Apache Hudi - Release Guide

Introduction

This release process document is based on Apache Beam Release Guide and Apache Flink Release Guide

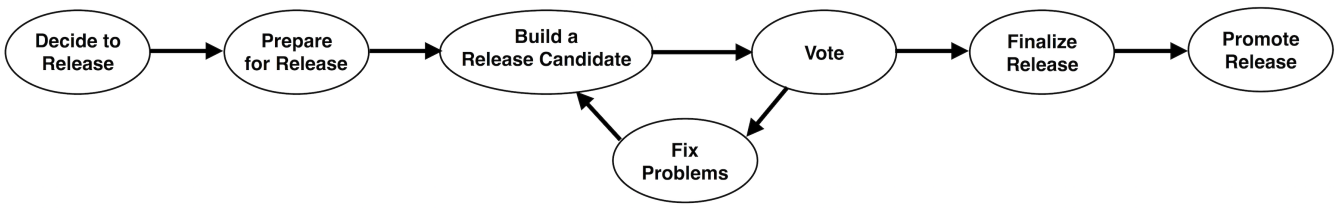
The Apache Hudi project periodically declares and publishes releases. A release is one or more packages of the project artifact(s) that are approved for general public distribution and use. They may come with various degrees of caveat regarding their perceived quality and potential for change, such as “alpha”, “beta”, “stable”, etc.

Hudi community treats releases with great importance. They are a public face of the project and most users interact with the project only through the releases. Releases are signed off by the entire Hudi community in a public vote.

Each release is executed by a *Release Manager*, who is selected among the Hudi PMC members. This document describes the process that the Release Manager follows to perform a release. Any changes to this process should be discussed and adopted on the dev@ mailing list.

Please remember that publishing software has legal consequences. This guide complements the foundation-wide Product Release Policy and Release Distribution Policy.

Overview



The release process consists of several steps:

1. Decide to release
2. Prepare for the release
3. Build a release candidate
4. Vote on the release candidate

5. During vote process, run validation tests
6. If necessary, fix any issues and go back to step 3.
7. Finalize the release
8. Promote the release

Decide to release

Deciding to release and selecting a Release Manager is the first step of the release process. This is a consensus-based decision of the entire community.

Anybody can propose a release on the dev@ mailing list, giving a solid argument and nominating a committer as the Release Manager (including themselves). There's no formal process, no vote requirements, and no timing requirements. Any objections should be resolved by consensus before starting the release.

In general, the community prefers to have a rotating set of 3-5 Release Managers. Keeping a small core set of managers allows enough people to build expertise in this area and improve processes over time, without Release Managers needing to re-learn the processes for each release. That said, if you are a committer interested in serving the community in this way, please reach out to the community on the dev@ mailing list.

Checklist to proceed to the next step

1. Community agrees to release
2. Community selects a Release Manager

Prepare for the release

As a release manager, you should create a private Slack channel, named `hudi-<version>_release_work` (e.g. hudi-0_12_0_release_work) in Apache Hudi Slack for coordination. Invite all committers to the channel.

Before your first release, you should perform one-time configuration steps. This will set up your security keys for signing the release and access to various release repositories.

To prepare for each release, you should audit the project status in the JIRA issue tracker, and do the necessary bookkeeping. Finally, you should create a release branch from which individual release candidates will be built.

NOTE: If you are using GitHub two-factor authentication and haven't configure HTTPS access, please follow the guide to configure command line access.

One-time Setup Instructions

You need to have a GPG key to sign the release artifacts. Please be aware of the ASF-wide release signing guidelines. If you don't have a GPG key associated with your Apache account, please follow the section below.

For Linux users

There are 2 ways to configure your GPG key for release, either using release automation script(which is recommended), or running all commands manually. If using Mac, please see below to handle known issues.

Use preparation_before_release.sh to setup GPG

- Script: preparation_before_release.sh
- `Usage  ./hudi/scripts/release/preparation_before_release.sh`

- Tasks included
    1. Help you create a new GPG key if you want.
    2. Configure git user.signingkey with chosen pubkey.
    3. Add chosen pubkey into dev KEYS and release KEYS
       NOTES: Only PMC can write into release repo.
    4. Start GPG agents.

