRole"
10   parent_id    = aws_organizations_organizational_unit.ou.id
11
12   tags = merge(
13     {
14       Environment = var.environment
15     },
16     var.tags
17   )
18 }
```

```
1  output "account_id" {
2    description = "The ID of the created AWS account"
3    value      = aws_organizations_account.account.id
4  }
5
6  output "account_arn" {
7    description = "The ARN of the created AWS account"
8    value      = aws_organizations_account.account.arn
9  }
10
11 output "organizational_unit_id" {
12   description = "The ID of the organizational unit"
13   value      = aws_organizations_organizational_unit.ou.id
14 }
```

```

1 variable "account_name" {
2   description = "The name of the AWS account to be created."
3   type       = string
4 }
5
6 variable "account_email" {
7   description = "The email address associated with the AWS account."
8   type       = string
9
10  validation {
11    condition = can(regex("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$", var.account_email))
12    error_message = "The account_email must be a valid email address."
13  }
14 }
15
16 variable "organizational_unit" {
17   description = "The organizational unit where the account will be placed."
18   type       = string
19 }
20
21 variable "environment" {
22   description = "The environment tag for the account (e.g., dev, prod)"
23   type       = string
24 }
25
26 variable "tags" {
27   description = "Additional tags to apply to the AWS account."
28   type       = map(string)
29   default    = {}
30 }
31
32 variable "organization_root_id" {
33   description = "The root ID of the organization"
34   type       = string
35 }

```

This module performs two actions:

1. Creates an **OU** using the provided name
2. Creates a new **account** under the OU with an IAM role (OrganizationAccountAccessRole)

NB: This is the module being called in the entrypoint's main.tf . **The variables.tf and outputs.tf** are necessary because without the **variables.tf**, your module would have no way to accept inputs, making it impossible to create different accounts with different names, emails, and OUs and **outputs.tf** allow your main code to reference the account IDs and OU IDs created by the module, which is essential for your SSO assignments and policy attachments.

◆ 4.3 AWS SSO & Permission Sets

We pull the SSO instance data and define permission sets for AdminAccess and ReadOnlyAccess roles. Session duration for Admin access is 8 hours and Read_only access for 4 hours. Session duration allowed range, **Minimum:** 1 hour, **Maximum:** 12 hours, if not specified default is **an hour**.

```
1  # SSO Setup
2  data "aws_ssoadmin_instances" "main" {
3    depends_on = [aws_organizations_organization.main]
4  }
5
6  locals {
7    sso_instance_arn = tolist(data.aws_ssoadmin_instances.main.arns)[0]
8    identity_store_id = tolist(data.aws_ssoadmin_instances.main.identity_store_ids)[0]
9  }
10
11 # Permission Sets
12 resource "aws_ssoadmin_permission_set" "admin_access" {
13   name           = "AdminAccess"
14   description    = "Full administrative access"
15   instance_arn   = local.sso_instance_arn
16   session_duration = "PT8H"
17 }
18
19 resource "aws_ssoadmin_permission_set" "readonly_access" {
20   name           = "ReadOnlyAccess"
21   description    = "Read-only access"
22   instance_arn   = local.sso_instance_arn
23   session_duration = "PT4H"
24 }
```

AWS managed policies are attached to these permission sets.

```
1  # Attach Policies
2  resource "aws_ssoadmin_managed_policy_attachment" "admin_policy" {
3    instance_arn       = local.sso_instance_arn
4    managed_policy_arn = "arn:aws:iam::aws:policy/AdministratorAccess"
5    permission_set_arn = aws_ssoadmin_permission_set.admin_access.arn
6  }
7
8  resource "aws_ssoadmin_managed_policy_attachment" "readonly_policy" {
9    instance_arn       = local.sso_instance_arn
10   managed_policy_arn = "arn:aws:iam::aws:policy/ReadOnlyAccess"
11   permission_set_arn = aws_ssoadmin_permission_set.readonly_access.arn
12 }
13
```


◆ 4.4 User and Group Setup

We create an Identity Store user and assign them to a group (SecurityTeam), which can be granted access to specific accounts. The policies associated with this group will affect the users assigned to this group.

