GoCD Design Document

Overview:

This document describes the design of the Continuous Delivery process for Axiad Conductor including, GoCD server configuration, authentication, backup and recovery, and CD's pipelines

This document is going to be split into two main categories

- GoCD Lifecycle: Everything in relation with the GoCD server itself and it's configuration
- CD Pipelines and Processes: All the pipelines, actions, and processes that are directly related to the deployment of Axiad Conductor

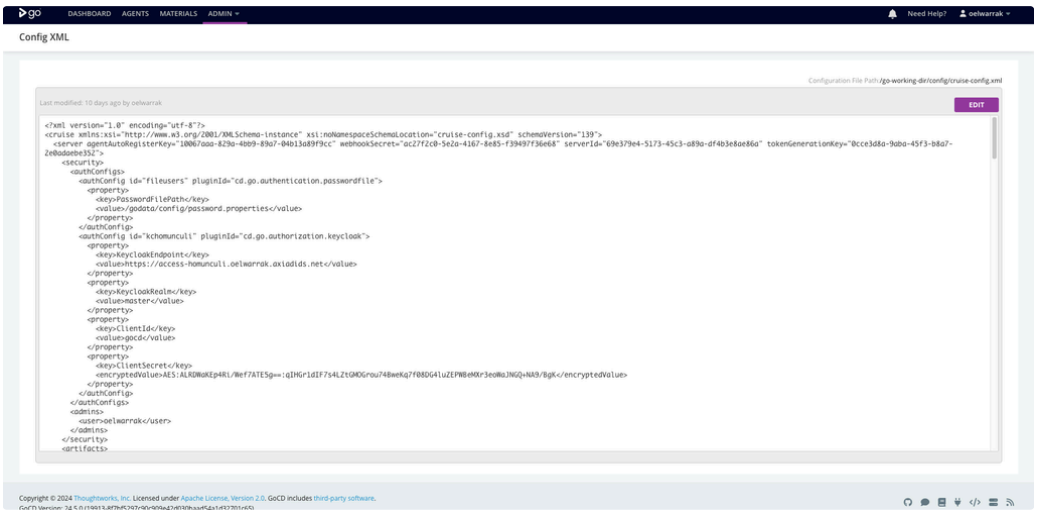
GoCD Lifecycle

In this section we will be describing the GoCD configuration and setup (including backup and maintenance)

1 - Configuration

GitHub - gocd-contrib/gomatic: A Python API for configuring GoCD

GoCD uses an XML based configuration file, that is stored and managed by GoCD server itself. Out of the box there is no automated way provided to load/update that configuration automatically from somewhere else, it does however provide an api endpoint to update that configuration.



In our case the plan is to build one extra pipeline for configuring GoCD Server, this pipeline when triggered will pull the GoCD config from git and any sensitive data, and upload it to the server using the API endpoint, this can be done through a couple of ways:

- Shell: we can use just shell inside the pipeline to get the config and use curl to upload the config to the server.
- Pyhon: GoCD has several Python library for handling the config, we can build the pipeline around one of them.

As of now the python way is prefered since while bash is simpler it would require other tools to be installed to do different things (curl, xmlstarlet or similar,...).

Conclusion: We need one pipeline dedicated to Server config loading.

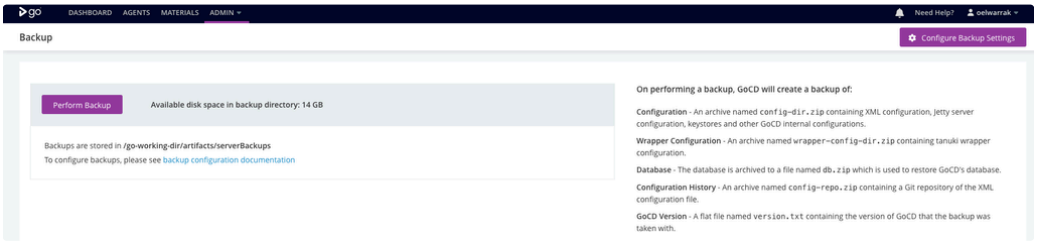
2 - Authentication/Authorization

GoCD has a Keycloak authentication plugin, which we will use to integrate with Corp Keycloak for authentication and authorization.

Conclusion: as part of GoCD server config we need to add the Keycloak Config.

3 - Backup/Recovery

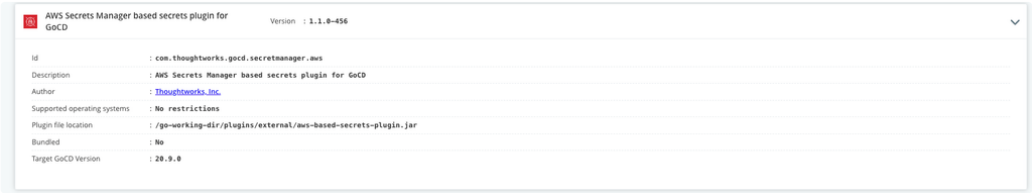
Out of the box GoCD offer backup restore features Backup GoCD Server | GoCD User Documentation , however it does not support any storage backends, and instead expects a post backup script to be provided. The script in question is in charge of copying all the backup files in any storage backend (S3 in our case).



