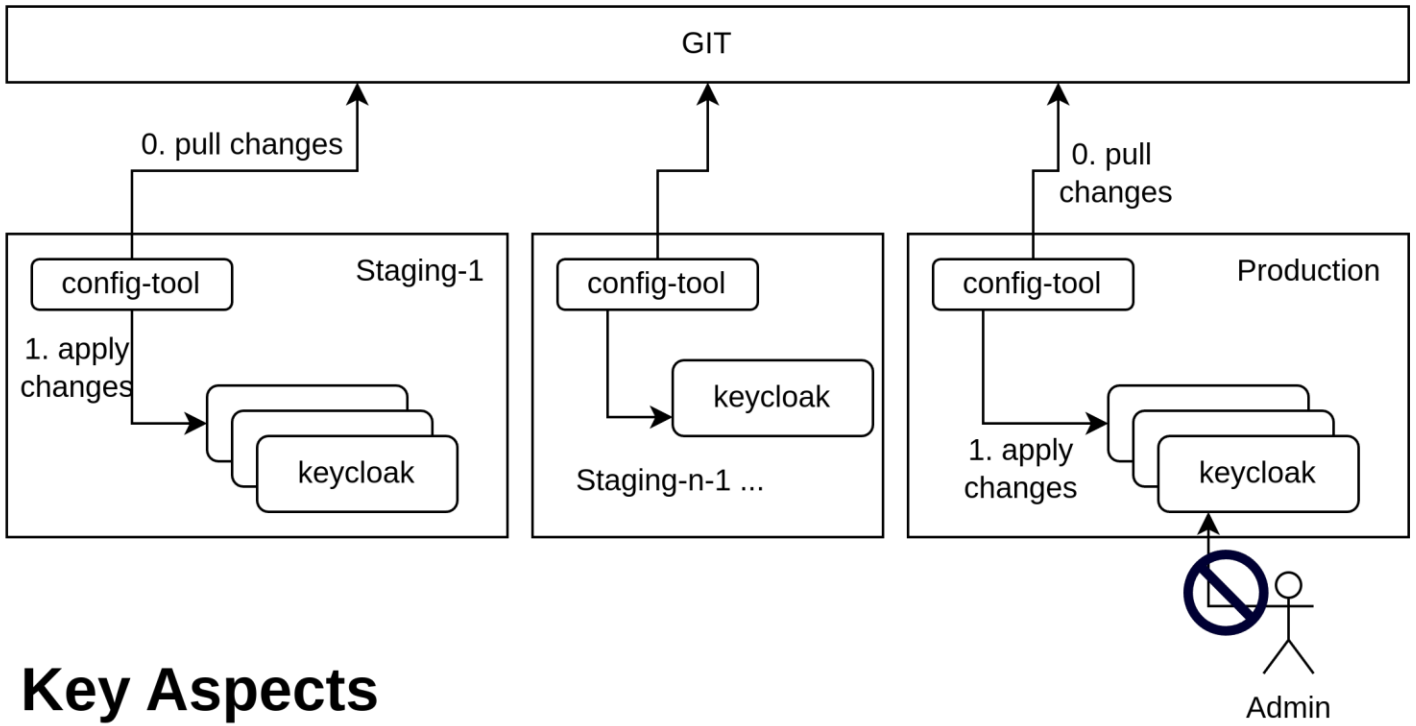
Streamlining Keycloak Configuration Management

Exploring keycloak-config-cli

Configuration Management Tools



Key Aspects

- Version Control
- Rollback/Updates
- Secrets Management
- Key Generation
- Keystore Management
- Key Rotation
- Auditability
- Segregation of Realms/Clients
- Maintenance
- Platform Support

Keycloak Admin API

… RESTfull endpoint for programmatic access to manage and configure Keycloak realms, clients, users, roles, and more ...

Key Features:

- **Full Realm Management:**
  - Create, update, and delete realms.
  - Configure authentication flows, policies, and settings.
- **User Management:**
  - Create, update, delete, and search for users.
  - Manage credentials (passwords, OTP), roles, and group assignments.
- **Client Configuration:**
  - Manage clients (applications), client secrets, and access token settings.
  - Configure roles, permissions, and protocol mappers.
- **Role and Group Management:**
  - Create, update, and manage roles (both realm and client-level).
  - Organize users into groups with group-specific roles.

Tools Comparison (1)

