Release Process

Metron Release Types

There are two types of Metron releases:

- Feature Release - this is a release that has a significant step forward in feature capability and is denoted by an upgrade of the second digit
- Maintenance Release - this is a set of patches and fixes that are issued following the FR and is denoted by an upgrade of the third digit

Release Naming Convention

Metron build naming convention is as follows: 0.[FR].[MR]. We keep the 0. notation to signify that the project is still under active development and we will hold a community vote to go to 1.x at a future time. The first release in a Feature Release line is 0.[FR].0. There may or may not ever be a release 0.[FR].1, depending whether the community decides to create follow-on Maintenance Release(s) or just go straight to the next Feature Release (which would be named 0.[FR+1].0).

Creating a Feature Release

Step 0 - Setup Release Signing Keys

In order to ensure files are identical to the ones created by the release manager, all releases are PGP signed and signatures are provided. Read more at Apache Release Signing.

Setting up the appropriate keys can be done following the instructions at Apache GnuPG instructions. Make sure to follow the instructions on avoiding SHA-1.

Once this is done, the release manager should ensure their public key is provided in the KEYS file, following the usual Pull Request process. See METRON-1575 - Add leet gpg public key to the KEYS file and PR#1028 for an example.

Currently, the KEYS file is only pushed with a main Metron release. This means that, when there is a new release manager, they will be unable to release apache/metron-bro-kafka-plugin without a release of apache/metron.

Step 1 - Initiate a [DISCUSS] thread

Prior to the release the Release manager should do the following (preferably starting about a month before the release):

- Make sure that the list of JIRAs slated for the release accurately reflects the pull requests that are currently in master
- Construct an email to the Metron dev board (dev@metron.apache.org) which discusses with the community the desire to do a release. This email should contain the following:
  - The version number for the new release to be.
  - Proposed timeframe for the code freeze and release.
  - The list of JIRAs already committed for the release, with descriptions.

  (See "CHANGES file" below for a grep command that will generate the list of JIRAs already committed.)

- A solicitation of additional JIRAs that would be desirable to include in the next release. Users should rate them as must-have / should-have / good-to-have, as well as volunteering.

A release email template is provided here.

Step 2 - Monitor and Verify JIRAs

Once the community votes for additional JIRAs they want included in the release verify that the pull requests are in before the release. close these JIRAs and tag them in the JIRA app with the release name. All pull requests and JIRAs that were not slated for this release will go into the next releases. The release manager should continue to monitor the JIRA to ensure that the timetable is on track until the release date. On the release date the release manager should message the Metron dev board (dev@metron.apache.org) announcing the code freeze for the release.

Also, run the JIRA verification tool at dev-utilities/release-utils/validate-jira-for-release, and before the end of the release process assure that its results are made consistent with the release CHANGES document (see below).

Step 3 - Increment Metron version if needed

Determine the current build version number on master branch, as described here. If it is not the desired release number (for instance, if we desire to release version 0.5.0, but the current build version is 0.4.1), then submit a PR to update the build version number as described in Change the Build Version Number, get approval, and commit.

Step 4 - Review the Apache Documentation for Releases
Review the Apache documentation around Release Distribution and Release Policy. Make sure to understand what build artifacts are expected to produced, what are not expected to be produced (e.g. MD5 and SHA-1 used to be supplied, but are now omitted), what is expected of download links, etc. This may necessitate updates to the release process and/or script to handle.

https://www.apache.org/dev/release-distribution

https://apache.org/legal/release-policy.html

Step 5 - Create the Release Candidate

A script exists at `dev-utilities/release-utils/prepare-release-candidate` to create a release candidate and the appropriate accompanying artifacts. Follow the instructions at `dev-utilities/release-utils/README.md#prepare-release-candidate`.

In the following descriptions, `[N]` is used to denote the release candidate number. This starts at 1 and must be incremented each time a RC is pushed to Git.