Run all commands manually

- Get more entropy for creating a GPG key
    - `sudo apt-get install rng-tools`

    - `sudo rngd -r /dev/urandom`

- Create a GPG key
    - `gpg --full-generate-key`

- Determine your Apache GPG Key and Key ID, as follows:
    - gpg --list-keys
- This will list your GPG keys. One of these should reflect your Apache account, for example:
    - ---
    - pub   2048R/935D191 2019-08-29
    - uid   Anonymous Anonymous <anonymous@apache.org>
    - sub   2048R/CD4C59FD 2019-08-29

Here, the key ID is the 8-digit hex string in the pub line: 845E6689 or more than 8-digit hex string like 623E08E06DB376684FB9599A3F5953147903948A. Now, add your Apache GPG key to the Hudi's KEYS file both in dev and release repositories at dist.apache.org. Follow the instructions listed at the top of these files. (Note: Only PMC members have write access to the release repository. If you end up getting 403 errors ask on the mailing list for assistance.)

- Configure git to use this key when signing code by giving it your key ID, as follows:
    - `git config --global user.signingkey CD4C59FD, or git config --global user.signingkey 623E08E06DB376684FB9599A3F5953147903948A`
    - You may drop the --global option if you'd prefer to use this key for the current repository only.
- Start GPG agent in order to unlock your GPG key
    - `eval $(gpg-agent --daemon --no-grab --write-env-file $HOME/.gpg-agent-info)`
    - `export GPG_TTY=$(tty)`
    - `export GPG_AGENT_INFO`

For Mac users

- apt-get is not available . So install gpg using https://gpgtools.org/
- Create gpg key with your apache emailId and publish to key server (refer to the section "Submit your GPG public key into MIT PGP Public Key Server" below)
- The KEYS file is in https://dist.apache.org/repos/dist/
    - To checkout you need subversion. If subversion is not available in Mac you might have to first install it using `brew install subversion`.

Access to Apache Nexus repository

Configure access to the Apache Nexus repository, which enables final deployment of releases to the Maven Central Repository.

1. You log in with your Apache account.
2. Confirm you have appropriate access by finding org.apache.hudi under Staging Profiles.
3. Navigate to your Profile (top right dropdown menu of the page).
4. Choose User Token from the dropdown, then click Access User Token. Copy a snippet of the Maven XML configuration block.
5. Insert this snippet twice into your global Maven settings.xml file, typically ${HOME}/.m2/settings.xml. The end result should look like this, where T OKEN_NAME and TOKEN_PASSWORD are your secret tokens:

```
<settings>
   <servers>
     <server>
       <id>apache.releases.https</id>
       <username>TOKEN_NAME</username>
       <password>TOKEN_PASSWORD</password>
     </server>
     <server>
       <id>apache.snapshots.https</id>
       <username>TOKEN_NAME</username>
       <password>TOKEN_PASSWORD</password>
     </server>
   </servers>
 </settings>
```

Submit your GPG public key into MIT PGP Public Key Server

In order to make yourself have the right permission to stage java artifacts in Apache Nexus staging repository, please submit your GPG public key into MIT PGP Public Key Server.

Also send public key to ubuntu server via

```
gpg --keyserver hkp://keyserver.ubuntu.com --send-keys ${PUBLIC_KEY} # send public key to ubuntu server
gpg --keyserver hkp://keyserver.ubuntu.com --recv-keys ${PUBLIC_KEY} # verify
```

would also refer to stackoverflow.

Create a new version in JIRA

When contributors resolve an issue in JIRA, they are tagging it with a release that will contain their changes. With the release currently underway, new issues should be resolved against a subsequent future release. Therefore, you should create a release item for this subsequent release, as follows:

Attention: Only PMC has permission to perform this. If you are not a PMC, please ask for help in dev@ mailing list.

1. In JIRA, navigate to Hudi > Administration > Versions.
2. Add a new release. Choose the next minor version number after the version currently underway, select the release cut date (today's date) as the S tart Date, and choose Add.
3. At the end of the release, go to the same page and mark the recently released version as released. Use the ... menu and choose Release.

Triage release-blocking issues in JIRA

There could be outstanding release-blocking issues, which should be triaged before proceeding to build a release candidate. We track them by assigning a specific Fix version field even before the issue resolved.