Conclusion: Build a post-backup script to upload backups to s3, and a restore script.

4 - Sensitive Data

As a rule, all sensitive data/secrets should be coming from AWS Secret Managers, this applies to any secrets needed by GoCD. This is done through the AWS Secret Manager Plugin of GoCD



The list of secrets includes but is not limited to Git Credential, Keycloak Client Secret, any secrets needed by the pipeline of the CD process itself. for details information about secrets see Secret Management | GoCD User Documentation

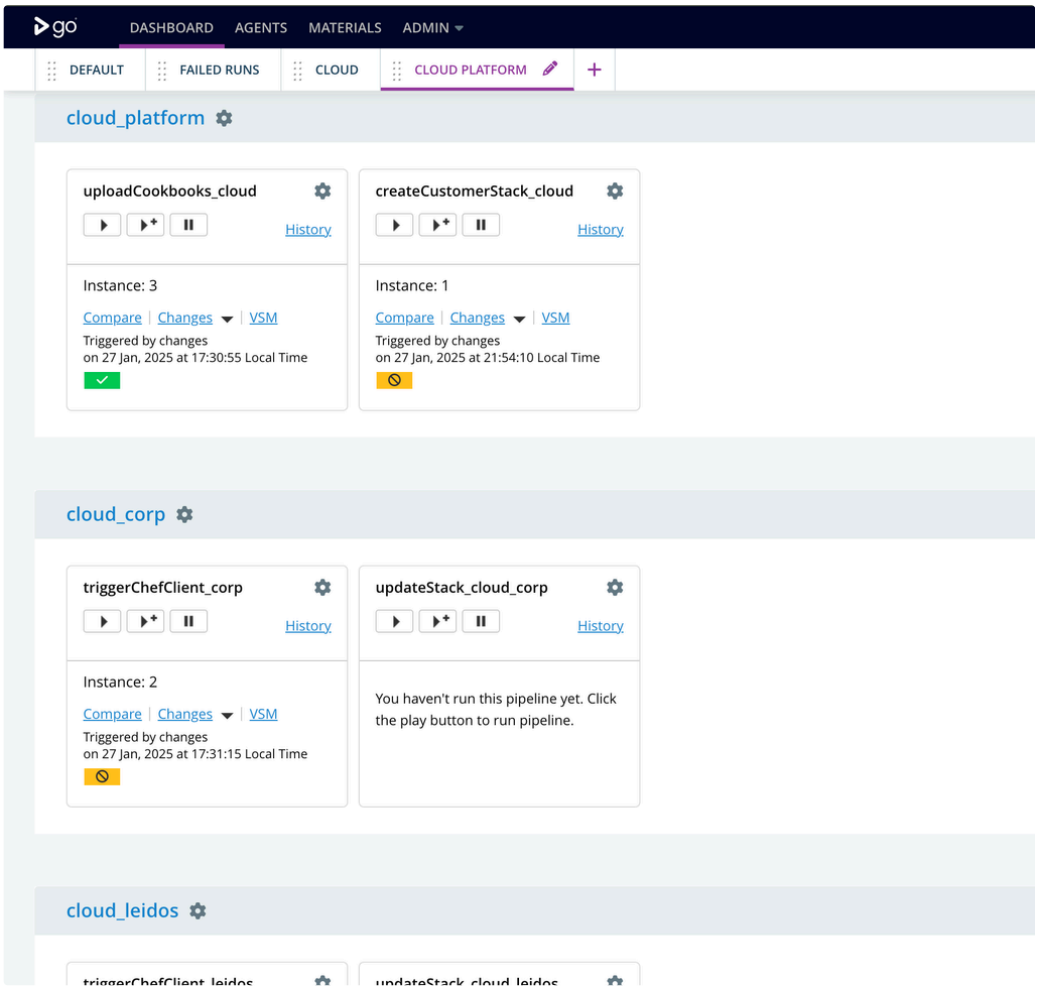
CD Pipelines and Processes

For our CD needs we will relying on GoCD Jsonnet config plugin, since we have a large number of customers and and multiple platforms this will allows us to manage pipelines for all those environments using Jsonnet templates

1 - Environments:

