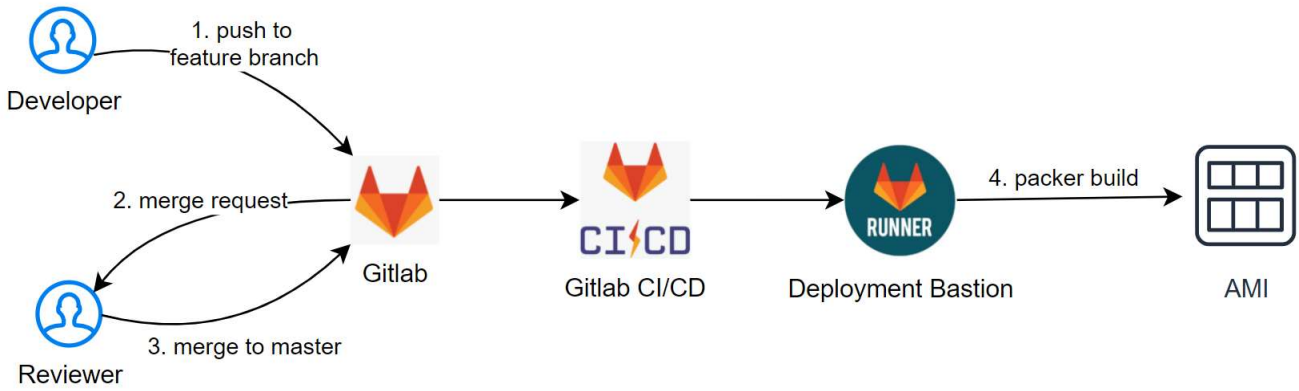
AMI Build Process

Summary

In AWS, code is deployed along with infrastructure in a single release. The process relies on updating an Amazon Machine Image (AMI) with software updates and dependencies and pushing out the new AMI to application / web auto-scaling groups. With a goal of running immutable servers in AWS, our release and rollback procedures are streamlined through the bundling of code, application, and OS updates into a unique version controlled AMI.

Deploying an AMI

AMIs are deployed using Gitlab CI/CD pipelines. The pipelines are configured in the AMI projects `.gitlab-ci.yml` file.



The `.gitlab-ci.yml` file manages the deployment of an AMI end-to-end. A Docker container acting as a GitLab Runner (with terraform / chef / packer also available) will launch on the Deployment Bastion server and process the deployment request. We can configure multiple flavors of the deployment Docker Container to meet the needs of a specific project, and reference which container is to be used when a deployment through the pipeline is initiated.

Where secrets are necessary, the GitLab Runner on the Deployment Bastion will communicate with Vault to request and apply secrets from our Vault store at deployment time. The full deployment process is described as .yml, allowing us to version-control changes to the deployment process for individual processes. This improves our ability to track and audit changes.

Rolling Back

In the event that a release does not go as planned a roll-forward approach is the preferred method of stabilizing the environment. This can manifest itself in two ways: make necessary updates to remediate the cause of the problematic release and push a new AMI to production or re-release a previous deployment. In either case, both actions will result in a new AMI being launched with the appropriate code and applications included.

There are many reasons for this approach:

1. Attempting to manually undo the changes made as part of a release is a timely process that is not guaranteed to leave the environment exactly as it was pre-release.
2. Swapping a newly released AMI for the version previously in use will not address other environmental changes that need to occur outside the AMI. For example, a database schema update that is made in parallel with a code deployment isn't addressed by reverting to the previously used AMI.
3. Because Terraform is a declarative language, re-deploying a previous release will re-define the environment as it functioned previously.
4. Roll-forward preserves the problematic configuration and datasets for potential forensic investigation.

Stack Cookbooks, Not AMIs

During Phase 1 of the cloud project, we learned that a complex hierarchy of AMIs will become difficult to change, and routine updates will have the potential to cause cascading environmental problems. This hierarchy also creates an unnecessarily complex environment in which to have to perform regression testing.

Instead of building custom AMIs on top of other Direct Supply managed AMIs, our strategy will be to use stacked Chef cookbooks to apply the necessary features and configurations at deployment/build time. For standardization purposes, all AMIs should run either the windows-base or linux-base cookbooks, in addition to application specific cookbooks.

See the shared cookbooks architecture page for more information.

Example Pipeline

- Sample AMI Git Repository
- Deployment

Creating an AMI

See this guide for information on how we create AMI's.

AMI Cleanup

Cloud Custodian, our tool for cloud policy enforcement, takes care of cleaning up old AMIs. The current Custodian policy for cleaning up old AMIs looks for AMIs that meet the following criteria:

- More than 60 days old
- Not in use by any existing EC2 instance

The policy, which executes every 5 minutes, will mark matching AMIs for deregistration within 7 days. To protect an AMI from deregistration, add a tag with a *case-sensitive* key of `RetainIndefinitely` and a *case-sensitive* value of `true`.