```
1  # Create Group
2  resource "aws_identitystore_group" "security_team" {
3    display_name      = "SecurityTeam"
4    description       = "Security team members"
5    identity_store_id = local.identity_store_id
6  }
7
8  # Create Users
9  resource "aws_identitystore_user" "demo_user" {
10   identity_store_id = local.identity_store_id
11   display_name      = "Demo User"
12   user_name         = "demo.user"
13
14   name {
15     given_name = "Demo"
16     family_name = "User"
17   }
18
19   emails {
20     value = var.demo_user_email
21     primary = true
22   }
23 }
24
25 # Add User to Group
26 resource "aws_identitystore_group_membership" "demo_user_membership" {
27   identity_store_id = local.identity_store_id
28   group_id          = aws_identitystore_group.security_team.group_id
29   member_id         = aws_identitystore_user.demo_user.user_id
30 }
31
```

◆ 4.5 Account Assignments

```
1 # Account Assignments
2 resource "aws_ssoadmin_account_assignment" "security_team_to_security_account" {
3   instance_arn      = local.sso_instance_arn
4   permission_set_arn = aws_ssoadmin_permission_set.admin_access.arn
5
6   principal_id = aws_identitystore_group.security_team.group_id
7   principal_type = "GROUP"
8
9   target_id = module.accounts["security"].account_id
10  target_type = "AWS_ACCOUNT"
11 }
12
13 resource "aws_ssoadmin_account_assignment" "security_team_to_prod_account" {
14   instance_arn      = local.sso_instance_arn
15   permission_set_arn = aws_ssoadmin_permission_set.readonly_access.arn
16
17   principal_id = aws_identitystore_group.security_team.group_id
18   principal_type = "GROUP"
19
20   target_id = module.accounts["prod"].account_id
21   target_type = "AWS_ACCOUNT"
22 }
23
24 resource "
```

We assign:

- **AdminAccess** to SecurityTeam for the security account
- **ReadOnlyAccess** to the same group for the production account

◆ 4.6 Service Control Policies (SCPs)

We define and attach an SCP to restrict the use of cost-heavy services like SageMaker, Redshift, and EMR. Service Control Policies (SCPs) act as guardrails that set the maximum permissions for accounts in your organization. Here's how they work:

How SCPs Work:

SCPs are DENY policies - they restrict what can be done, they don't grant permissions.

In the Setup:

SSO users with AdminAccess would still be restricted by any SCPs you attach. For example:

SSO gives: Full admin permissions

SCP denies: SageMaker, redshidft and emr access.

Result: Admin access to everything EXCEPT SageMaker, redshift and emr.