The list of release-blocking issues is available at the version status page. Triage each unresolved issue with one of the following resolutions:

For all JIRA issues:

- If the issue has been resolved and JIRA was not updated, resolve it accordingly.

For JIRA issues with type "Bug" or labeled "flaky":

- If the issue is a known continuously failing test, it is not acceptable to defer this until the next release. Please work with the Hudi community to resolve the issue.
- If the issue is a known flaky test, make an attempt to delegate a fix. However, if the issue may take too long to fix (to the discretion of the release manager):
    - Delegate manual testing of the flaky issue to ensure no release blocking issues.
    - Update the Fix Version field to the version of the next release. Please consider discussing this with stakeholders and the dev@ mailing list, as appropriate.

For all other JIRA issues:

- If the issue has not been resolved and it is acceptable to defer this until the next release, update the Fix Version field to the new version you just created. Please consider discussing this with stakeholders and the dev@ mailing list, as appropriate.
- If the issue has not been resolved and it is not acceptable to release until it is fixed, the release cannot proceed. Instead, work with the Hudi community to resolve the issue.

If there is a bug found in the RC creation process/tools, those issues should be considered high priority and fixed in 7 days.

Review Release Notes in JIRA

JIRA automatically generates Release Notes based on the Fix Version field applied to issues. Release Notes are intended for Hudi users (not Hudi committers/contributors). You should ensure that Release Notes are informative and useful.

Open the release notes from the version status page by choosing the release underway and clicking Release Notes.

You should verify that the issues listed automatically by JIRA are appropriate to appear in the Release Notes. Specifically, issues should:

- Be appropriately classified as Bug, New Feature, Improvement, etc.
- Represent noteworthy user-facing changes, such as new functionality, backward-incompatible API changes, or performance improvements.
- Have occurred since the previous release; an issue that was introduced and fixed between releases should not appear in the Release Notes.
- Have an issue title that makes sense when read on its own.

Create a release branch in apache/hudi repository

Attention: Only committer has permission to create release branch in apache/hudi. **Skip this step if it is a bug fix release. But do set the env variables.**

Release candidates are built from a release branch. As a final step in preparation for the release, you should create the release branch, push it to the Apache code repository, and update version information on the original branch.

Export Some Environment variables in the terminal where you are running the release scripts

- `export RELEASE_VERSION=<RELEASE_VERSION_TO_BE_PUBLISHED>`
- `export NEXT_VERSION=<NEW_VERSION_IN_MASTER>`
- `export RELEASE_BRANCH=release-<RELEASE_VERSION_TO_BE_PUBLISHED>`
- `export RC_NUM=<release_candidate_num_starting_from_1>`

Use cut_release_branch.sh to cut a release branch

- Script: cut_release_branch.sh

Usage

```
# Cut a release branch
Cd scripts && ./release/cut_release_branch.sh \
--release=${RELEASE_VERSION} \
--next_release=${NEXT_VERSION} \
--rc_num=${RC_NUM}
# Show help page
./hudi/scripts/release/cut_release_branch.sh -h
```

For Bug Fix release:

Here is how to go about a bug fix release.

- Create a branch in your repo (<user>/hudi).
- cherry pick commits from master that needs to be part of this release. (git cherry-pick commit-hash). You need to manually resolve the conflicts. For eg, a file might have been moved to a diff class in master where as in your release branch, it could be in older place. You need to take a call where to place it. Similar things like file addition, file deletion, etc.
- Update the release version by running "`mvn versions:set -DnewVersion=${RELEASE}-rc`${RC_NUM}", with "RELEASE" as the version and "RC_NUM" as the RC number. Make sure the version changes are intended. Then git commit the changes.
- Ensure both compilation and tests are good.
- I assume you will have apache/hudi as upstream. If not add it as upstream
- Once the branch is ready with all commits, go ahead and push your branch to upstream.
- Go to apache/hudi repo locally and pull this branch. Here after you can work on this branch and push to origin when need be.
- Do not forget to set the env variables from above section.