It's strongly recommended to do a practice run first, validate the build ([Verifying Builds](http://www.apache.org/foundation/license-faq.html)) and licensing (see [Apache指导下](http://www.apache.org/foundation/license-faq.html)), then do a live run. A live run will do the appropriate Git and SVN pushed.

This script will:

- Do an SVN clone of the repo at `https://dist.apache.org/repos/dist/dev/metron`. We will refer to this as the dev repo. It will hold the release candidate artifacts.
- Do an SVN clone of the repo at `https://dist.apache.org/repos/dist/release/metron`. We will refer to this as the release repo. It will hold the release artifacts.
- Do a Git clone of the repo at `https://git-wip-us.apache.org/repos/asf/metron.git`. This holds the source code for the release candidate.
- Create appropriate tags (following the naming convention `git tag apache-metron_0.[FR].[MR]-rc[N]`).
- Create the release tarball (following the naming convention `apache-metron_0.[FR].[MR]-rc[N].tar.gz`).
- Create the appropriate release artifacts. The script will create these, and manual instructions are provided here:
  - Release (candidate) Tarball
    - Per [Apache guidance](http://www.apache.org/foundation/license-faq.html), we use SHA512 hash digests in the signing process, see [how to set your gpg defaults, and upgrade your signing keys if you have old ones](http://www.apache.org/foundation/license-faq.html). Likewise, do not use 4-byte fingerprints, like "DEAD BEEF" or "0xDEADBEEF", to refer to your signing key, as these have been demonstrated vulnerable. Instead use 8-byte fingerprints, like "BADD CAFE DEAD BEEF" or "0xBADDCAFEDEADBEEF".
    - SHA512 hash of the release tarball; here we will also provide SHA256 hashes for backward compatibility:
      - `gpg --print-md SHA512 apache-metron_0.[FR].[MR]-rc1.tar.gz > apache-metron_0.[FR].[MR]-rc1.tar.gz.sha512`
      - `gpg --print-md SHA256 apache-metron_0.[FR].[MR]-rc1.tar.gz > apache-metron_0.[FR].[MR]-rc1.tar.gz.sha256`
    - GPG signature of release tarball by the release manager:
      - Assuming your 8-byte public code signing key fingerprint is 0xBADDCAFEDEADBEEF, so your signing command would be: `gpg -u 0xBADDCAFEDEADBEEF --armor --output apache-metron_0.[FR].[MR]-rc1.tar.gz.asc --detach-sig apache-metron_0.[FR].[MR]-rc1.tar.gz`
      - Note: You only need the `-u` arg if you have more than one public/private key pair generated. If you have forgotten it, you can find it from the output of `gpg --fingerprint`. It's the last 8 bytes (16 hex characters) from the key fingerprint.
      - If you do not know, or cannot recover, your code signing key and password as release manager, you must start over with a new code signing key. Follow the instructions at [https://www.apache.org/dev/release-signing.html#generate](https://www.apache.org/dev/release-signing.html#generate). You should then link your new code signing key into Apache's web of trust.
    - The LICENSE and NOTICE files from the release tarball
    - The KEYS file from the release tarball
    - A CHANGES file denoting the changes
  - Create the directory in the dev repo for the artifacts (following the convention `0.[FR].[MR]-RC[N]`). A live run will also commit them (`svn commit -m "Adding artifacts for Metron 0.[FR].[MR]-RC[N]"`). A practice run will simply print the commands.

**Historical Note:** Releases for Metron versions 0.6.0 and earlier used a - to delineate the prefix (apache-metron) from the version/release information (such as 0.6.0-release or 0.6.0-rc1).

Step 6 - Push the Git Tag

When you're confident this release candidate is worthy to be tested by the community, push the tag to the public git repo. Chdir back to the local repo from which you created the release candidate tarball, and verify that the tag is present:

```
git tag
```

Then push the tag to the public repo:

```
git push origin tag apache-metron_0.[FR].[MR]-rc[N]
```

Browse the public repo and confirm the tag has been pushed successfully.

