Infisical access control

let’s go step by step through **Infisical access control** — how it works, how you can use it to protect secrets per environment, team, and microservice.

? 1. Access Control Model Overview

Infisical has a **role-based access control (RBAC)** system that operates at **two levels**:

Level	Scope	Example
Workspace Level	Who can access the entire workspace (e.g., “Beza”)	Admins, Developers, Viewers
Project Level	Who can access a specific project (e.g., “Beza Project”)	Microservice owners, DevOps
Environment Level	Which environment’s secrets can they see/edit (e.g., dev/staging/prod)	“Auth Dev Team” can see `dev`, not `prod`

Each member’s access can be **finely tuned per project and per environment**.

? 2. Core Roles and Permissions

Role	Description	Access Capabilities
□□ **Admin**	Full control	Manage users, roles, secrets, integrations, service tokens
□□□ **Developer**	Typical engineer	Read & write secrets, but cannot manage workspace-level settings
□□ **Viewer**	Read-only	Can view secrets, but not edit or rotate
⚙ **Custom Role**	Your own rule set	Fine-tuned access, e.g., “Can view dev, edit staging only”

> □□ Pro tip: Always create **custom roles** for production environments to enforce least privilege.

? 3. Environment-Based Access

Each **Infisical Project** (e.g., `Beza`) has multiple **environments** (`dev`, `staging`, `prod`, etc.).
You can control who can access which environment.

For example:

Member	Environment	Access
`gkibria@beza.com`	dev	Read & Write
`chris@beza.com`	prod	Read only
`automation@beza.com` (service token)	prod	Read only (used by CI/CD)

 Use case: Developers can freely modify dev secrets, but prod secrets are locked down to only admins and CI/CD pipelines.

? 4. Service Tokens (For CI/CD & Kubernetes)

Instead of giving your GitHub Actions or Kubernetes Operator your personal login, Infisical uses **Service Tokens**.

They can be **scoped** by:

- **Project** → e.g., only for “Beza”
- **Environment** → e.g., only `staging`
- **Secret path** → e.g., only `/auth` or `/order`

Example (in Dashboard → Access Control → Service Tokens):

Token Name	Scope	Expiry	Permissions
`beza-devops-prod`	Project: Beza → Env: prod → Path: /	90 days	Read
`beza-dev-ci`	Project: Beza → Env: dev → Path: /auth		

? 5. Group & Folder-Based Restrictions

If you use **folders** or **nested paths** (like `/auth`, `/order`, `/sell`):

- You can assign specific permissions per **path**.
- e.g.,
  - `/auth` → Auth Team
  - `/order` → Order Team
  - `/shared` → Global DevOps Team

This lets each team only see secrets relevant to their microservice.

Here’s how I’d structure **access for your My microservices** (Beza) (`auth`, `order`, `sell`):

Project	Environment	Role	Member / Token	Access
Beza	dev	Developer	Your dev team	Read/Write
Beza	staging	Developer	QA team	Read only
Beza	prod	Admin	You (GK)	Full control
Beza	prod	Service Token (ArgoCD / GitHub Actions)	CI/CD	Read only
Beza	all	Viewer	Client (optional)	Read-only view

Revision #1
Created 3 November 2025 14:49:52 by Admin
Updated 3 November 2025 14:52:13 by Admin