Verify that a Release Build Works

Run "*mvn -Prelease clean install*" to ensure that the build processes are in good shape. // You need to execute this command once you have the release branch in apache/hudi

Good to run this for all profiles

```
mvn -Prelease clean install
mvn -Prelease clean install -Dscala-2.12
mvn -Prelease clean install -Dspark3
```

Checklist to proceed to the next step

- Release Manager's GPG key is published to dist.apache.org
- Release Manager's GPG key is configured in git configuration
- Release Manager has org.apache.hudi listed under Staging Profiles in Nexus
- Release Manager's Nexus User Token is configured in settings.xml
- JIRA release item for the subsequent release has been created
- All test failures from branch verification have associated JIRA issues
- There are no release blocking JIRA issues
- Release branch has been created
- Release Notes have been audited and added to RELEASE_NOTES.md

Build a release candidate

The core of the release process is the build-vote-fix cycle. Each cycle produces one release candidate. The Release Manager repeats this cycle until the community approves one release candidate, which is then finalized.

Set up a few environment variables to simplify Maven commands that follow. This identifies the release candidate being built. Start with RC_NUM equal to 1 and increment it for each candidate.

a. `git checkout ${RELEASE_BRANCH}`

b. Run mvn version to set the proper rc number in all artifacts
   i. `mvn versions:set -DnewVersion=${RELEASE_VERSION}-rc${RC_NUM}`

c. Run Unit tests and ensure they succeed
   i. `mvn test -DskipITs=true`

d. Run Integration Tests and ensure they succeed
   i. `mvn verify -DskipUTs=true`

e. Commit and push this change to RELEASE branch
   i. `git commit -am "Bumping release candidate number ${RC_NUM}"`

      There will be some backup files created which needs to be removed. You could do "git clean -fd" before doing the commit.
   ii. `git push origin ${RELEASE_BRANCH}`

If you already have a remote tag with same name as your branch, you can try below command.

git push origin refs/heads/${RELEASE_BRANCH}

"refs/heads/" refers to a branch.

"refs/tags/" refers to tag.

f. Generate Source Release: This will create the tarball under hudi/src_release directory
   i. `git checkout ${RELEASE_BRANCH}`
   ii. `cd scripts && ./release/create_source_release.sh`

   If you have multiple gpg keys(gpg --list-keys), then the signing command will take in the first key most likely. You will release this when it asks for a passphrase in a pop up. When asked for passphrase, ensure the intended key is the one asked for.

   Command used in script:

   gpg --armor --detach-sig ${RELEASE_DIR}/hudi-${RELEASE_VERSION}.src.tgz

   To use a specific key: update as follows: // replace FINGERPRINT

   gpg --local-user [FINGERPRINT] --armor --detach-sig ${RELEASE_DIR}/hudi-${RELEASE_VERSION}.src.tgz
   iii. Verify Source release is signed and buildable
      1. cd hudi/src_release
      2. gpg --verify hudi-${RELEASE_VERSION}-rc${RC_NUM}.src.tgz.asc hudi-${RELEASE_VERSION}-rc${RC_NUM}.src.tgz
      3. tar -zxvf hudi-${RELEASE_VERSION}-rc${RC_NUM}.src.tgz && cd hudi-${RELEASE_VERSION}-rc${RC_NUM} && mvn clean package -DskipTests -Pintegration-tests
      4. If they pass, delete the repository we got from the tar-ball
         a. cd ../ && rm -rf hudi-${RELEASE_VERSION}-rc${RC_NUM}

g. Create tag
   i. `git tag -s release-${RELEASE_VERSION}-rc${RC_NUM} -m "${RELEASE_VERSION}"`

   If you run into some issues, and if want to re-run the same candidate again from start, ensure you delete existing tags before retrying again.

   // to remove local

   git tag -d release-${RELEASE_VERSION}-rc${RC_NUM}

   // to remove remote
   git push --delete origin release-${RELEASE_VERSION}-rc${RC_NUM}
   ii. if apache repo is origin.
      1. `git push origin release-${RELEASE_VERSION}-rc${RC_NUM}`

      If a branch with the same name already exists in origin, this command might fail as below.

      error: src refspec release-0.5.3 matches more than one

      error: failed to push some refs to 'https://github.com/apache/hudi.git'