Step 7 - Call for a community release vote

Next initiate a [VOTE] thread on the dev list to announce the build vote. The vote email template can be found here: [Build Vote Template](http://www.apache.org/foundation/release-voting.html). Allow at least 72 hours for the community to vote on the release.
- If issues are found with the release and the vote fails, then the vote thread is closed with a synopsis of the voting results and a new RC is worked on in the community.
- If issues are found with the release and the vote succeeds, then we proceed to cut the release, but should notify the community of the issues via an email on the dev list with the accompanying JIRA(s) required to correct the issue(s).
- If no issues are found, then we can cut a release.

When you get enough votes, close the vote by replying `[RESULT][VOTE]` to the email thread with the tally of all the votes. Again, wait for at least 72 hours before closing the vote.

### Step 8 - Tag the finished release

**apache/metron**

After the vote passes, re-tag the RC as the release. In your working repo for the release (cloned from https://git-wip-us.apache.org/repos/asf/metron.git), do:

```bash
git tag apache-metron_0.[FR].[MR]-release apache-metron_0.[FR].[MR]-rc[N]  
git push origin tag apache-metron_0.[FR].[MR]-release
```

**apache/metron-bro-plugin-kafka**

After the vote passes, re-tag the RC as the release. In your working repo for the release (cloned from https://git-wip-us.apache.org/repos/asf/metron-bro-plugin-kafka.git), do:

```bash
git tag apache-metron-bro-plugin-kafka_0.[FR].[MR]-release apache-metron-bro-plugin-kafka_0.[FR].[MR]-rc1

git tag 0.[FR] apache-metron-bro-plugin-kafka_0.[FR].[MR]-release

git push origin tag apache-metron-bro-plugin-kafka_0.[FR].[MR]-release

git push origin tag 0.[FR]
```

The 0.[FR] tag is created to allow the release to be handled appropriately by bro-pkg. Note that we expect to be able to change this to 0.[FR].[MR] as of the release of bro 2.7 via https://github.com/bro/bro/commit/615ff78234fb6d1be889e3544eb034c0b6891a4, but it would need to be coordinated with a change in apache/metron-bro-plugin-kafka itself (https://github.com/apache/metron-bro-plugin-kafka/pull/8).

### Step 9 - Stage the finished release

A directory with the name of the release version should be made in the release svn repository.

Collateral from the release candidate in the dev repo should be moved to the release repo directory and renamed to remove the rc (e.g. mv apache-metron_0.[FR].[MR]-rc1.tar.gz apache-metron_0.[FR].[MR].tar.gz)

Regenerate the `sha*` files (but NOT the asc signature file), and compare against the original to confirm they have exactly the same hash fingerprints. Alternatively one may edit the `sha*` files to remove the “-rc1” internal references, but there is greater potential for human error with a manual edit.

Add the directory and commit via the subversion client:

```bash
svn add 0.[FR].[MR]
svn commit -m "Adding artifacts for Metron 0.[FR].[MR]"
```

Confirm that the new release is visible in dist, by pointing your browser at https://dist.apache.org/repos/dist/release/metron/. It should appear promptly, but may not be instantaneous.

### Step 10 - Update the Website to point at the new release location

Open a JIRA with title "update public web site to point at 0.[FR].[MR] new release". Suppose it is METRON-1234. Make a new git clone of Metron, and create a working branch for the website update submission:

```bash
cd <workdir>
git clone https://github.com/apache/metron.git
cd metron

git checkout -b METRON-1234 apache-metron_0.[FR].[MR]-release
```
Now rebuild the doc and web sites, and copy the built site-book to site/current-book

```
mvn clean site
cd site
rm -rf current-book
cp -r ../site-book/target/site current-book
```

To validate the current-book, do `open current-book/index.html` in your browser and confirm that it shows the correct Metron version number in the upper right.

Now edit the file `documentation/index.md` to change all instances of the old version number, in both human readable text and URIs.

