Terraform IaC - Entra ID SAML SSO and Identity Automation

Field	Details
Document Type	Automation
Applies To	Entra ID & Terraform
Audience	Entra ID engineer, Devops admin
Author	AK. Udofeh
Last Updated	March 2026

Overview

This documentation describes a reusable Terraform InfrastructureasCode (IaC) setup for automating Microsoft Entra ID (Azure AD) identity resources.

The project focuses on:

· SAML SSO application provisioning (App Registration + Service Principal + signing certificate)

· Conditional Access policies

· Groups, roles, and app role assignments

· Token protection and advanced identity features (optional modules)

· Environmentspecific configurations for dev and prod

Terraform uses the AzureAD provider to manage Entra ID objects such as applications, service principals, groups, and Conditional Access policies. AzAPI and Microsoft Graph can be layered in for advanced scenarios (claims mapping, Graphonly features).

A public GitHub repository with the full Terraform code is available here:

https://github.com/ak-wizzy/Terraform-SAML-SSO

Repository Structure

terraform-SAML-SSO/
│
├── environments/

```
│   ├── dev/
│   │   ├── backend.tf
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   ├── providers.tf
│   │   ├── terraform.tfvars
│   │   └── variables.tf
│   │
│   └── prod/
│       ├── backend.tf
│       ├── main.tf
│       ├── outputs.tf
│       ├── providers.tf
│       ├── terraform.tfvars
│       └── variables.tf
│
├── modules/
│   ├── saml_sso/
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   ├── variables.tf
│   │   └── README.md
├── .gitignore
└── Readme.md
```

The project is organized to separate environments from modules, and to keep Terraform state isolated per environment.

Key Design Points

- Environments (environments/dev, environments/prod):
- Contain backend.tf, providers.tf, variables.tf, and an environmentspecific main.tf.
- Each environment has its own state (local or remote backend).
- Values like URLs, tenant IDs, and object IDs live in terraform.tfvars.
- Modules (modules/*):
- Implement reusable logic for SAML SSO apps, OIDC apps, Conditional Access, groups, service principals, and token protection.
- Modules are parameterdriven and environmentagnostic.
- No providers or backends are declared inside modules; those are defined per environment.

Components

1. SAML SSO Module (modules/saml_sso)

This module provisions everything needed for an Entra ID–backed SAML service provider:

- App Registration for the SAML application (identifier URI / Entity ID)
- Service Principal for the application
- Optional selfsigned SAML signing certificate:
- Generated via the TLS provider
- Uploaded to the Service Principal as a certificate credential
- App role assignments for users or groups (basic access control)

· Outputs for:

- Application (client) ID
- Service Principal ID
- Certificate details (thumbprint, expiry, etc.)
- Values you need to configure your external SAMLaware application

The SAML SSO *protocol configuration* (reply URLs, Entity ID, and claims) is surfaced through:

- The App Registration (identifier URIs)
- The Enterprise Application created for the service principal (SAML SSO blade in the portal)

> Note: Microsoft does not currently offer a firstclass Terraform resource for full SAML Enterprise Application configuration; you can extend the module with AzAPI/Microsoft Graph calls if you need to automate claims mapping or the full SAML blade setup.

2. Conditional Access Module (modules/conditional_access)

This module encapsulates Conditional Access policies such as:

- Requiring MFA or phishingresistant authentication for admins
- Blocking legacy authentication
- Enforcing location, device, or signin risk–based controls

It typically wraps resources like azuread_conditional_access_policy.

Policies can be toggled per environment via variables, making it easy to:

- Enable strict policies in prod
- Use relaxed or test policies in dev

3. Groups and Roles (modules/groups)

Manages:

- Security groups and Microsoft 365 groups
- Roleassignment helper logic (e.g., mapping groups to application roles or directory roles)

This keeps identity governance consistent across environments, and allows app access to be managed entirely through group membership rather than static lists of users.

4. Service Principals (modules/service_principals)

Reusable module for:

- Creating service principals for automation
- Assigning directory roles or app roles
- Supplying credentials (client secrets or certificates) where needed

This is useful when Terraform itself runs under a dedicated Entra ID service principal and you want to manage its lifecycle as code.

5. Token Protection / Advanced Modules (modules/token_protection)

Placeholder / optional module for more advanced identity features, such as:

- Token protection policies
- Enhanced session controls
- Future Entra ID features not covered by the core providers

These can be added without breaking the core SAML SSO or Conditional Access modules.