      In such a case, try below command

      git push origin refs/tags/release-${RELEASE_VERSION}-rc${RC_NUM}

h. Stage source releases on dist.apache.org
   i. If you have not already, check out the Hudi section of the dev repository on dist.apache.org via Subversion. In a fresh directory
   ii. if you would not checkout, please try *svn checkout https://dist.apache.org/repos/dist/dev/hudi again.*
      1. `svn checkout https://dist.apache.org/repos/dist/dev/hudi --depth=immediates`

   iii. Make a directory for the new release:
      1. `mkdir hudi/hudi-${RELEASE_VERSION}-rc${RC_NUM}`

   iv. Copy Hudi source distributions, hashes, and GPG signature:
      1. `mv <hudi-dir>/src_release/* hudi/hudi-${RELEASE_VERSION}-rc${RC_NUM}`

   v. Add and commit all the files.
      1. `cd hudi`
      2. `svn add hudi-${RELEASE_VERSION}-rc${RC_NUM}`
      3. `svn commit`

   vi. Verify that files are present
   vii. Run Verification Script to ensure the source release is sane
      1. For RC: cd scripts && ./release/validate_staged_release.sh --release=${RELEASE_VERSION} --rc_num=${RC_NUM} --verbose
      2. For finalized release in dev: cd scripts && ./release/validate_staged_release.sh --release=${RELEASE_VERSION} --verbose

i. Deploy maven artifacts and verify
    i. This will deploy jar artifacts to the Apache Nexus Repository, which is the staging area for deploying jars to Maven Central.
    ii. Review all staged artifacts (https://repository.apache.org/). They should contain all relevant parts for each module, including po m.xml, jar, test jar, source, test source, javadoc, etc. Carefully review any new artifacts.
    iii. `git checkout ${RELEASE_BRANCH}`.
    iv. ./scripts/release/deploy_staging_jars.sh 2>&1 | tee -a "/tmp/${RELEASE_VERSION}-${RC_NUM}.deploy.log"
        1. when prompted for the passphrase, if you have multiple gpg keys in your keyring, make sure that you enter the right passphase corresponding to the same key (FINGERPRINT) as used while generating source release in step f.ii.
            a. If the prompt is not for the same key (by default the maven-gpg-plugin will pick up the first key in your keyring so that could be different), then add the following option to your ~/.gnupg/gpg.conf file

```
default-key <FINGERPRINT_OF_KEY_USED_FOR_SOURCE_RELEASE>
```

        2. make sure your IP is not changing while uploading, otherwise it creates a different staging repo
        3. Use a VPN if you can't prevent your IP from switching
        4. after uploading, inspect the log to make sure all maven tasks said "BUILD SUCCESS"
    v. Review all staged artifacts by logging into Apache Nexus and clicking on "Staging Repositories" link on left pane. Then find a "open" entry for apachehudi
    vi. Ensure it contains all 3 (2.11, 2.12 with spark2 and 2.12 with spark3) artifacts, mainly hudi-spark-bundle-2.11/2.12, hudi-spark3-bundle-2.12, hudi-spark-2.11/2.12, hudi-spark2-2.11/2.12, hudi-spark3-2.12, hudi-utilities-bundle_2.11/2.12 and hudi-utilities_2.11/2.12.
        1. With 0.10.1, we had 4 bundles. spark2 with scala11, spark2 with scala12, spark3.0.x bundles and spark3.1.x bundles. Ensure each spark bundle reflects the version correctly. hudi-spark3.1.2-bundle_2.12-0.10.1.jar and hudi-spark3.0.3-bundle_2.12-0.10.1.jar are the respective bundle names for spark3 bundles.
    vii. Once you have ensured everything is good and validation of step 7 succeeds, you can close the staging repo. Until you close, you can re-run deploying to staging multiple times. But once closed, it will create a new staging repo. So ensure you close this, so that the next RC (if need be) is on a new repo. So, once everything is good, close the staging repository on Apache Nexus. When prompted for a description, enter “Apache Hudi, version ${RELEASE_VERSION}, release candidate ${RC_NUM}”.
    viii. After closing, run the script to validate the staged bundles again:
        1. `./scripts/release/validate_staged_bundles.sh orgapachehudi-<stage_repo_number> ${RELEASE_VERSION}-rc${RC_NUM} 2>&1 | tee -a /tmp/validate_staged_bundles_output.txt`

Checklist to proceed to the next step

1. Maven artifacts deployed to the staging repository of repository.apache.org
2. Source distribution deployed to the dev repository of dist.apache.org and validated

Vote on the release candidate

Once you have built and individually reviewed the release candidate, please share it for the community-wide(dev@hudi) review. Please review foundation-wide voting guidelines for more information.

