Snowflake + Terraform (TFC + GitLab) — Beginner-Friendly Presentation

> Audience: Snowflake platform/dev teams currently using **GitLab CI/CD** and **Flyway**, moving to **Terraform** with **Terraform Cloud Enterprise (TFC)**. Built for absolute beginners to Snowflake + Terraform.

---

1) Why Terraform for Snowflake?

Problem today (typical with Flyway-only): - Ad-hoc object creation (warehouses, roles, stages) via UI/SQL; hard to audit. - Drift between environments (dev/uat/prod) and between teams. - Manual grants and cost guardrails (resource monitors) are error-prone.

What Terraform adds: - **Infrastructure as Code (IaC)** for Snowflake account objects (warehouses, databases, schemas, roles, grants, resource monitors, stages, integrations, pipes, network policies, users, etc.). - **Version control** in GitLab, peer review via Merge Requests (MRs). - **Automated plans/applies** in Terraform Cloud Enterprise (TFC) with state locking & RBAC. - **Policy-as-code** (Sentinel) to enforce guardrails (e.g., auto_suspend must be $\leq$ 300 sec; `prevent_destroy` on prod databases). - **Separation of concerns**: Flyway continues to handle **application/schema migrations**; Terraform handles **platform & access**.

> Takeaway: Keep Flyway for table/view DDL that evolves with the app. Use Terraform to standardize the **Snowflake platform** around it.

---

2) Target Architecture (high-level)

- **GitLab** hosts the Terraform repo (and optionally application/Flyway repos).
- **Terraform Cloud Enterprise** integrates with GitLab for VCS-driven runs **or** API-driven runs from GitLab pipelines.
- **Snowflake SaaS** is provisioned via the **Snowflake Terraform provider** using **key-pair** or **OAuth** auth.
- Workspaces per environment (**dev / uat / prod**). Variables & policies isolated per workspace.
- Optional **Sentinel** policies to enforce standards and **Run Tasks** for security checks.

> See the **diagram** in chat and the Mermaid block below (can paste into draw.io via *Arrange → Insert → Advanced → Mermaid*).

```
flowchart LR
  subgraph GL[GitLab]
    R[Terraform repo]\nmodules / envs
    CI[GitLab CI/CD]\n(MR -> plan, main -> apply)
```

```
  end

  subgraph TFC[Terraform Cloud Enterprise]
    WS_DEV[Workspace: snowflake-dev]
    WS_UAT[Workspace: snowflake-uat]
    WS_PROD[Workspace: snowflake-prod]
    VARS[Var Sets (account, user, private key, role)]
    POL[Policy Sets (Sentinel)]
  end

  subgraph SF[Snowflake SaaS]
    ACC[Account]
    WH[Warehouses]
    DB[Databases & Schemas]
    RL[Roles & Grants]
    RM[Resource Monitors]
    STG[Stages & Integrations]
  end

  R -->|push/MR| CI
  CI -->|VCS or API| TFC
  TFC -->|plan/apply via provider| SF
  VARS -.-> WS_DEV
  VARS -.-> WS_UAT
  VARS -.-> WS_PROD
  POL -.-> WS_PROD
```

---

3) Repo & Module Layout (simple, scalable)

```
repo-root/
 ├─ modules/
 │   ├─ warehouse/
 │   ├─ database_schema/
 │   ├─ rbac/
 │   ├─ resource_monitor/
 │   └─ stage_integration/
 ├─ envs/
 │   ├─ dev/
 │   │   ├─ main.tf
 │   │   ├─ providers.tf
 │   │   └─ variables.tf
 │   ├─ uat/
 │   └─ prod/
```

```
├─ .gitlab-ci.yml  (if API-driven)
└─ README.md
```

- **modules/** encapsulate reusable patterns (e.g., a standard warehouse or RBAC set).
- **envs/** pin versions & inputs per environment. Each env folder maps to a **TFC workspace**.

---

4) Provider & Authentication (beginner-friendly)

Option A — Key pair (recommended for service principals):

```
# envs/dev/providers.tf
terraform {
  required_providers {
    snowflake = {
      source  = "Snowflake-Labs/snowflake"
      version = "~> 1.0" # pin a tested version
    }
  }
}

provider "snowflake" {
  account  = var.snowflake_account         # e.g., "xy12345.ap-south-1"
  user     = var.snowflake_user            # service user for IaC
  role     = var.snowflake_role            # e.g., SECURITYADMIN/ACCOUNTADMIN
(least privilege!)
  private_key_path         = var.private_key_path         # or private_key
  private_key_passphrase   = var.private_key_passphrase   # sensitive
}
```

Option B — OAuth (SSO/IdP flows): store refresh token/client creds in TFC workspace variables; configure provider accordingly. Key pair is usually simpler for CI.

