Managing Terraform State in Azure

Christos Galanopoulos

DevOps Engineer | Microsoft MVP

- → **Website:** christosgalano.github.io
- → **LinkedIn:** in/christos-galanopoulos
- → **GitHub:** @christosgalano
- → **Medium:** @christosgalanop
- → **Sessionize:** christos-galanopoulos

Note: Everything discussed about Terraform also applies to OpenTofu.

Terraform: What & Why?

→ Infrastructure as Code (IaC)
→ Declarative resource provisioning
→ Manages infrastructure lifecycle efficiently
→ Automates deployments with reproducible configurations

Why Terraform Needs State

→ Maps real-world resources to configurations for accurate tracking

→ Ensures changes follow the correct order of operations

→ Reduces API calls, improving performance at scale

→ Detects drift and maintains a history of changes

Before any operation, Terraform *refreshes* the state to align with real infrastructure.

Where Should You Store State?

- → Local storage (not recommended)
  - → Risk of accidental deletion
  - → Difficult for team collaboration
- → Remote storage
  - → Enables team access and versioning
  - → Prevents data loss with better security

Storing State in Azure

Azure Blob Storage is a reliable backend for Terraform state, providing:

→ High availability and scalability
→ Built-in locking to prevent conflicts
→ Secure access control with RBAC and encryption
→ Private endpoints for network isolation
→ Geo-redundant storage for resilience

Project-Based State Management

Approach 1: Storage Account per Environment

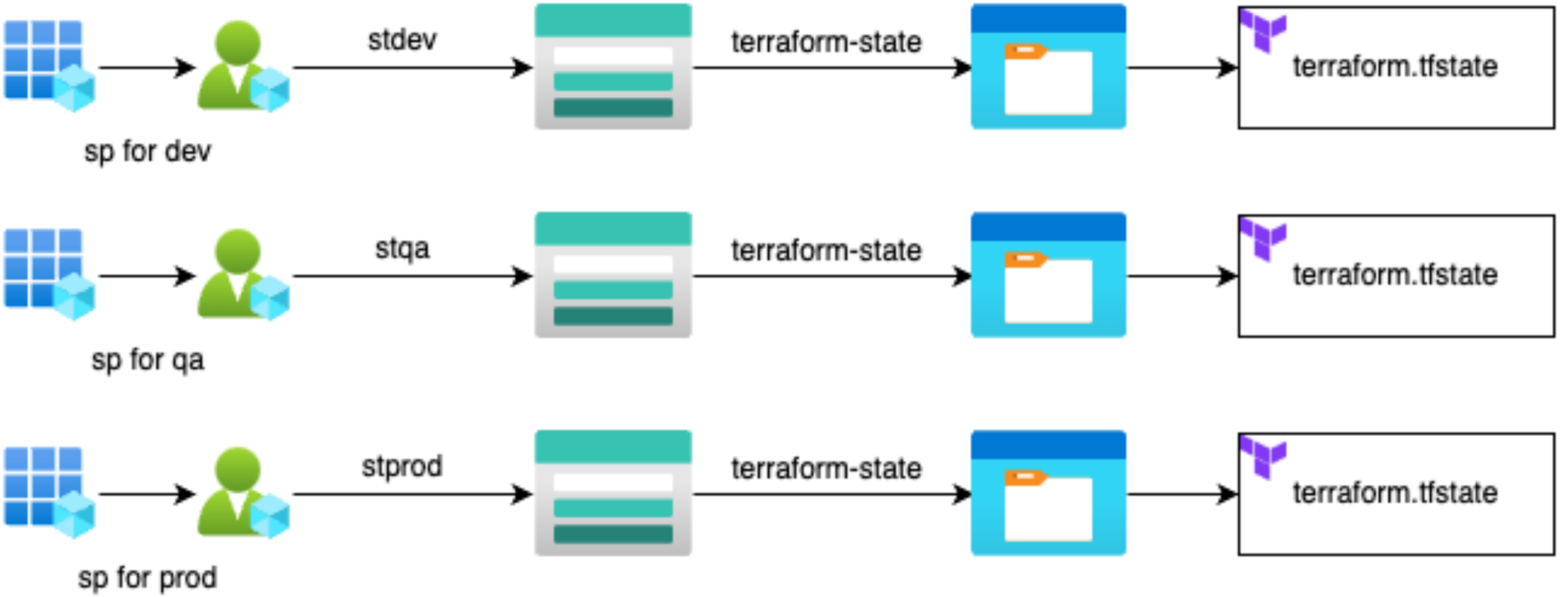
Each environment has its own dedicated storage account.