Start the review-and-vote thread on the dev@ mailing list. Here’s an email template; please adjust as you see fit.

From: Release Manager

To: dev@hudi.apache.org

Subject: [VOTE] Release 1.2.3, release candidate #3

Hi everyone,

Please review and vote on the release candidate #3 for the version 1.2.3, as follows:

[ ] +1, Approve the release

[ ] -1, Do not approve the release (please provide specific comments)

The complete staging area is available for your review, which includes:

* JIRA release notes [1],

* the official Apache source release and binary convenience releases to be deployed to dist.apache.org [2], which are signed with the key with fingerprint FFFFFFFF [3],

* all artifacts to be deployed to the Maven Central Repository [4],

* source code tag "1.2.3-rc3" [5],

The vote will be open for at least 72 hours. It is adopted by majority approval, with at least 3 PMC affirmative votes.

Thanks,

Release Manager

[1] link

[2] link

[3] https://dist.apache.org/repos/dist/release/hudi/KEYS

[4] link

[5] link

[6] link

If there are any issues found in the release candidate, reply on the vote thread to cancel the vote, and there's no need to wait 72 hours if any issues found. Proceed to the Fix Issues step below and address the problem. However, some issues don't require cancellation. For example, if an issue is found in the website, just correct it on the spot and the vote can continue as-is.

If there are no issues, reply on the vote thread to close the voting. Then, tally the votes in a separate email. Here's an email template; please adjust as you see fit.

From: Release Manager

To: dev@hudi.apache.org

Subject: [RESULT] [VOTE] Release 1.2.3, release candidate #3

I'm happy to announce that we have unanimously approved this release.

There are XXX approving votes, XXX of which are binding:

* approver 1

* approver 2

* approver 3

* approver 4

There are no disapproving votes.

Thanks everyone!

Please look at previous examples in previous releases. For example : Please see examples here : voting in dev , voting in general and result of voting

Checklist to proceed to the finalization step

1. Community votes to release the proposed candidate, with at least three approving PMC votes

Fix any issues

Any issues identified during the community review and vote should be fixed in this step.

Code changes should be proposed as standard pull requests to the master branch and reviewed using the normal contributing process. Then, relevant changes should be cherry-picked into the release branch. The cherry-pick commits should then be proposed as the pull requests against the release branch, again reviewed and merged using the normal contributing process.

Once all issues have been resolved, you should go back and build a new release candidate with these changes.

Checklist to proceed to the next step

1. Issues identified during vote have been resolved, with fixes committed to the release branch.

Finalize the release

Once the release candidate has been reviewed and approved by the community, the release should be finalized. This involves the final deployment of the release candidate to the release repositories, merging of the website changes, etc.

Current follow the below steps to finalize the release.

1. change the version from ${RELEASE_VERSION}-rc${RC_NUM} to ${RELEASE_VERSION} against release branch, use command `mvn versions:set -DnewVersion=${RELEASE_VERSION}`, e.g. change 0.5.1-rc1 to 0.5.1.
2. Commit and push the version change to release branch.
    a. git commit -am "[MINOR] Update release version to reflect published version ${RELEASE_VERSION}"
    b. git push origin release-${RELEASE_VERSION}
3. Repeat the steps from Generate Source Release(f) to Stage source releases on dist.apache.org(i). Including staging jars with the release version and uploading source release. **Note that make sure remove the -rc${RC_NUM} suffix when repeat the above steps. and please also verify the steps. Ensure git tag is also done without -rc${RC_NUM}**
4. One more step is to deploy source code to release dist. *https://dist.apache.org/repos/dist/release/hudi. Only PMC will have access to this repo. So, if you are not a PMC, do get help from somone who is.*
    a. *svn checkout https://dist.apache.org/repos/dist/release/hudi --depth=immediates,* if you would not checkout, please try *svn checkout https://dist.apache.org/repos/dist/release/hudi again.*
    b. Make a directory for the new release:
        i. mkdir hudi/${RELEASE_VERSION}
    c. Copy Hudi source distributions, hashes, and GPG signature:
        i. mv <hudi-dir>/src_release/* hudi/${RELEASE_VERSION}