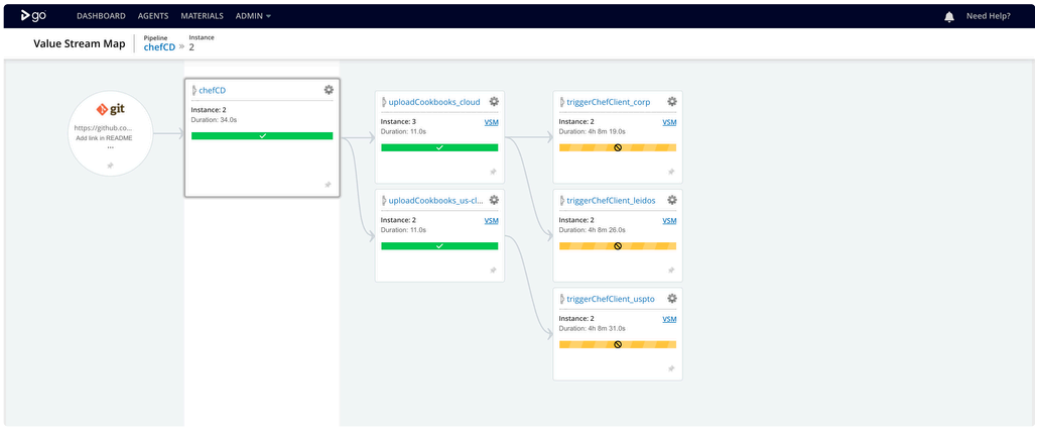
The idea here is to have one GoCD environment per platform (cloud_platform and us-cloud_platform) and one GoCD environment per customer in those platforms (cloud_corp for example) this way we would have operations segregated by

environments, and triggering a pipeline for a customer is as simple as going to that customer and pressing run on the pipeline.



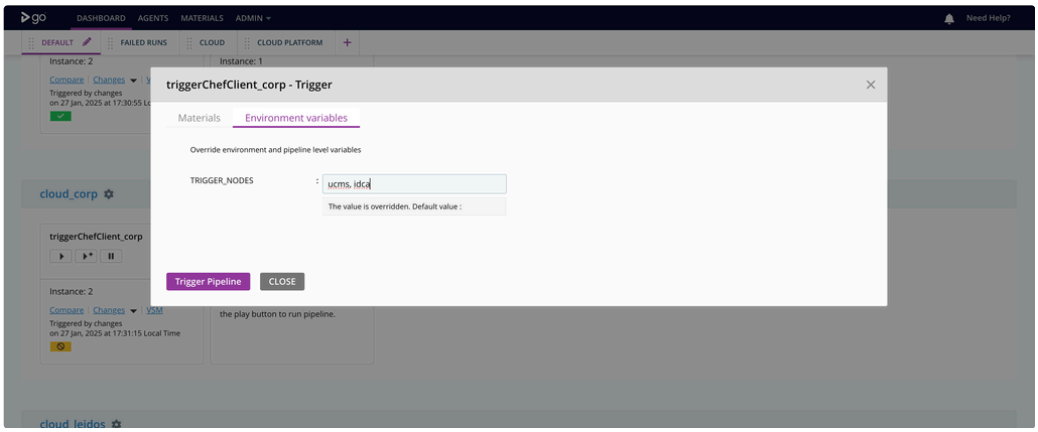
2 - Chef CD

First part of CD is chef pipeline where we are uploading the cookbooks and triggering the chef client, a run tree would look similar to the following



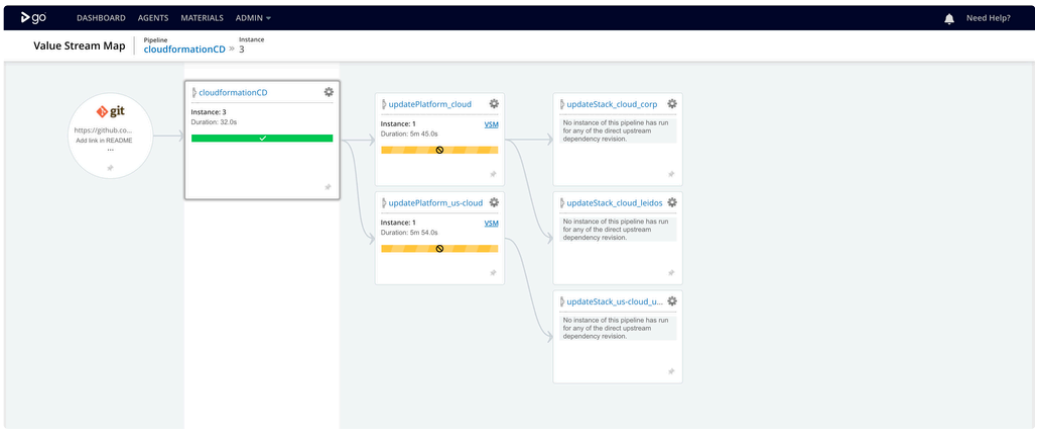
The first pipeline is a wrapper part of a root global env called axiadConductor, which will trigger upload cookbooks in all platform stacks namely (cloud and us-cloud) which in turn will trigger the chef client run for all the customers. it is

possible to trigger only one branch of the tree let’s say just cloud platform branch, or even a single customer. Triggering chef on a single node will be handled by environment variable when triggering a customer.