```
1  resource "aws_organizations_policy" "security_restricted_access" {
2    name           = "SecurityRestrictedAccess"
3    description    = "Prevent use of expensive AWS services"
4    type           = "SERVICE_CONTROL_POLICY"
5
6    content = jsonencode({
7      Version = "2012-10-17"
8      Statement = [
9        {
10         Effect = "Allow"
11         Action = "*"
12         Resource = "*"
13       },
14       {
15         Effect = "Deny"
16         Action = [
17           "sagemaker:*",
18           "redshift:*",
19           "emr:*"
20         ]
21         Resource = "*"
22       }
23     ]
24   })
25 }
26
27 # Attachment to target SecurityOU
28 resource "aws_organizations_policy_attachment" "security_restricted" {
29   policy_id = aws_organizations_policy.security_restricted_access.id
30   target_id = module.accounts["security"].organizational_unit_id
31
32   lifecycle {
33     create_before_destroy = true
34   }
35 }
```

5. Cost and Management Considerations

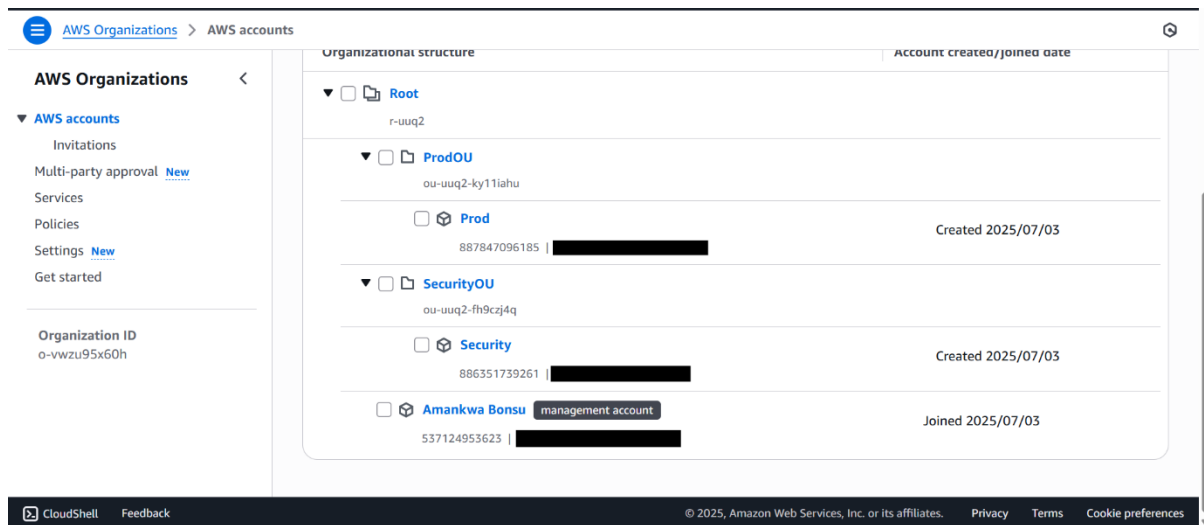
- AWS Cost Explorer for organization-wide view.
- AWS Budgets or cost anomaly detection for account-level alerts.
- Cost allocation tags for resource tracking.
- Plan OU structure for future growth.
- Consider nested OUs for complex organizations.
- Enable AWS Config for compliance monitoring.

6. Security Best Practices

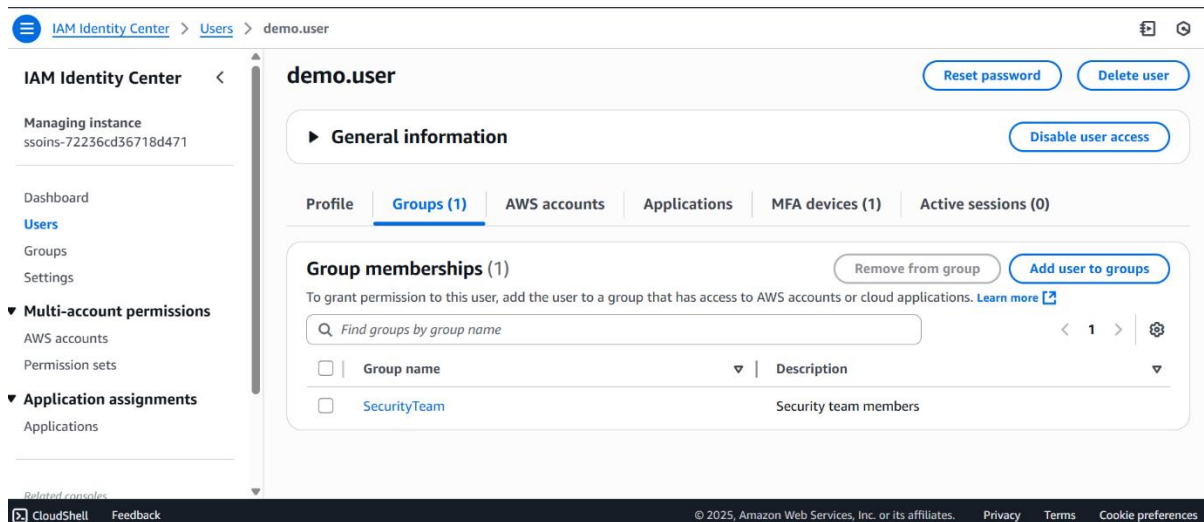
- Use **SCPs** to restrict access to sensitive or expensive services.
- Centralize access with **SSO** and **Permission Sets**.
- Assign users to groups, and use those groups for account-level access assignment.
- Use AWS Security Hub for organization-wide view.
- Maintain SSO permission sets as roles change.
- Tag resources with environments and ownership.

7. Proof Of Concept

After terraform apply the Organization is provisioned with its hierarchichal view below.



As you can see below user is provisioned and its in the securityteam group with the permission sets displayed.



IAM Identity Center <

Managing instance
ssoins-72236cd36718d471

Dashboard
Users
Groups
Settings

Multi-account permissions
AWS accounts
Permission sets

Application assignments
Applications

Related resources

demo.user [Reset password](#) [Delete user](#)

General information [Disable user access](#)

Profile Groups (1) **AWS accounts** Applications MFA devices (1) Active sessions (0)

AWS account access (2) [Assign accounts](#)

Search by account name, ID or email

AWS accounts (1/2)

- ☒ **Prod**
887847096185
- ☐ **Security**
886351739261

Prod
ID: 887847096185

Applied permission sets (1)

- ☐ **ReadOnlyAccess**
ps-eb7f7265ae3c3795

Before the user can login to the portal an email should be sent to the user's email for the user to verify the email. After verification the user can access the dashboard through a portal link and set a new password. MFA can also be enforced upon sign in for another layer of security.

IAM Identity Center > **Users** > demo.user

IAM Identity Center <

Managing instance
ssoins-72236cd36718d471

Dashboard
Users
Groups
Settings

▼ **Multi-account permissions**
AWS accounts
Permission sets

▼ **Application assignments**
Applications

Related resources

Primary information

Attribute key	Value
Username	demo.user
Email	[redacted] Verified
First name	Demo
Last name	User
Display name	Demo User

Contact methods

Attribute key	Value
Phone number	-

Job-related information

Attribute key	Value
---------------	-------

[CloudShell](#) [Feedback](#) © 2025, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

IAM Identity Center > **Settings**

IAM Identity Center <

Managing instance
ssoins-72236cd36718d471

Dashboard
Users
Groups
Settings

▼ **Multi-account permissions**
AWS accounts
Permission sets

▼ **Application assignments**
Applications

Related resources

Identity source [Actions](#)

Choose the directory where you want to manage your users and groups. [Learn More](#)

Identity source
Identity Center directory

Authentication method
Password

Provisioning method
Direct

AWS access portal URL
<https://d-90663b67a2.awsapps.com/start>

Identity Store ID
[d-90663b67a2](#)

Issuer URL
<https://identitycenter.amazonaws.com/ssoins-72236cd36718d471>

When the user logs in, the user can see the accounts that can be accessed in the portal.

