CI/CD Roadmap

Jacob Archambault

# 1 The problems we’re facing that CI/CD should solve

## 1.1 Problems our current cloud migration plan exacerbates

1. Developers spend too much time on deployments and environment management relative to time spent coding solutions to business needs.
2. Environments based on different branches quickly and easily get out of sync with each other both through lack of knowledge and corner-cutting
3. Data disparities between environments can make replicating a problem on production time-consuming for staging and local environments.

## 1.2 Existing problems our current plan fails to address

1. The cycle between finishing a feature and seeing it in production is much too long (up to 3 weeks), making it much harder to know what caused a break when something does break.
2. *There are cases where our branching workflow, by design, will break at least one main branch*. A common occurrence of this is when there are concurrent NuGet package updates in an API respectively branching off of both master and develop.
3. Our git flow and lack of CI/CD tooling is a flight risk. I know personally that it was a factor in at least three moves from the company over the last year.

# 2 Solution, immediate term

* [GitHub flow](https://githubflow.github.io/) – this requires no more resources than we’re currently using, and should be in principle adoptable immediately
  + One main branch: master (not two or three).
  + Eliminate the distinction between features and hotfixes. Everything gets merged into master.
  + **Rule:** Master must always be in a deployable state (we should be doing this anyway).
* **3 Environments** 
  + Pre-release.
    - Based off of master
    - Automatically deploys on merge to master. Manual deployments aren’t allowed.
    - Available for viewing by QA testers, developers, *and stakeholders* for UAT testing.
    - Nightly integration testing and scrubbed DB backups from production to ensure environment is as close to production as possible
    - Use NuKeeper or similar tool for automating PR’s for NuGet package updates.
  + Production
    - Just like pre-release, but manual deployments for Orca and our websites.
    - Given bugfixes, manual deployments will likely happen at least once every few days.
    - Automatic deployment for NuGet package libraries and APIs.
    - Release commits are tagged and versioned on master for referencing.
  + Pull request
    - Deployed from a PR branch when ready for review before merging
    - Both code *and* any *functionality to be demonstrated* must be reviewed at code review time.
      * For PR’s a second approval may come from a QA who has tested the functionality.
      * At least one PR approval must come from a dev with 1 or more years’ experience at the company.
    - Code reviews must pass all unit tests to be eligible for merging.
    - Environment automatically updated on merge to master.

# 3 What this solves

1. Eliminates the need for backmerging and multiple deployments. Instead, this is taken care of naturally by the commit history.
2. Fewer environments means fewer out of sync environments. It also cuts costs.
3. Identical or near-identical data between pre-release and production.
4. Pre-merge functionality testing and more frequent deployments means fewer surprises.
5. Reducing the number of main branches to one greatly minimizes the problem of merge conflicts caused by concurrent NuGet package updates.
6. Decouples thinking about sprint planning from thinking about branching and deployment strategy.
7. Modernizing will improve talent retention

# 4 Risks

1. Requires slightly more discipline to avoid breaking the main branch.
   1. This is worth it because the discipline required is something we should be encouraging anyway.
   2. Another mitigating factor here could be more active mentorship, e.g. through pair programming.