→ Provides full isolation

→ Allows separate access controls per environment

→ Increases management overhead

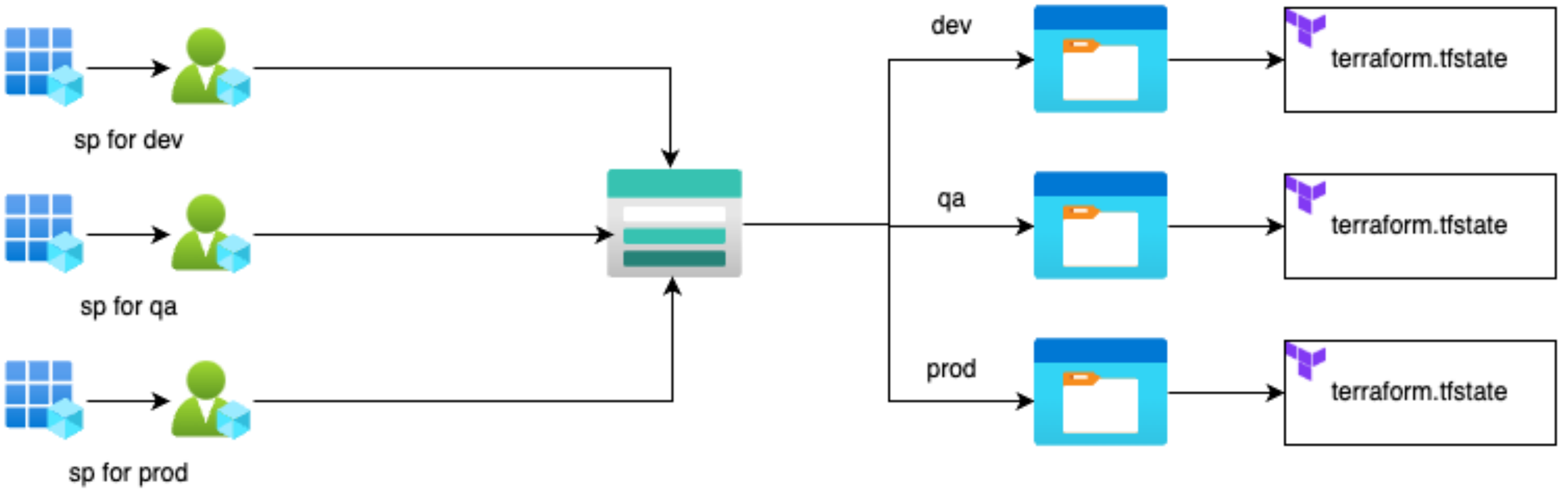
→ Higher costs due to multiple accounts


stdev
terraform-state
terraform.tfstate
sp for dev
stqa
terraform-state
terraform.tfstate
sp for qa
stprod
terraform-state
terraform.tfstate
sp for prod

Approach 2: Storage Account per Project

A single storage account for the project, with separate containers for each environment.

- → Easier to manage with fewer accounts
- → More cost-efficient
- → Reduced isolation compared to per-environment storage
- → Requires RBAC at the container level for access control


sp for dev
sp for qa
sp for prod
dev
qa
prod
terraform.tfstate
terraform.tfstate
terraform.tfstate

Centralized State Management

Core Principles

→ Single storage account for all projects
→ Dedicated container per project
→ Custom role-based access with conditions
→ Each state file linked to a dedicated service principal

Custom Role: Terraform State Contributor

```
{
  "properties": {
    "roleName": "Terraform State Contributor",
    "description": "Allows a service principal to manage Terraform state files by granting read, write, and add access to Azure Storage blob containers.",
    "assignableScopes": [
      "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    ],
    "permissions": [
      {
        "actions": [
          "Microsoft.Storage/storageAccounts/blobServices/containers/read",
          "Microsoft.Storage/storageAccounts/blobServices/containers/write",
          "Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey/action"
        ],
        "notActions": [],
        "dataActions": [
          "Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read",
          "Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write",
          "Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action"
        ],
        "notDataActions": []
      }
    ]
  }
}

```

Conditional Role Assignment

A condition is an additional check that refines role assignments for more granular access control.