    d. Add and commit all the files.
        i. cd hudi
        ii. svn add ${RELEASE_VERSION}
        iii. svn commit
    e. Verify that files are present
5. Use the Apache Nexus repository to release the staged binary artifacts to the Maven Central repository. In the `Staging Repositories` section, find the relevant release candidate `orgapachehudi-XXX` entry and click `Release`. Drop all other release candidates that are not being released. It can take up to 24 hours for the new release to show up in Maven Central repository.
6. In Jira, go to Releases <Release Version> and ensure that all Jiras for the release are 'Closed' state, if not transition all 'Resolved' jiras to 'Closed'.
7. Finalize the Release in Jira by providing the release date.
8. Update DOAP file in the root of the project via sending a PR like this one.
9. Create a new Github release, off the release version tag, you pushed before

Steps to cut doc version and update website.

1. Follow the instructions for cutting a doc for this new release.
    a. For a minor release, make sure any docs updates regarding the new features for the next major release, not in the minor release, are excluded. This involves reverting the changes or manually removing the relevant docs in a separate branch.
2. Build the site locally and ensure the new doc version is available as intended.
3. Update site using instructions
4. There are few adhoc fixes that needs to be taken care apart from above steps. Adding examples of commits from 0.10.1 that you can follow.
    a. https://github.com/apache/hudi/pull/4703/commits/1e7c9af976b075c9ed5a780da1cdb57f019a41d3
    b. https://github.com/apache/hudi/pull/4703/commits/b474ec266fe2243f8146ba7a112045bbc8b0ddc8
    c. https://github.com/apache/hudi/commit/549fa7a51b30b162dcd6fc70b42cf1779de1900b
    d. https://github.com/apache/hudi/commit/de3405855c23aeb449113de197591c186396a4c2

Promote the release

Once the release has been finalized, the last step of the process is to promote the release within the project and beyond. Please wait for 1h after finalizing the release in accordance with the ASF release policy.

Apache mailing lists

Announce on the dev@ mailing list that the release has been finished.

Announce on the release on the user@ mailing list, listing major improvements and contributions.

Announce the release on the announce@apache.org mailing list. **NOTE:** put announce@apache.org in **bcc** to avoid disrupting people with followups.

Considering that announce@ ML has restrictions on what is published, we can follow this email template:

From: Release Manager

To: announce@apache.org

Subject: [ANNOUNCE] Apache Hudi <VERSION> released

The Apache Hudi team is pleased to announce the release of Apache

Hudi <VERSION>.

Apache Hudi (pronounced Hoodie) stands for Hadoop Upserts Deletes

and Incrementals. Apache Hudi manages storage of large analytical

datasets on DFS (Cloud stores, HDFS or any Hadoop FileSystem compatible storage)

and provides the ability to query them.

This release comes xxx months after xxx. It includes more than
xxx resolved issues, comprising of a few new features as well as
general improvements and bug-fixes. It includes support for
xxx, xxx, xxx, and many more bug fixes and improvements.

For details on how to use Hudi, please look at the quick start page located at https://hudi.apache.org/docs/quick-start-guide.html

If you'd like to download the source release, you can find it here:

https://github.com/apache/hudi/releases/tag/<VERSION>

You can read more about the release (including release notes) here:

https://issues.apache.org/jira/secure/ReleaseNote.jspa?projectId=12322822&version=<JIRA_VERSION>

We welcome your help and feedback. For more information on how to
report problems, and to get involved, visit the project website at:

https://hudi.apache.org/

Thanks to everyone involved!

XXX

Recordkeeping

Use reporter.apache.org to seed the information about the release into future project reports.

Social media

Tweet, post on Facebook, LinkedIn, and other platforms. Ask other contributors to do the same.

Improve the process

It is important that we improve the release processes over time. Once you've finished the release, please take a step back and look what areas of this process and be improved. Perhaps some part of the process can be simplified. Perhaps parts of this guide can be clarified.

If we have specific ideas, please start a discussion on the dev@ mailing list and/or propose a pull request to update this guide. Thanks!