Continuous Versions RFC

Background

The solidifying industry best practice, as supported by research and practice, is to modify systems by deploying small changes rapidly to production in a continuous and incremental fashion.

Implementing such a process can cause friction with a variety of different software change management approaches.

Continuous Versions is a proposal for enabling distributed projects to enjoy low lead times and continuous integration. It does so by enhancing semantic versioning with a few additional conventions.

Solution

Follow these rules to implement **Continuous Versions**:

> Never increment MAJOR or MINOR version numbers
> Never ship breaking changes for active PATCH versions
> Maintain a record of the END-OF-LIFE PATCH version
> Provide push notifications of new PATCH versions

Start off with 1.0.0. The next release would be 1.0.1, then 1.0.2, and so on. There are two options for shipping breaking changes:

- Create a new package
- Phase out the old functionality and update the END-OF-LIFE patch version

Push notifications enable package consumers to minimize lead time by deploying changes to production in a timely and automated fashion. Consumers can either set package dependencies to a range (^1.0.0), or can use tools like dependabot or renovate to update via automation.

Packaging systems are recommended to maintain a reverse lookup listing every PACKAGE dependent on a given (PACKAGE, VERSION), along with statistics on the number of installs. This can help a PACKAGE maintainer make productive decisions.

Problems

Problems with Semantic Versioning Packages

Semantic Versioning has two major problems. It encourages feature branches and breaking changes. Details on these problems are described below.

MAJOR versions are breaking changes

- Semver implies that releasing major version upgrades is as easy as incrementing a counter.
- Most successful packages don't increment the major version.
- When a package does succeed at incrementing the major version, it does so with significant effort outside the packaging system.
- It is not uncommon to create aliases to support different major versions inside of a project. I.e. PACKAGE-v1 and PACKAGE-v2

The major version of semver is a dangerous feature. If a feature is rarely used and difficult to use correctly, one should consider removing it.

MINOR version are feature branches

- Maintaining feature branches for MINOR versions increases the friction of project maintenance
- Often older MINOR branches are un-maintained.
- Aggregating enhancements into larger MINOR releases increases the potential for incompatibility
- It is easier and safer to release changes continuously to production

Lead Time Impact of Upstream and Downstream Deployments

Often a package is in one repo and is consumed by a system in another repo. It commonly takes two deployments for a change in the upstream package to be running in production in the downstream application. When both deployments require manual steps this increases lead time.

Problems with Version Control

If packages cause problems, then why not stick to version control?

Unfortunately, version control also introduces friction.

Problems with Git

Git SHAs require that all parties adopt git as a version control system. Git SHAs are not easy to remember, share, or communicate. So instead, there have been multiple attempts at federating versions across multiple git repos.

- Builds off a single git mono-repo tend to generate ever increasing CI times with slower and slower builds.
- Git subtree and submodule require manual updates to the pinned version, thus sharing the double deployment problem.
- The Android Open Source Project repo tool is a comprehensive solution, but can be unwieldy to line engineers

Problems with Mercurial / Perforce / etc

The author of this RFC lacks experience with every version control system. It is conceivable that one exists with support for federated versions within a single continuously integrated product. However these solutions still require that all parties adopt the same version control system. This creates friction between teams, and certainly is unreasonable across organizations.

Appendix

Relationship to Conventional Commits

Conventional Commits provides a way to generate semver automatically from commit messages. It solves the problem of bumping version numbers automatically. It does not discourage use of MAJOR or MINOR versions.