PNDA Release Planning

Background

PNDA releases up to PNDA 5.0 were deployed using Saltstack from a pre-built mirror of RPM, Python, NPM, Hadoop and 3rd party packages. A release built for a version of Centos, e.g. 7.5-1804, will typically break for a new Centos release, e.g. 7.6-1810. Likewise, change in the Python or NPM package repositories can also break the PNDA release. This results in a maintenance burden on project contributors, just to keep PNDA releases working. A significant proportion of discussion on the mailing lists is helping people to resolve this release fall-out.

Current Status

The PNDA contributors are currently working on a Cloud-native PNDA that uses Helm and Kubernetes to deploy and manage containerised PNDA components. This ensures that dependencies will get resolved at build time and container images will provide the necessary isolation at deployment/run time.

There are currently only a few active PNDA contributors making changes on a best-effort basis so it is not realistic to plan a date for the next PNDA release. It will be released when it is usable enough to be useful.

Future Direction

Once we have a first Cloud-native release of PNDA, it is realistic to aim for a modest release cadence such as 3 or 4 releases per year. This would allow for new features to be developed and delivered along with existing PNDA components that could remain unchanged. It would also allow for us to be responsive to security vulnerabilities and upstream changes on a per PNDA component basis, rather than for a monolithic release.

Git Workflow Proposal

PNDA code is publicly available in a set of github repositories. Commiters should follow GitFlow workflow:

https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

A summary of theoverall flow of Gitflow is:

1. A develop branch is created from master
2. A release branch is created from develop
3. Feature branches are created from develop
4. When a feature is complete it is merged into the develop branch
5. When the release branch is done it is merged into develop and master
6. If an issue in master is detected a hotfix branch is created from master
7. Once the hotfix is complete it is merged to both develop and master

Commiters have permissions to directly push changes into PNDA git repositories.
Review by another commiter is advised but not required when merging a feature into the develop branch.

Contributors can contribute new features and hotfixes following Forking workflow:
https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow

New features should be added by forking/pull-request develop branch. The contribution must be review and approved by at least one commiter to merge feature into the develop branch.

Hotfixes are normally done by commiters to fix issues, but they can also be contributed by contributors. The hotfix contribution must be review and approved by at least one commiter to merge hotfix into the master branch.

TODO: we should configure automated tests and a method to reject commits not passing the tests.

Dockerhub Repositories

We should enable automated Builds from the git repos. The container images tags should be as follow:

Git branch/tag	Docker Image tag
branch/master	latest
tag/vX.Y.Z	X.Y.Z
branch/develop	develop

VERSIONING and Release Cicle

All PNDA GIT repos must follow Semantic Versioning 2.0.0: https://semver.org/

the pnda-helm-chart repo is the main cloud-native PNDA repository that integrates the deployment of all pnda components as containers.

First version of Cloud-native PNDA is expected to be 6.0.0 in Feb 2020.

Release Schedule

New PNDA releases (minor or major) are expected to be out each 6 months.
Patches updates are released each time a hotfix is merged into master.

PNDA is also composed of multiple components running together. Cloud-PNDA sets a default version of all its components for each Cloud-PNDA release. These versions can also be customized in values.yaml by the final user, however, default versions are the only ones tested by the community, and known to work and integrate with each other.

The release cycle of each component is independent of Cloud-PNDA releases.

When a new version of cloud-PNDA is released the PATCH version of all components are updated to the last available.

If an update of the MAJOR or MINOR versions of a component want to be included in the next PNDA release, a new feature-request issue should be opened in cloud-native pnda-helm-chart repo.

Then, the process of updating a component is triggered.

Upgrading a PNDA component

Default PATCH version upgrades of PNDA components in Cloud-PNDA must be performed through hotfixes. We strongly advise commiters of each PNDA component to issue a hotfix-request for the security-fixes of their components.

MINOR version upgrades of PNDA components are automatically increased to their latest value in each Cloud-PNDA release.

MAJOR version upgrades of any of the Cloud-PNDA components can carry conflicts due to incompatible changes in APIs. MAJOR version upgrades should be done through a new feature-request issue opened in cloud-native pnda-helm-chart repo.

- If the new feature-request is accepted by the commiters, a new feature branch is created to work on the integration of the upgraded component.
- Dependencies with the rest of the components must be identified, and in case of conflict, the dependent component must be updated with and its own MAJOR version increased.
- Once all dependent components are processed, the feature branch can be merged into develop, resulting in a new version of the component and also all its dependent-components.

Upgrading an external dependency (e.g. Kafka)

Since external dependencies can follow its own versioning and releasing cycle and are out of PNDA control, the upgrade process of external dependencies is more restrictive.

PATCH version upgrades of PNDA external dependencies can be requested through hotfixes-request in case of security-fixes.

PATCH version upgrades of external dependencies are automatically increased to their latest value in each Cloud-PNDA release.

MAJOR and MINOR version upgrades of external dependencies should be requested through new feature-request issues and motivated.