TFC Variables (workspace or var set): - `SNOWFLAKE_ACCOUNT` (env var or tf var), `SNOWFLAKE_USER`, `SNOWFLAKE_ROLE`. - `SNOWFLAKE_PRIVATE_KEY` (sensitive), `SNOWFLAKE_PRIVATE_KEY_PASSPHRASE` (sensitive). - Mark **Sensitive** in TFC; restrict via workspace RBAC.

> Security tip: Create a dedicated **service user** with the minimal role needed (often `SECURITYADMIN` for role/grant mgmt and `SYSADMIN` for objects). Avoid `ACCOUNTADMIN` in prod.

---

5) Core Resources You'll Manage

- **Warehouses**: size, auto_suspend, auto_resume, scaling policy.
- **Databases & Schemas**: standard names, lifecycle, time travel.
- **RBAC**: roles, role hierarchies, **future grants**, user/role grants.
- **Stages & Integrations**: S3/Azure storage integrations, external/internal stages.
- **Resource Monitors**: credit limits & notifications for cost guardrails.
- **Network/Session Policies**, **Pipes/Tasks**, **External Functions** if applicable.

Example: Warehouse + DB + Schema + RBAC + Grants

```
module "wh_etl" {
  source          = "../../modules/warehouse"
  name            = "WH_ETL"
  size            = "XSMALL"
  auto_suspend    = 60
  auto_resume     = true
  scaling_policy  = "ECONOMY"
}

module "db_sales" {
  source = "../../modules/database_schema"
  db_name    = "SALES"
  schemas    = ["RAW", "CURATED", "APP"]
  time_travel_days = 1
}

module "rbac" {
  source = "../../modules/rbac"
  roles = {
    ROLE_SALES_READ   = { inherits = ["SYSADMIN"], comment = "Read-only for
SALES" }
    ROLE_SALES_WRITE  = { inherits = ["SYSADMIN"], comment = "Write for ETL" }
  }
  role_grants = [
    { role = "ROLE_SALES_READ",  to_role = "ANALYSTS" },
    { role = "ROLE_SALES_WRITE", to_role = "ETL" }
  ]
}

# Database/schema grants (incl. future grants)
resource "snowflake_database_grant" "sales_usage" {
  database_name = module.db_sales.db_name
  privilege     = "USAGE"
  roles         = ["ROLE_SALES_READ", "ROLE_SALES_WRITE"]
}
```

```
resource "snowflake_schema_grant" "sales_raw_select" {
  database_name = module.db_sales.db_name
  schema_name   = "RAW"
  privilege     = "SELECT"
  roles         = ["ROLE_SALES_READ"]
}

resource "snowflake_schema_grant" "sales_all_future" {
  database_name   = module.db_sales.db_name
  schema_name     = "APP"
  privilege       = "ALL PRIVILEGES"
  roles           = ["ROLE_SALES_WRITE"]
  on_future       = true
}
```

---

6) Environments & Workspaces (dev/uat/prod)

Pattern: one TFC workspace per environment. Each workspace points to the respective `/envs/<env>` folder (VCS-driven) or is targeted by GitLab (API-driven).

- **State isolation**: each workspace has its own state, variables, and policies.
- **Promotion**: merge to `main` triggers dev apply; tags or protected branches trigger uat/prod; approvals required.
- **Drift detection**: run scheduled `plan` in TFC; investigate any drift and fix via code.

---

7) CI/CD Options

Option A — VCS-driven (simplest)

- Connect TFC workspace to the GitLab repo/folder.
- Push/MR triggers **Plan**; merge to protected branch triggers **Apply**.

Option B — API-driven from GitLab (.gitlab-ci.yml)

```
stages: [plan, apply]

variables:
  TFC_ORG: "your-org"
  TFC_WORKSPACE: "snowflake-dev"
  TFC_TOKEN: "$TFC_TOKEN"   # masked CI var

plan:
  stage: plan
```

```
  script:
    - 'curl -s -H "Authorization: Bearer $TFC_TOKEN" -H "Content-Type:
application/vnd.api+json"
       -d @- https://app.terraform.io/api/v2/runs <<EOF\n{\n  "data": {\n
"type": "runs",\n    "attributes": {"is-destroy": false},\n    "relationships":
{\n      "workspace": {"data": {"type": "workspaces", "id": "${TFC_WORKSPACE}"}}
\n    }\n  }\n}\nEOF'
  when: manual   # manual on MR, or auto if you prefer

apply:
  stage: apply
  script:
    - echo "Enable auto-apply in TFC workspace or promote via policy/approvals"
  when: manual
  rules:
    - if: "$CI_COMMIT_BRANCH == \"main\""
```

Pick one model. VCS-driven is usually easier; API-driven gives more control in GitLab.

