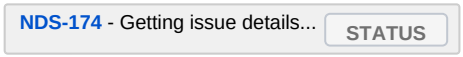
Developer Workflows

All documentation pertaining to how developers can contribute to NDS Labs.

New to NDS Labs?

Start here: New Developer Workflow

Develop Workflows

- JIRA Workflows: Issue and project tracking workflows
- Git Workflows: Forking workflow with feature branches
  1. Fork repo (if applicable)
     - Press "Fork" in GitHub UI
  2. Clone repo to make changes locally (if applicable)
     - git clone https://github.com/USERNAME/ndslabs.git
     - git remote add upstream https://github.com/nds-org/ndslabs.git
  3. Ensure correct branch and sync with upstream before making additional changes
     - git checkout master
     - git pull upstream master
  4. Create a branch named after the **Story** (for example 

)
     - git checkout -b NDS-174
  5. Make any necessary modifications locally on your branch
  6. Stage any modified files for commit
     - git add path/to/modified/file.ext
  7. Commit any modifications to your local branch with a comment
     - git commit -m "A comment about this commit"
  8. Push any local commits back up to your remote branch (your forked repo)
     - git push origin NDS-174
  9. When you are satisfied with your set of commits, create a Pull Request (PR) to view the diff
     - Press "Pull Request" in GitHub UI
     - Be sure to select the correct base and compare branches
       - Select **nds-org/ndslabs** as the *base fork*
       - Select **master** as the *base branch*
       - Select your personal fork (**USERNAME/ndslabs**) as the *head fork*
       - Select your personal **Story** branch as the *compare branch*
     - Scroll down and click on the "Files Changed" tab to briefly review your own Pull Request
       - Ensure that all changes made on this branch were intentional
       - If you are unsure about any specific code segments, comment in-line on the PR to ask for clarification
       - If you are unsure about any general concepts changed or introduced, comment in the section at the bottom of the PR
     - Name your Pull Request after the **Story** / branch (i.e. "NDS-174: User can access console of running service via CLI")
     - Enter a short description of any modifications, additions, or removals from the codebase
       - If applicable, include a Test Case that the reviewer should run before merging the Pull Request
     - Click "Create Pull Request"
- Docker Workflows: Push any necessary test images to Docker Hub
  1. Build test image
     - docker build -t ndslabs/apiserver:dev .
  2. Tag test image with **Story** id (i.e. NDS-174)
     - docker tag ndslabs/apiserver:dev ndslabs/apiserver:NDS-174
  3. Push test image to Docker Hub
     - docker push ndslabs/apiserver:NDS-174
- Kubernetes Workflows: Sometimes used in testing new services or the API server

An Example

https://github.com/bodom0015/developer-workflow

More Detail

- New Developer Workflow
- JIRA Workflows
- Git Workflows
    - Forking Workflow
    - Feature Branches
    - Resolving Merge Conflicts
    - Splitting a Repository While Preserving History
- Docker Workflows
    - Installing Docker
    - Using Docker Images
    - Building Custom Docker Images
- Kubernetes Workflows
    - Downloading kubectl
    - Deploying a Cluster
- Multi Node Kubernetes developer environment - using VBox/Vagrant/CoreOS

Release Workflows

New Unstable "Test" Release

Prerequisites

- Pull request has been created containing the changes to be reviewed / tested
- Ensure that associated JIRA ticket contains a test case
- Ensure that your current code passes the test case that you have written
- Ensure documentation in Confluence is up-to-date
- Checkout your feature branch
    - git checkout master
- Sync with upstream
    - git pull upstream master
    - git push origin master
- Update any relevant documentation in GitHub

Developer's Process (Semi-automatic)

