Notes on CX Cloud CI/CD

0.1.0

Joshua Dotson

Build Date 2020-05-04

Table of Contents

1 Changelog

- **0.1.0** (josdotso@cisco.com)
  - Initial version.

2 Assumptions

1. Every git repo MUST of type `ci` (OR) `cd`.
2. CI git repos are those which produce an artifact that MAY be deployed later by one or more CD git repos. Artifact examples might be: container image, TAR.GZ file, read-only JSON blob, web content such as a Single Page App (SPA).
3. If the git repo doesn't contain Terraform or similar deployment code, it's 99% probable that it is a CI git repo.
4. Terraform is the deployment mechanism of choice, as it can deploy all the things to all the clouds and even to non-clouds (e.g. Kubernetes).
5. CD does not deal at all with "in the server" configuration management, so favor immutable deployment models (e.g. cloud-init for EC2), where an instance is replaced in full instead of config-altered in place (e.g. EKS kubernetes clusters)
6. Secrets SHOULD be stored in AWS Secrets Manager in at least one region if not two (TBD by Ryan).

- TODO (Ryan): Raise a question to AWS team to see if they have an approved and reliable secret replication mechanism

3 Continuous Integration (CI)

1. Use CircleCI for CI only.

2. Container image is the CI artifact kind that gets our highest level of support. Every other kind of CI artifact comes second to container image.
3. Helm Charts are not as important as Docker images as priority and support are concerned.
4. CI doesn't "deploy" anything other than ephemeral test environments and these are for its own use and nobody else's.
5. Ephemeral test environments are squarely in the domain of CI. They are not CD even if they reuse the automation code of CD (SHOULD: be fully decoupled from CD). That is, testing is CI and simulating stage and prod (standing environments) is completely in the CI (NOT CD) pipeline. For example, if some test of a Docker image in its epehemeral test envirnonment fails, then we shouldn't even push the artifact as "ready to use".
6. Security scanning of artifacts (e.g. Docker images) is completely in the domain of CI.
7. Use whatever security scanning tools mandated by Cisco security compliance. If none are mandated, use something open source and modern or (exceptionally) easy to use and paid/closed-source.
8. All container images MUST be pushed to two regions of AWS ECR to ensure we can reasonably endure 1-region outages.

- TODO (Ryan): Find out if AWS has a way to do this replication

4 Continuous Deployment (CD)

1. Try and probably use Atlantis + terraform-jsonnet as the default CD toolkit.
2. Defer choice on how to deploy to Kubernetes until a later date, because we need Terraform CD, FAST. There really are only two contenders for that space anyway: Atlantis and Flux. TODO (Ryan)
3. CD repos are in the "direnv deploy" format. That is a git repo with directories like `clusters/mycluster.example.org/infra` and `clusters/mycluster.example.org/apps` where the direnv tool reads `.envrc` files in each directory to allow users to use it interactively if need be.
4. Deployment aka. CD git repos SHOULD keep everything on master branch.
5. CD git repos MUST be watched by CD tools (e.g. atlantis) for changes. Upon change detected, CD should cause deployments to converge on the state declared by their declarative source code (e.g. written using terraform-jsonnet).
6. Tag bumps (e.g. container image tags) SHOULD be done manually for the near term, contrary to the pattern set by Flux. Flux detects semver patterns and writes back to git, we do NOT favor that pattern.
7. All terraform state MUST be backed by encrypted with versioning S3 and DynamoDB locking

tables that are management by cfn-jsonnet.

8. PR-driven CD is the goal. That is, to get anything deployed to stage and prod, one must do so by going through the PR and review + merge process.
9. Nobody should be running terraform from their laptops nor should they be reading (stage, prod) secrets directly from AWS Secrets Manager if we did our jobs right.
10. There SHOULD be only two standing environments for CX Cloud: stage and prod. This mandate prevents a (very expensive in OpEx) proliferation of unnecessary standing environments. Encouraging such a proliferation would allow teams to become so siloed as to hide ripple effects of potential state changes.
11. Side-effects of sharing stage and prod across teams MUST be embraced as a learning opportunity rather than a reason to run from the realities of how prod really works (dependencies become real).
12. Personal 1-person dev enviornments are not to be considered “standing environments” with respect to the above mandate. However, each 1-person dev envionment MUST be clearly identified as owned/maintained by an individual person (read as: person to nag about idle usage).