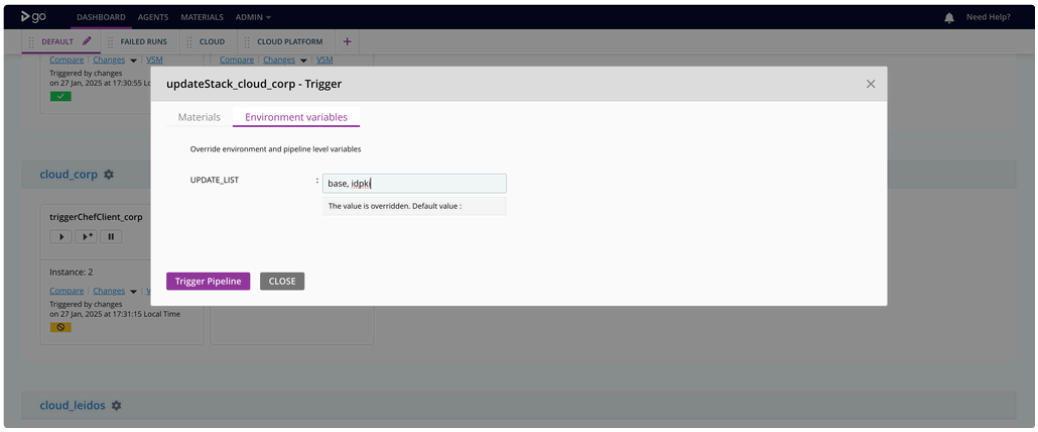


3 - Cloudformation

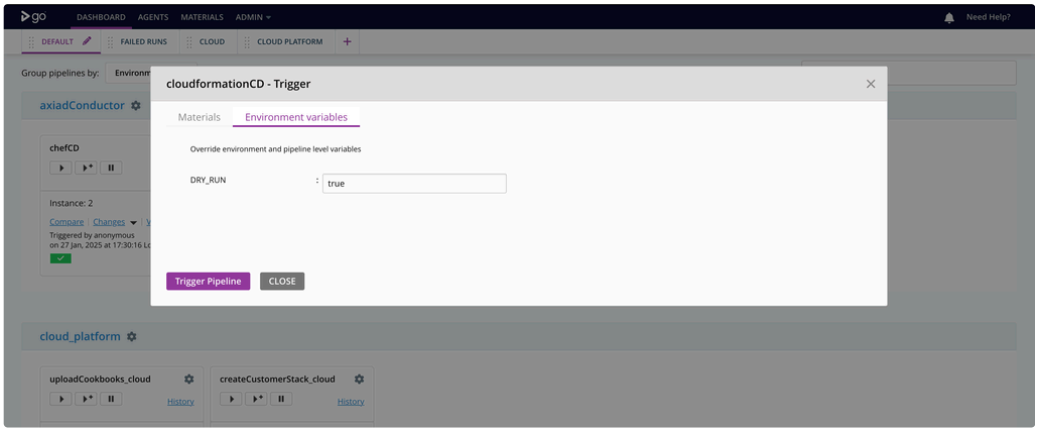
Similar to chef, we have a pipeline tree for update stack, that updates platforms first, then customer base, then asgs



And similarly, we can trigger only one subset of the tree to the single customer. A more fine-grained control is given over a single customer update, for the case where we only want to update let’s say a specific ASG



One notable difference is the cloudformationCD pipeline also supports a DRY_RUN variable that would allow us to run the pipeline ahead of time and only generate template diffs for validation prior to a real update.



We also provide self-standing pipelines for platform and customer stacks creation, as well as start and stop stack pipelines.

4 - Agents

We will use ephemeral agents to instantiate based on our need, similar to what we do for Jenkins.

we need to build customer docker images for the agents with our dependencies (python, Chef knife, ssh,...)
https://hub.docker.com/u/gocd

5 - Pipeline generation

Since our list of environments (customers) and thus the list of pipelines is dynamic, we need to dynamic generate those pipelines based on customer configs, this can be achieved either using the Jsonnet plugin, or through the Server config pipeline mention in the GoCD server config section above.

Deliverables:

- Agent Images.
- GoCD Configuration as code (server config + envs definitions).
  - Server config Pipeline.
  - Keycloak integration + user roles.
  - AWS Secret Manager integration.
- Backup post-script and restore script.
- Chef pipelines (auto generated).
- Cloudformation pipelines (auto generated).
- Deploy in dev account and test with demo before going to prod.