It ensures an action is allowed only if the specified conditions evaluate to true.

Idea: Use conditions to narrow access from the *container level* down to *specific blob paths* for even finer control.

```hcl
data "azurerm_storage_account" "state" {
  name                = var.storage_info.storage_account_name
  resource_group_name = var.storage_info.resource_group_name
}

data "azurerm_storage_container" "state" {
  name                 = var.storage_info.container_name
  storage_account_name = data.azurerm_storage_account.state.name
}

resource "azurerm_role_assignment" "blob_contributor" {
  role_definition_id = var.sp_access_config.role_definition_id
  scope              = "${data.azurerm_storage_account.state.id}/blobServices/default/containers/${data.azurerm_storage_container.state.name}"
  principal_id       = var.sp_access_config.object_id
  principal_type     = "ServicePrincipal"
  description        = "Allow the service principal to read from and write to specific paths within the blob container."
  condition_version  = "2.0"
  condition          = <<-EOT
(
 (
  !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'})
  AND
  !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'})
  AND
  !(
    ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}
    AND
    NOT SubOperationMatches{'Blob.List'}
  )
 )
 OR
 (
  @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path]
  StringLike
  '${var.sp_access_config.blob_path}'
 )
)
EOT
}

```

```
variable "sp_access_config" {
  type = object({
    object_id          = string
    blob_path          = string
    role_definition_id = string
  })
  description = <<DESCRIPTION
The access configuration for the service principal. Supported fields are:

- `object_id`: The object id of the project's Terraform service principal.
- `blob_path`: The path under the blob container where the service principal should have read/write access.
- `role_definition_id`: The id of the role definition to assign to the service principal.

DESCRIPTION
}

variable "storage_info" {
  type = object({
    resource_group_name  = string
    storage_account_name = string
    container_name       = string
  })
  description = <<DESCRIPTION

The storage account information for the Terraform state. Supported fields are:

- `resource_group_name`: The name of the resource group where the storage account is located.
- `storage_account_name`: The name of the storage account.
- `container_name`: The name of the blob container.

DESCRIPTION
}