1. "Start Progress" on one of your assigned tickets (assign a new one if you have none assigned)
2. If you haven't already, fork the upstream repository (you will only need to do this once per repository)
    - Set up an automatic build of your new fork on DockerHub
    - Configure the build to build all new branches from GitHub (choose "Branch" and leave the branch name blank)
    - Push these new branch builds to a Docker image tag of the same name (simply leave the tag name blank)
    - Once saved,
3. Clone your fork onto your local machine
4. Create / switch to a development branch (named after one the JIRA ticket associated with the work being done, i.e. NDS-XXX)
5. Make any necessary changes to fulfill the JIRA ticket
6. Commit all associated changes and push them to GitHub
    - Any new changes pushed to any branch on your GitHub fork will be automatically built into an image of the same name on DockerHub
7. Mark ticket as "In Review" and assign to an available Tester
8. Wait for the ticket to be assigned back to you
9. Review the Tester's results
    - If the Tester encountered problems, choose **Review Rejected** go back to #5 and address them
    - If the Tester submitted comments or feedback, do your best to address their concerns or comment back to come to consensus
    - If both Developer and Tester

Tester's Process (Manual)

- Ensure that all associated auto-build images have completed their builds before beginning testing
    - Links to these images should be provided with the test case.
- Run through the test case described in the associated ticket(s)
    - A test case should be provided in the comments of each ticket, where appropriate. If it is not, send it back to the Developer.
    - The test case should include success and / or failure criteria. If it does not, send it back to the Developer.
- Update the associated JIRA tickets:
    - Briefly include the results of your testing
    - Be sure to leave feedback for the developer if you need them to take action
    - If something went wrong or the Tester still has questions, choose **Review Rejected,** assign it back to the Developer, and wait for a reply or the ticket to comeback to you

    - If all test cases pass according to the standards set in the ticket and its comments, choose **Review Accepted** and assign it back to the Developer
- After all tickets in the associated JIRA tickets are marked as **Resolved**, merge the Pull Request into the **master** branch on the nds-org GitHub
    - Merging any PRs to **upstream master** will automatically trigger a build of "latest" on DockerHub (see below)

New Unstable "Latest" Release

Prerequisites

- Any related PRs have been merged to master
- Ensure that smoke test passes
- Ensure documentation in Confluence is up-to-date
- Checkout your master branch
    - git checkout master
- Sync with upstream
    - git pull upstream master
    - git push origin master
- Update ALL documentation in GitHub

Process (Automatic)

- New "latest" Docker images are automatically built from the **upstream master** branch on GitHub.
- All new changes that make it into master on GitHub will automatically trigger a build on DockerHub
    - For example: https://hub.docker.com/r/ndslabs/apiserver/builds/

Official Tagged Version Release (Stable)

Prerequisites

- Ensure that all tests pass
- Ensure documentation in Confluence is up-to-date
- Checkout your master branch
    - git checkout master
- Sync with upstream
    - git pull upstream master
    - git push origin master
- Update ALL documentation in GitHub

Process (Semi-automatic)

1. For each repo: branch off of `develop` to create a release branch
    - ndslabs and workbench-helm-chart

      OR

    - kubeadm-bootstrap and kubeadm-terraform
2. Roll versions forward
    a. ndslabs
        i. gui/swagger.yaml: NDS Labs swagger API spec version number
        ii. apiserver/build.sh: NDS Labs API Server Docker image version tag
        iii. apiserver/cmd/clientVersion.go: NDS Labs CLI version number / build date
        iv. gui/Dockerfile: NDS Labs UI / webserver Docker image version tag
        v. gui/package.json: NDS Labs UI / webserver NPM package version number
        vi. gui/ConfigModule.js: NDS Labs UI Angular app build version number
    b. workbench-helm-chart
        i. values.yaml: NDS Labs API Server + UI Docker image version tags
    c. kubeadm-bootstrap
        i. install-kubeadm.bash: Kubernetes / Docker version numbers
        ii. init-master.bash: Helm version numbers
    d. kubeadm-terraform
        i. kubeadm-bootstrap git release/tag: assets/bootstrap.sh
3. PR from release branch to master
    a. Thorough testing, then more testing, then merge to master
4. Tag master with new version number (freshly tested and stable)
    a. Any new merges and tags will trigger new Docker images to be built
5. Wait for newly-tagged resources to automatically finish building and pushing Docker images
    a. Run a quick smoke test with newly-tagged resources
    b. Fix any last-minute errors directly on master and recreate release
6. Backport any missing changes from release-x.x.x into develop
    a. This should include, at the very least, a commit from the release branch that rolls forward to new version numbers
    b. git checkout develop && git pull origin master && git push origin develop? Why does this not work with a PR...