The screenshot shows the AWS Access Portal interface. At the top, there's a header with the AWS logo, 'access portal', and a user profile 'Demo'. Below the header, there's a navigation bar with 'Accounts' and 'Applications' tabs. The 'Accounts' tab is active, showing a list of AWS accounts. The list includes two accounts: 'Prod' (ID: 887847096185) and 'Security' (ID: 886351739261). Each account has links for 'ReadOnlyAccess' and 'Access keys'. A 'Create shortcut' button is visible in the top right of the accounts list. The footer contains a 'Feedback' link, copyright information, and links for 'Privacy', 'Terms', and 'Cookie Preferences'.

AdminAccess for demo.user

The screenshot shows the AWS AdminAccess console for 'demo.user'. The interface has a top navigation bar with the AWS logo, a search bar, and a user profile 'AdminAccess/demo.user'. Below the navigation bar, there's a 'Console Home' section with a 'Reset to default layout' button and an 'Add widgets' button. The main content area is divided into two panels. The left panel, titled 'Recently visited', shows 'No recently visited services' and suggests exploring common AWS services like EC2, S3, Aurora and RDS, and Lambda. The right panel, titled 'Applications (0)', shows 'No applications' and suggests creating an application. The 'Applications' panel includes a 'Select Region' dropdown set to 'us-east-2 (Current Region)' and a 'Find applications' search bar. The footer contains a 'CloudShell' link, 'Feedback' link, copyright information, and links for 'Privacy', 'Terms', and 'Cookie preferences'.

ReadOnlyAccess for demo.user

The screenshot shows the AWS ReadOnlyAccess console for 'demo.user'. The interface is identical to the AdminAccess console, with a top navigation bar, a 'Console Home' section, and two main panels: 'Recently visited' and 'Applications (0)'. The 'Applications' panel shows 'No applications' and a 'Create application' button. The footer contains a 'CloudShell' link, 'Feedback' link, copyright information, and links for 'Privacy', 'Terms', and 'Cookie preferences'.

Because of SCP denying Sagemaker when user tries to access sagemaker user is denied access even with AdminAccess permission set applied to user.

The screenshot shows the Amazon SageMaker console interface. At the top, a red error banner displays the message: "AccessDeniedException: User: arn:aws:sts:886351739261:assumed-role/AWSReservedSSO_AdminAccess_0b9abe2bf21f0360/demo.user is not authorized to perform: datazone:ListDomains on resource: arn:aws:datazone:us-east-2:886351739261:domain/*". A "Diagnose with Amazon Q" button is visible next to the error message. Below the banner, the main content area features the "Amazon SageMaker" logo with a "NextGen" tagline, followed by the text "The center for data, analytics, and AI". A paragraph describes the integrated experience of SageMaker. To the right, a white box titled "Get started with Amazon SageMaker Unified Studio" contains a "Create a Unified Studio domain" button. Below this, a section titled "Continue with Amazon DataZone" provides instructions on how to proceed. The footer includes links for "CloudShell", "Feedback", and copyright information for Amazon Web Services, Inc. or its affiliates, along with "Privacy", "Terms", and "Cookie preferences" links.

Analytics

Amazon SageMaker NextGen

The center for data, analytics, and AI

Amazon SageMaker brings together comprehensive AI and analytics services into an integrated experience, to enable data processing, SQL analytics, model development and training, and generative AI. From a unified studio, access and act on all your data using the best tool for the job, assisted by Amazon Q at every step. Get unified access to your data whether it's stored in data lake, data warehouse, or federated data sources, with governance built-in to meet your enterprise security needs.

Amazon DataZone is now part of Amazon SageMaker.

Get started with Amazon SageMaker Unified Studio

Create an Amazon SageMaker Unified Studio domain in this AWS account.

[Create a Unified Studio domain](#)

Continue with Amazon DataZone

Continue using Amazon DataZone without any additional Amazon SageMaker capabilities.

[CloudShell](#) [Feedback](#) © 2025, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)