```

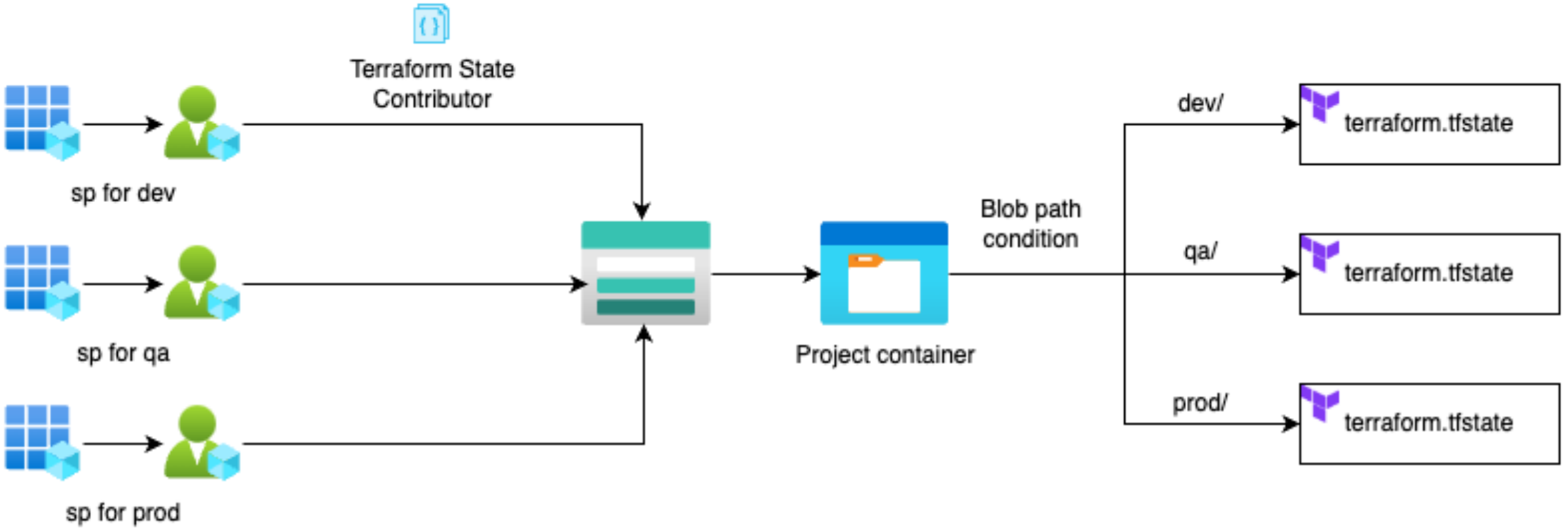
Scenario: Single Module

Each environment has a dedicated service principal (N service principals for N environments).

→ Ensures strong isolation

→ Maintains a clean and scalable architecture

→ Simple to manage with least privilege enforcement


Terraform State
Contributor
sp for dev
sp for qa
sp for prod
Project container
Blob path
condition
dev/
qa/
prod/
terraform.tfstate
terraform.tfstate
terraform.tfstate

Scenario	Allowed?	Reason
Read any Terraform state (`terraform.tfstate`)	No	Read access is blocked unless path matches `sp_access_config.blob_path`
List all blobs (`terraform state pull`)	Yes	Only listing (`Blob.List`) is allowed
Modify (write) state in allowed path	Yes	Within `sp_access_config.blob_path`
Modify (write) state outside allowed path	No	Write blocked unless within `sp_access_config.blob_path`
Delete any Terraform state file	No	No explicit delete permissions granted

Scenario: Multiple Modules

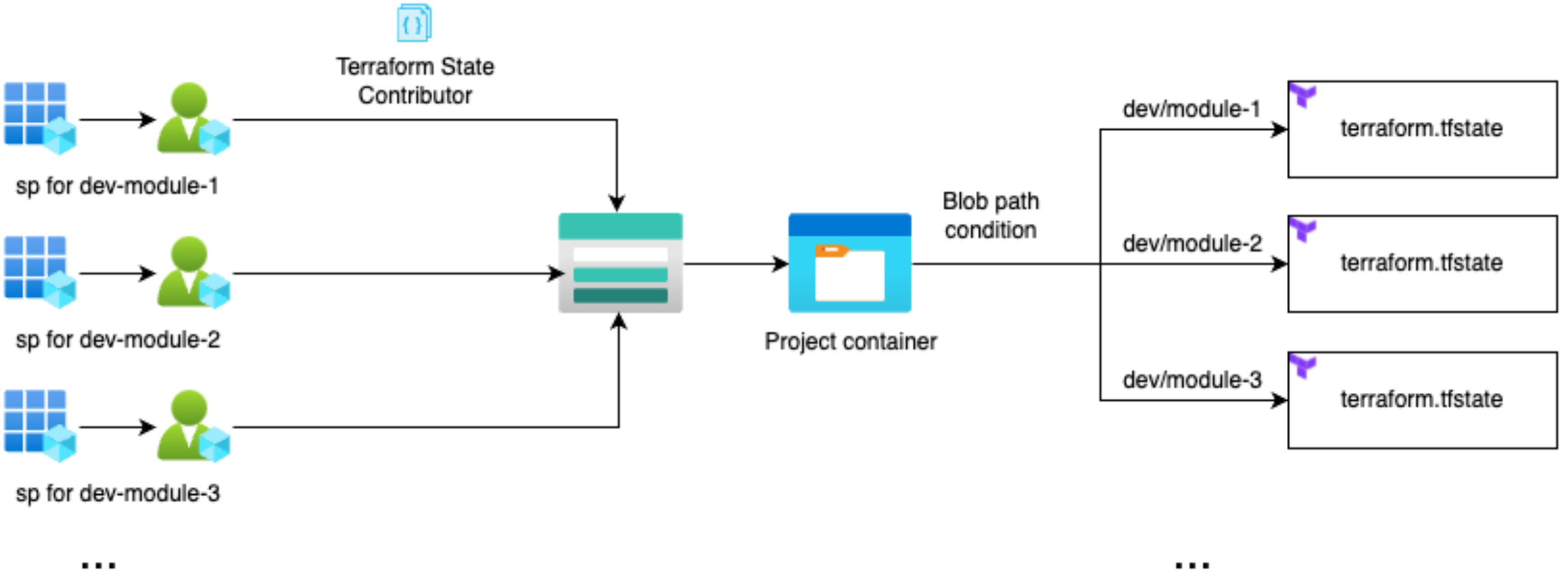
Each module has its own state file, requiring separate service principals for each environment (NxM service principals, where N is environments and M is modules).

