Continuus TECHNOLOGIES

SOLUTION BRIEF

SNOWFLAKE INFRASTRUCTURE AS CODE

Maximize the time your engineers spend on solving problems, not setting up environments. Automate the creation, modification, and deletion of critical Snowflake objects with Continuus' Infrastructure as Code (IaC) for Snowflake.

CHALLENGE

Snowflake is a cloud-based data platform which proudly boasts near-zero management. This is certainly true; however, even if it does not require much in terms of patching, upgrades, or performance tuning, any sufficiently complex data platform will manage hundreds – if not thousands – of objects. In Snowflake some of these key objects include: Warehouses, Users, Roles, Databases, Schemas, Tables, Views, and the Grants that control access to those objects.

Creating, modifying, and deleting these Snowflake objects can be done in the Snowflake web interface or by running Snowflake SQL. Most Snowflake clients who migrate their analytics platform to Snowflake do all their Snowflake account configuration by hand. In the best case, they create runbooks with template SQL snippets; in the worst case, the process of configuring some aspect of their account was “institutional knowledge” that lived only in one engineer’s head.

This can leave clients with a Snowflake account that is working well, but is hard to adjust and even harder to understand.

SOLUTION

The basic premise of Infrastructure as Code (IaC) is that the state of your cloud resources should be defined by configuration files (typically checked into a version-controlled repository). Changes to the resources are made by updating those configuration files and “deploying” the change using a CLI. It’s best practice to run this in a CI/CD pipeline.

When using IaC, you can understand the state of your resources by simply reading the configuration files, and you can collaborate on changes to your resources in the same way as you would other code: with Git history, pull requests, code reviews, etc.

IaC tools were originally focused on managing the servers in a cloud computing environment, but today you can manage the configuration of basically any third-party tool your team uses.

Consider our Snowflake use case: we don’t need to configure the actual virtual machines our account uses; Snowflake does that for us! But we do want to manage key Snowflake resources like Users and Databases.

GitOps refers to using a Git repository as the single source of truth for all the code that goes into building infrastructure and deploying applications. By using a version control system such as Git as the single source of truth, engineers are able to update the underlying source code for their applications and infrastructure in a continuous delivery format.

GitOps automates infrastructure and code management using a Git workflow with effective CI/CD. After code is merged to the main branch, the CI/CD pipeline initiates the change in the environment.

Manual changes and human error can cause configuration drift in Snowflake environments, but GitOps automation and continuous deployment overwrites them so the environment always deploys a consistent desired state.

See Back for Benefits and Sample Technology Stack for IaC.

Continuus TECHNOLOGIES

SNOWFLAKE IaC BENEFITS + SAMPLE TECHNOLOGY STACK

AUDITABILITY

⚠ **Problem:** Answering questions about users, roles, and access is difficult and requires a highly privileged Snowflake user.

✓ **Solution:** Easily manage users, roles, and data access using tools like Flyway and Liquibase.

STANDARDIZATION

⚠ **Problem:** Which role/user should be used to create new objects? The role used by the creator matters because that role becomes the object owner and defines the default privileges on the object. Manual creation of objects is error-prone since an engineer might accidentally create an object using the settings of their unrelated last session.

✓ **Solution:** Deployment process always uses same role. All objects owned by same role. Further permissions are completed via the deployment pipeline.

COLLABORATION

⚠ **Problem:** A privileged user modifying a resource in the Snowflake console or by running SQL does so in a vacuum. While we trust that our teammates know what they're doing, we'd prefer to be able to have a formal code review process of these changes.

✓ **Solution:** All DML/DDL scripts are deployed via the CI/CD pipeline via the pipeline role. All changes are reviewed and approved prior to running. No changes are made manually without oversight or review.

HISTORY

⚠ **Problem:** All account changes are recorded and accessible via the QUERY_HISTORY view. However, parsing those historical logs is problematic for a several reasons: requires both SQL and Snowflake knowledge, there is a limited history of changes, and we'd much prefer the changes be version-controlled with something universal like Git.

✓ **Solution:** The code itself (in the repository) serves as the history with all changes captured for each release.

SPEED

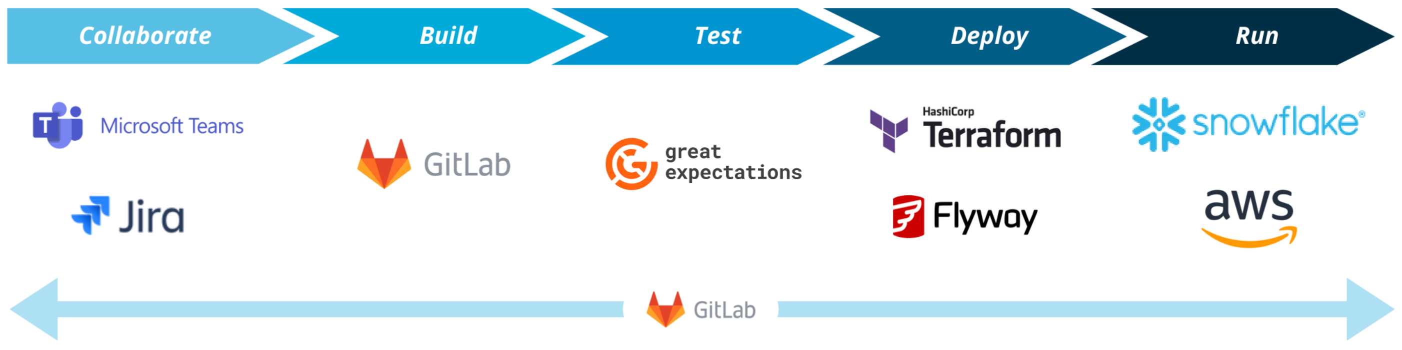
⚠ **Problem:** It's annoying and slow to change resources by hand. Our engineers' time is valuable; we shouldn't spend it on tedious tasks like writing the SQL command to adjust the default query timeout of a given warehouse.

✓ **Solution:** Once built, updates are incremental and save massive amounts of time that engineers can spend innovating and building new solutions.

COST

⚠ **Problem:** Engineers are spending an inordinate amount of time on tedious tasks like writing the SQL command to adjust the default query timeout of a given warehouse. Again, our engineers' time is valuable and we shouldn't spend it here.

✓ **Solution:** Stacking all the time savings, we end up saving the enterprise on engineering and consulting costs.



continuus-technologies.com