Environment Layout and Workflow

Each environment (dev, prod) wires the modules together and supplies environmentspecific values.

Typical environments/dev responsibilities:

- Configure the backend (backend.tf):
- Local state for lab/testing
- Or remote backend (Terraform Cloud / HCP) for shared state and collaboration
- Configure providers (providers.tf):
- AzureAD provider (identity and directory objects)
- AzAPI provider (optional)
- TLS provider (certificate generation)
- Declare variables (variables.tf) and defaults suitable for dev
- Compose modules in main.tf:
- Instantiate saml_sso for one or more apps
- Instantiate conditional_access, groups, etc.

State and Backends

Backends are typically configured per environment:

- environments/dev/backend.tf might point to a dev workspace in Terraform Cloud.
- environments/prod/backend.tf points to a separate prod workspace.

- This separation avoids any crossenvironment state contamination and allows different approval flows for prod vs. dev runs.

Using the Project

All commands are run from the *environment* directory you want to manage (e.g., environments/dev).

1. Prerequisites

- Terraform CLI installed
- Azure CLI or service principal credentials available
- Appropriate Microsoft Entra ID permissions to:
- Read and Write Directory data
- Create app registrations and service principals
- Manage certificates for service principals
- Manage Conditional Access policies (if using that module)

2. Authenticate

For interactive use (Azure CLI):

```
az login
az account set --subscription "<subscription-id>"
```

For service principal use, run these in your terminal before your plan

```
$env:ARM_CLIENT_ID = "YOUR_APP_ID"
$env:ARM_CLIENT_SECRET = "YOUR_CLIENT_SECRET"
$env:ARM_TENANT_ID = "YOUR_TENANT_ID"
$env:ARM_SUBSCRIPTION_ID = "YOUR_SUBSCRIPTION_ID"
```

Why is this better? It’s faster, more stable for automation, and avoids the "exit status 1" errors you're seeing from the CLI wrapper.

3. Configure Environment Variables

Copy the sample tfvars and adjust for your environment:

```
cd environments/dev
cp terraform-sample.tfvars terraform.tfvars
```

Edit terraform.tfvars with values such as:

- app_display_name

- identifier_uris (SAML Entity ID)
- reply_urls (SAML ACS URLs)
- sign_on_url
- logout_url
- User/group object IDs to assign to the app
- Flags for selfsigned cert generation vs. bringing your own certificate

4. Initialize Terraform

```
terraform init
```

Initializes providers and backends for this environment.

5. Validate and Plan

```
terraform validate
terraform plan -out=tfplan
```

Review the generated plan to see which Entra ID resources will be created or updated.

6. Apply

```
terraform apply tfplan
```

This will:

- Create the App Registrations and Service Principals
- Generate and upload a selfsigned SP signing certificate (if enabled)
- Assign users or groups
- Create Conditional Access policies and other identity artifacts configured in the environment

SAML SSO Flow (What You Get After Apply)

Once the SAML SSO module has been applied:

- App Registration is created in Entra ID with the configured display name and identifiers.
- Service Principal (Enterprise Application) is created and visible under Enterprise applications in the portal.
- Signing Certificate is generated and attached to the Service Principal (if selfsigned mode is enabled).
- Users/Groups are assigned to the Enterprise Application based on the IDs you supplied.
- SAML metadata (certificate + identifiers) can be downloaded from the Enterprise Application’s SAML SSO configuration blade and used to configure the external service provider.

If needed, you can extend the module with AzAPI calls to fully configure SAML attributes & claims, relay state, and other advanced settings through Microsoft Graph.

Extending and Contributing

The GitHub repository

https://github.com/ak-wizzy/terraform-entra-id

contains:

- Full Terraform code for all modules and environments
- Modulelevel READMEs with variable and output descriptions

· Examples for:

- Creating additional SAML SSO apps
- Adding more Conditional Access policies
- Integrating with CI/CD workflows for automated deployments

Contributions are welcome via pull requests or issues:

- New modules (e.g., SCIM provisioning, advanced claims, token protection)
- Bug fixes or enhancements to the SAML SSO and Conditional Access logic
- Documentation improvements and realworld usage examples

Revision #4
Created 2026-03-25 11:47:27 UTC by AK. Udofeh
Updated 2026-04-17 16:25:40 UTC by AK. Udofeh