→ More complex setup

→ Supports large teams and modular infrastructure

→ Isolates state per module and environment for better security

→ Requires more granular access control


Terraform State
Contributor
sp for dev-module-1
sp for dev-module-2
sp for dev-module-3
...
Blob path
condition
Project container
dev/module-1
dev/module-2
dev/module-3
terraform.tfstate
terraform.tfstate
terraform.tfstate
...

```hcl
data "azurerm_storage_account" "state" {
  name                = var.storage_info.storage_account_name
  resource_group_name = var.storage_info.resource_group_name
}

data "azurerm_storage_container" "state" {
  name                 = var.storage_info.container_name
  storage_account_name = data.azurerm_storage_account.state.name
}

resource "azurerm_role_assignment" "blob_contributor" {
  role_definition_id = var.sp_access_config.role_definition_id
  scope              = "${data.azurerm_storage_account.state.id}/blobServices/default/containers/${data.azurerm_storage_container.state.name}"
  principal_id       = var.sp_access_config.object_id
  principal_type     = "ServicePrincipal"
  description        = "Allow the service principal to read from and write to specific paths within the blob container."
  condition_version  = "2.0"
  condition          = <<-EOT
(
 (
  !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'})
  AND
  !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'})
 )
 OR
 (
  @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path]
  StringLike
  '${var.sp_access_config.blob_path}'
 )
)
EOT
}

```

Scenario	Allowed?	Reason
Read any Terraform state (`terraform.tfstate`)	Yes	Read access is not restricted at the container level
List all blobs (`terraform state pull`)	Yes	Only listing (`Blob.List`) is allowed
Modify (write) state in allowed path	Yes	Within `sp_access_config.blob_path`
Modify (write) state outside allowed path	No	Write blocked unless within `sp_access_config.blob_path`
Delete any Terraform state file	No	No explicit delete permissions granted

RBAC and ABAC in Terraform State Management

Role-Based Access Control (RBAC)

- → Assigns permissions based on predefined roles
- → Custom *Terraform State Contributor* role for state management
- → Enables principle of least privilege
- → Simplifies access management at scale

Attribute-Based Access Control (ABAC)

- → Extends RBAC with dynamic, context-aware access control
- → Implemented through conditional role assignments
- → Enhances security for Terraform state management:
  - → Restrict access by blob path within containers
  - → Limit access based on attributes such as resource tags or user properties

Combining RBAC and ABAC provides robust, scalable access control for Terraform state management in Azure.

Comparing Project-Based vs Centralized State Management

Feature	Project-Based	Centralized
Storage Structure	Multiple storage accounts	Single storage account with containers
Isolation	Strong (per environment)	Logical separation via containers & RBAC
Management Effort	High (more accounts to maintain)	Lower (centralized with structured access)
Access Control	Simpler (per account RBAC)	More complex (RBAC + conditional role assignments)
Scalability	Limited (harder to scale)	High (supports large environments & teams)

Rule of Thumb

Choose Centralized If	Choose Project-Based If
Managing multiple projects and teams	Working with a small, stable number of projects
Planning for growth and scalability	Preferring isolated environments with minimal overhead

Switching from Project-Based to Centralized later requires migrating state files, adding complexity. Plan ahead.

Summary

Project-Based Approach

→ Two strategies: Storage Account per Environment or per Project

→ Strong isolation between environments

→ Simpler access control with per-account RBAC

→ Higher management overhead and costs

→ Suitable for small, stable environments

Centralized Approach

- → Single storage account with dedicated containers per project
- → Combines RBAC and ABAC for scalable access control
- → Supports single and multiple module scenarios
- → Scalable and efficient for managing multiple teams
- → Ideal for managing multiple projects and future growth

Thank You