---

8) Coexistence & Migration from Flyway

Recommended split: - **Flyway**: app-centric DDL (tables, views, procedures) versioned per service. - **Terraform**: platform/RBAC/warehouses/stages/monitors + optionally baseline DB/Schema scaffolding.

Migration steps: 1) **Inventory** existing Snowflake objects and grants (script ACCOUNT_USAGE views). 2) **Create code** in Terraform to represent current state. 3) **Import** existing objects where supported: - e.g., `terraform import snowflake_warehouse.this NAME` 4) For non-importables: create matching resources with `lifecycle { ignore_changes = [...] }` temporarily, or recreate in lower env first. 5) **Freeze** manual changes; require MR approvals. 6) **Enable future grants** to reduce churn. 7) **Keep Flyway** for schema evolution; coordinate releases (Terraform first for access/infra, then Flyway migrations).

Gotchas: - Grant duplication across roles — centralize in modules. - `prevent_destroy` on critical objects. - Future grants do not retroactively apply to existing objects — handle both initial & future grants.

---

9) Cost & Safety Guardrails

- **Resource monitors** at account/warehouse level with notification/auto-suspend.
- Standardized **warehouse sizes** and **auto_suspend <= 300 sec**.
- **Sentinel policies** in TFC:

```
# Example Sentinel (pseudo)
import "tfplan/v2" as tfplan
```

```
main = rule {
  all tfplan.resources.aws_snowflake_warehouse as _, r {
    r.applied.auto_suspend <= 300 and r.applied.auto_resume is true
  }
}
```

- **Tagging/Naming** conventions enforced in modules.

---

10) Testing, Validation, & Observability

- **terraform fmt/validate**, **tflint**, **checkov/tfsec** in CI.
- **Pre-apply checks**: lightweight SQL probes via a service user (optional) to confirm role/warehouse usability.
- **Post-apply smoke tests**: query ACCOUNT_USAGE for created/changed objects.
- **Drift**: scheduled plans; alert on differences.

---

11) Day-2 Operations

- **Change management**: MRs with clear diffs from `terraform plan`.
- **Access requests**: implement via code (add a role grant → MR → apply).
- **Backups**: state is in TFC (encrypted, versioned); Snowflake has Time Travel/Fail-safe for data.
- **Rollbacks**: revert commit + re-apply; for grants, ensure idempotency.

---

12) Live Demo (suggested)

1) Create a small warehouse + DB + role in **dev**. 2) Merge MR; see TFC Plan/Apply. 3) Show Snowflake UI reflecting changes. 4) Promote to **uat** using the same module with different inputs.

---

13) Appendix: Minimal Working Example

```
# envs/dev/main.tf
module "warehouse" {
  source         = "../../modules/warehouse"
  name           = var.wh_name
  size           = var.wh_size
  auto_suspend   = 120
  auto_resume    = true
}

module "database" {
```

```
  source          = "../../modules/database_schema"
  db_name         = var.db_name
  schemas         = ["RAW","APP"]
}

module "rbac" {
  source = "../../modules/rbac"
  roles = {
    ROLE_APP = { inherits = ["SYSADMIN"], comment = "App role" }
  }
  db_usage_roles = ["ROLE_APP"]
}
```

```
# envs/dev/variables.tf
variable "snowflake_account" {}
variable "snowflake_user" {}
variable "snowflake_role" {}
variable "private_key_path" {}
variable "private_key_passphrase" { sensitive = true }
variable "wh_name" { default = "WH_DEV" }
variable "wh_size" { default = "XSMALL" }
variable "db_name"  { default = "APPDEV" }
```

---

14) What to Present (Slide-by-Slide)

1. **Title & Goals** — what we'll achieve.
2. **Current vs Target** — Flyway-only vs Flyway + Terraform.
3. **Architecture Overview** — GitLab ⇄ TFC ⇄ Snowflake.
4. **Provider & Auth** — key pair, least privilege.
5. **Modules & Layout** — reusable building blocks.
6. **RBAC & Grants** — future grants, role hierarchy.
7. **Environments** — workspaces, promotion model.
8. **CI/CD Model** — VCS-driven vs API-driven; sample pipeline.
9. **Migration Plan** — inventory → import → freeze → enforce.
10. **Guardrails** — Sentinel, resource monitors, naming.
11. **Testing/Drift** — linters, smoke tests, scheduled plans.
12. **Demo** — create a warehouse & DB; show plan/apply.
13. **Q&A & Next Steps** — pilot scope, timeline, owners.

---

15) Next Steps for Your Org

- Pick **service user** & auth method; create key pair.

- Stand up **TFC workspaces** (dev/uat/prod) + var sets + (optional) policy sets.
- Bootstrap **modules** (warehouse, db/schema, rbac, monitors).
- Migrate a **pilot domain** (one app/team) end-to-end.
- Document standards; enforce via Sentinel & code review.