Legacy Process

1. Regenerate Swagger API / Client from spec (this can be skipped if the spec has not changed)
    - ~~apiserver/???: generated Go swagger server~~
    - gui/js/app/shared/api.js: generated AngularJS swagger client
2. ~~Roll forward version numbers in **ndslabs-deploy-tools** and ensure that all values match **the version number you are about to create**:~~
    - ~~roles/cluster-backup/defaults/main.yml~~
    - ~~roles/ndslabs-api-gui/defaults/main.yml~~
    - ~~roles/k8s-nagios-nrpe/defaults/main.yml~~
    - ~~roles/k8-glfs-server-pods/defaults/main.yml~~
    - ~~roles/k8-glfs-client-set/defaults/main.yml~~
3. Create a new tag from master in GitHub for the new version (i.e. 1.0.0, 1.0.1, etc):
    a. Repositories should be tagged in the following order when possible:
        i. ndslabs (API server / REST API / CLI / UI)
        ii. ~~ndslabs-specs~~ (service specs - starting with 1.2.0, this can be versioned separately from Workbench, but it should be noted upon a new release which version of Workbench the specs release will target)
        iii. workbench-helm-chart (Helm deployment to Kubernetes)
        iv. ~~gluster~~ (global file system - deprecated, no longer used)
        v. ~~cluster-backup~~ (cron job for backing up glfs / etcd / kubectl dump - starting with 1.2.0, this can be versioned separately from Workbench)
        vi. ~~ndslabs-nrpe~~ (nagios monitoring - these can now be versioned separately from the rest of Labs Workbench)
        vii. ~~ndslabs-startup~~ (dev-cluster startup - deprecated, no longer used)
        viii. ~~ndslabs-deploy-tools~~ (ansible scripts - deprecated, no longer used)
        ix. kubeadm-bootstrap (kubernetes deployment scripts - versioned separately, but it should be noted upon a new release which version of Workbench the release will target)
        x. kubeadm-terraform (terraform deployment procedure - versioned separately, but it should be noted upon a new release which version of Workbench the release will target)
        xi. ~~ndslabs-devenvs~~ (developer environments - these can now be versioned separately from the rest of Labs Workbench)
            - **NOTE**: ndslabs-devenvs contains a large number of cascading images that will quickly fill up the build queue, that's why we do it last
    b. New versioned Docker images are automatically built from the **upstream** tags created on GitHub.
    c. All new tags that are created will trigger a build
        - For example: https://hub.docker.com/r/ndslabs/apiserver/builds/
4. Roll forward version numbers in source and ensure that all values match on **upstream master** on GitHub:
    - **Swagger API**
        - ~~apis/swagger-spec/ndslabs.yaml: NDS Labs swagger API spec version number~~ (deprecated - use the file below instead)
        - gui/swagger.yaml: NDS Labs swagger API spec version number
    - **API Server**:
        - apiserver/build.sh: NDS Labs API Server Docker image version tag
        - ~~apiserver/version.go: NDS Labs API / Server version number~~ file no longer exists
    - **CLI Client**:
        - ~~apictl/build.sh: NDS Labs CLI version number~~ file no longer exists
        - ~~apictl/cmd/clientVersion.go: NDS Labs CLI / API version number~~ file no longer exists
    - **UI Client**:
        - gui/Dockerfile: NDS Labs UI / webserver Docker image version tag
        - gui/package.json: NDS Labs UI / webserver NPM package version number
        - ~~gui/bower.json: NDS Labs UI Angular app Bower package version number~~ file no longer exists
        - gui/ConfigModule.js: NDS Labs UI Angular app build version number