Aspect easy ✅ fair 😞 tough ❌	Keycloak Admin CLI	keycloak-config-cli	Terraform Provider for Keycloak	Keycloak Ansible Collection	Keycloak JSON Import/Export	Keycloak Operator (Kubernetes)
Version Control, **Consistency Across Env (dev, staging, prod)**	1. Shell scripts stored in Git. ✅ 2. Script execution specific to runtime. ❌ 3. Scripts must be manually maintained and synchronized across environments. ❌	1. JSON/YAML config in Git. ✅ 2. Single source of truth. ✅	1. Terraform HCL in Git. ✅ 2. Script execution specific to runtime. ❌ 3. Environments use workspaces or env-specific files. 😞	1. Playbooks stored in Git. ✅ 2. Script execution specific to runtime. ❌ 3. Ansible inventories manage configurations per env. 😞	1. Entire JSON realm exports can be stored in Git. ❌ 2. Single source of truth. ✅ 3. Applies full realm dumps; less flexible for partial changes. ❌	1. YAML manifests/Helm charts in Git. ✅ 2. Script execution specific to runtime. ❌ 3. Managed through Kubernetes namespaces or clusters. 😞
Rollback / Updates, **Complex Configurations**	1. Track via script versions, no native state management. ❌ 2. Idempotency . ❌	1. Revert via Git and reapply. ✅ 2. Automatic state management. ✅ 3. Idempotency . ✅	1. Previous config from git. ✅ 2. Explicit state management through state files. ❌ 3. Idempotency . ❌	1. Reapply older playbooks. ✅ 2. Explicit state management. ❌ 3. Idempotency . ❌	Restore previous realm versions by importing older JSON. ✅ 1. Idempotency . ✅	1. K8s handles rollback. ✅ 2. Explicit state management (manifests). ❌ 3. Idempotency . ❌
Secrets	External secret managers. 😞	Use environment variables or external managers. ✅	Inject secrets from external sources like Vault. ✅	Use Ansible Vault or external managers to handle secrets. ✅	Strip secrets from JSON files before committing to Git. ❌	Kubernetes Secrets and external managers (e.g., Vault). ✅

Tools Comparison (2)

Aspect easy ✅ fair 😞 tough ❌	Keycloak Admin CLI	keycloak-config-cli	Terraform Provider for Keycloak	Keycloak Ansible Collection	Keycloak JSON Import/Export	Keycloak Operator (Kubernetes)
Cryptographic Key Management	1. Key Generation: 😞 2. Key Store Management: 😞 3. Key Rotation: ❌	1. Key Generation: ✅ 2. Key Store Management: 😞 3. Key Rotation: ✅	1. Key Generation: 😞 2. Key Store Management: 😞 3. Key Rotation: ❌	1. Key Generation: ✅ 2. Key Store Management: 😞 3. Key Rotation: 😞	1. Key Generation: ✅ 2. Key Store Management: ❌ 3. Key Rotation: ❌	1. Key Generation: ✅ 2. Key Store Management: 😞 3. Key Rotation: 😞
Auditability	1. Git for traceability. ✅ 2. No logging API calls. 😞	1. Git for traceability. ✅ 2. Native logging API calls. ✅	1. Git for traceability. ✅ 2. No logging API calls. 😞 3. Terraform state. ✅	1. Git for playbook changes. ✅ 2. Ansible logs for execution tracking. ✅	1. Git tracks JSON changes. ✅ 2. Lacks granular control. ❌	1. Git for traceability. ✅ 2. Kubernetes logs. 😞 3. No logging API calls. 😞
Segregation: All / Realm / Client	Assign corresponding role to used admin credentials. ✅	Assign corresponding role to used admin credentials. Separate files per realm/client ✅	Assign corresponding role to used admin credentials. Modular config for realms/clients. ✅	Assign corresponding role to used admin credentials, Separate playbooks per realm/client. ✅	No built-in support for segregating realms/clients. ❌	Assign corresponding role to admin credentials. Segregate realms via Kubernetes CRDs. ✅
Maintenance	Evolves with Keycloak. API changes -> manual updates. ❌	Community maintained. Evolves with Keycloak. ✅	Community maintained. Lagging behind Keycloak releases. ❌	Community. Usually kept up to date with Keycloak but slow. ❌	Native to Keycloak, evolves directly with Keycloak. ✅	Community. evolves with Keycloak and Kubernetes. ✅
Supported Platforms	Bare metal, VMs, Docker, Kubernetes. ✅	Bare metal, VMs, Docker, Kubernetes. ✅	Bare metal, VMs, Docker, Kubernetes (with Terraform). 😞	Bare metal, VMs, Docker, Kubernetes (via Ansible). 😞	Bare metal, VMs, Docker. ❌	Kubernetes (cloud-native environments). 😞

Keycloak Config CLI

… The keycloak-config-cli tool has several unique advantages over other methods for managing Keycloak configurations ...

Key Advantages Summary:

- **Declarative Configuration:** Ensures Keycloak aligns with a well-defined desired state.
- **Version Control Friendly:** Easy integration with Git and traceable configuration changes (logging calls).
- **Consistency:** Ensures environments are consistently configured.
- **Idempotency:** Avoids duplication and ensures safe reapplication of configurations.
- **Simple Rollbacks:** Revert to previous configurations easily through version control and state management.
- **Modularity:** Separate configurations for realms and clients, easy to manage.
- **Stateless:** No need for state file management, unlike Terraform, Ansible.
- **Lightweight and fast:** Minimal overhead for use in CI/CD pipelines, unlike Terraform, Ansible.
- **Easier Learning Curve**: You write plain JSON or YAML, which is intuitive and easy to understand compared to Terraform and Ansible.
- **Customizable and Extensible**: Can be extended and scripted to fit more advanced use cases

Thank you!

www.adorsys.com | info@adorsys.com

adorsys