Continuous Integration and Continuous Deployment for LARI

Introduction:

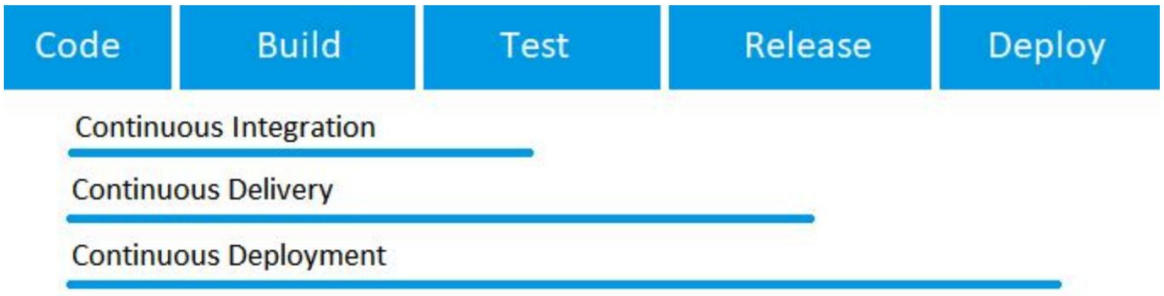
Continuous Integration, Delivery, and Deployment are relatively new development practices that have gained a lot of popularity in the past few years. Continuous Integration is all about validating software as soon as it's checked in to source control, more or less guaranteeing that software works and continues to work after new code has been written. Continuous Delivery succeeds Continuous Integration and makes software just a click away from deployment. Continuous Deployment then succeeds Continuous Delivery and automates the entire process of deploying software for consumer use.

This paper is a follow-up guide for someone to take over the CI/CD setup process for LARI in the AFSL lab

Requirements:

- Source Control e.g git
- CI Server
- Unit Tests

The cycle of CI/CD goes like this:



Steps:

Continuous Integration:

A VM is needed to run the integration. The operating system for the VM should be a Linux Ubuntu (other Linux versions should work too). For version controlling, git must be installed. Repository services like Gitlab provide free ci/cd pipeline services, and I've been looking into how to integrate LARI from the GitHub project to the gitlab ci/cd system. For Gitlab, there needs to be a gitlab runner that needs to be installed in the linux (https://docs.gitlab.com/runner/). Gitlab runner runs the job of building the project. It takes in a yaml file that can be configured to run a script in the linux sandbox.

This link has good documentation on how to install gitlab on your ubuntu/linux server: https://about.gitlab.com/install/#ubuntu1604

A sample yaml (https://en.wikipedia.org/wiki/YAML) file should have the following script. This here is an example of a project built on Ruby on Rails:

```yaml
image: "ruby:2.5"

before_script:
  - apt-get update -qq && apt-get install -y -qq sqlite3 libsqlite3-dev nodejs
  - ruby -v
  - which ruby
  - gem install bundler --no-document
  - bundle install --jobs $(nproc)  "${FLAGS[@]}"

rspec:
  script:
    - bundle exec rspec

rubocop:
  script:
    - bundle exec rubocop
```

For the yaml file to run, a file named `.gitlab-ci.yml` should be in the root directory of the gitlab/github repository.

Once GitLab Runner is installed, it needs to be registered with GitLab. Steps are outlined here: https://docs.gitlab.com/runner/register/index.html

My research work has lead me to successfully build the runner agnostic of LARI's requirements. Now since LARI is a C# project, after building the runner it needs to be able to build the C# project. I'm not sure what that requirement entails to, but further research is needed in that case.

My next steps would be to use the VM to build the LARI project. Then unit tests can be incorporated within the yaml file to have iterative tests to check different functionalities of LARI before it's being deployed.

Continuous Deployment:

The continuous deployment here means running the software on a cloud platform. From my experience, having an AWS EC2 instance works well, since it allows the program to run on a data center in Amazon, which allows the devs to not worry about server management. Gitlab has good support for integrating into AWS systems and can be done through registering the GITLAB runner in the AFSL AWS account (if permitted to use).

Once the whole environment is set, devs working on LARI can push their changes to git. If the YAML is set correctly, this will trigger the Linux to build the project, do unit testing and deploy it to a server by the command of one line.

In conclusion, Continuous Deployment helps in getting software out to the customer as soon as it is written. Continuous Delivery is a good alternative if we need more control over the deployments. To minimize the risk of deploying bugs, the software should be thoroughly tested using Continuous Integration. Continuous Integration is all about making sure the software is tested and deployable.