Finally, use bundle and the jekyll local server to build and check the actual website, following the instructions here: Website PR Merge. Validate that the Download links are correct, and that the Book link is correct. Now submit a PR for METRON-1234, get approval, and commit the new "site" source to master, in the usual way. But don't do the final step yet (commit built site image from "target" to asf-site branch), until the release has propagated to mirrors; see next two steps.

Remember that download links in the Website should never point directly at dist, but rather always point at mirrors via http://www.apache.org/dyn/closer.cgi/metron/[VERSION]/... However, the signature files (.asc and .sha*) must point at https://www.apache.org/dist/metron/0.[FR].[MR]/...

### Step 11 - Wait for the new release to propagate to mirrors

According to Apache docs, "It may take up to 24 hours or more for a newly published release to be sync'd to all mirrors. Mirrors have their own schedules. Mirrors are required to check at least once a day, but most will check for updates 2 to 4 times per day." At least wait until a spot-check of mirrors on http://www.apache.org/dyn/closer.cgi/ show the new release.

### Step 12 - Add the release to the Apache database

Add the release to Apache so it shows up in the board report generator by going here: https://reporter.apache.org/addrelease.html?metron

### Step 13 - Announce the release

Commit the built Website to the Metron asf-site branch, per the final instructions from Website PR Merge, so it is publicly visible.

Send an email out to user@ and dev@ to announce the release along with the changelog and a word of thanks/praise.

### Step 14 - Clean up

Remove the old releases from the release repo, after making sure they are already mirrored into the archive repo. Only the current version and the KEYS file should be in the release repo.

It is good practice to increment the build version in master immediately after a Feature Release, so that dev builds with new stuff from master cannot be mistaken for builds of the release version. So, immediately after a release, increment the MINOR version number (eg, with the 0.4.0 just released, set the new version number to 0.4.1) per Change the Build Version Number.

In addition, if a new version of apache/metron-bro-plugin-kafka has been released, the development environments in apache/metron should be updated to point to the latest plugin release.

### Creating a Maintenance Release

Creation of the Maintenance Release should follow exactly the same set of steps as creating the Feature Release as outlined above, but with two exception. First, the version incremented on the maintenance release should be the MR++ so that the release is named 0.[FR].[MR++]. Second, if a critical JIRA comes in that requires an immediate patch, the votes with three binding +1's are still required, but the following can be waived:

- Step 1 (discussion)
- Step 2 (Jira collecting and tracking)
- the 72 hour waiting period in Step 9 (vote)
- Step 13 (mirror propagation)

A critical JIRA is something that is either a security vulnerability or a functional show stopper.

### Maintaining a Maintenance Branch

Being able to maintain the previous release train, with only critical or important bug fixes and security fixes (generally not new features) for users who are averse to frequent large changes is very important for production use. They get stability, while the new feature release code line proceeds as fast as the community wishes.
When needed, and with community discussion, create a Maintenance Branch for the previous Metron release by incrementing the "third" digit of the previous release like so 0.[FR]."[MR++"]. All patches to the previous Metron release will be checked in under the MR branch and where it makes sense also under the master branch. All new features will be checked in under the master branch.

Ensuring Consistency between Feature and Maintenance releases

It is important to assure that all commits to the maintenance branch also get made in the master branch (if relevant), to avoid the appearance of regressions in the next Feature Release from master branch. The formal process for assuring this is as follows:

- Every maintenance release JIRA should have a corresponding feature JIRA to make sure that the patch is applied consistently to both branches. The maintenance JIRA should be cloned and appropriate fix version for the feature release should be applied. If the fix is not relevant to the feature or maintenance branch then the submitter must explicitly state this. In general reviewers should refuse a maintenance branch patch PR unless both feature and maintenance JIRAs have been created.
- The release manager has a responsibility to review all commits to the maintenance line since last release, and make sure they were duplicated to the feature branch (unless not relevant, which must also be determined).