Spike - Superset Monorepo

Contributors	Ⓥ Ville Brofeldt Yongjie Zhao M Michael Molina Ⓔ Evan Rusackas Ⓙ Junlin Chen K Kamil Gabryjelski
Created	@October 12, 2021 3:07 PM
Created by	Ⓥ Ville Brofeldt
Last edited	@October 27, 2021 9:19 AM
Last edited by	Ⓥ Ville Brofeldt
Stakeholders	
Status	
Tags	
sign off	K Kamil Gabryjelski

Background

As is well known, the source code for the Apache Superset project resides on the Apache GitHub repo `apache/superset`. Less known, however, is that a significant portion of the core frontend logic and *all* visualization plugins reside on a separate repo called `apache-superset/superset-ui`, effectively making Superset a multi-repo project. This fragmented architecture has caused numerous issues over the years, sparking SIP-58 which aims to move Superset to a monorepo architecture. The SIP constitutes a high-level proposal for the monorepo and discussion involving people familiar with the current and proposed architecture. This is a summary of that discussion and a more detailed technical proposal for solving the current situation.

Current architecture

Currently, the `superset-frontend` package in the `apache/superset` repo depends on the following external npm packages:

1. `@superset-ui/core`: Core functionality
2. `@superset-ui/chart-controls`: Control components and associated functionality
3. `@superset-ui/plugin-chart-*`: Chart plugins using the V1 chart data API
4. `@superset-ui/legacy-(preset|plugin)-chart-*`: Legacy chart plugins using the legacy chart data endpoint.

The source code for these reside on repos belonging to the `apache-superset` GitHub organization (no official affiliation to ASF or Apache Superset). This code was spun out of the main repo during 2018/2019 (See SIP-4), and has been developed mainly by Superset committers.

The first three packages are of high quality compared to the frontend code on the main repo, as they are written in 100% TypeScript (they were fully rewritten during the multi-repo migration), with both `@superset-ui/core` and `@superset-ui/chart-`

`controls` (hereafter referred to as “the core packages”) exhibiting 100% test coverage. The V1 chart plugins are also 100% TypeScript and have decent code coverage, but have no code coverage requirements on CI.

The legacy plugins, on the other hand, were mostly moved over as-is, and are predominantly written in JavaScript with practically no unit tests. The majority of these reside on the same `superset-ui` repo as the core packages, with the exception of the deck.gl plugin, which is on a monorepo in the same GH org called `superset-ui-plugins-deckgl`. This monorepo consists of three plugins; However, only one of these is maintained and referenced by the Superset app (the deck.gl one), and the monorepo is mostly broken, with failing CI and npm deploys having to be done by hand.

Challenges with the current setup

Working in the current multirepo architecture is difficult for multiple reasons. Here are some of the most problematic aspects of the current setup.

1. Shared context between main repo and `superset-ui`

Usually, when a new feature is added to Superset, it requires changes on both the main repo and `superset-ui`. Typical cases include adding a new feature flag, new visualization features, or fixing a chart plugin. Usually, the flow goes something like this:

1. Decide on the change to be done and `npm link` all necessary `superset-ui` packages. If the change involves changes to one or multiple plugins, one is usually forced to link at least three packages.
2. Make the change in the main repo.
3. Make the change in `superset-ui`.
4. Open PRs on both repos with a TODO to bump the `superset-ui` version once the PR is merged and deployed to npm.
5. Get the `superset-ui` PR merged and perform an npm deploy.
6. Wait for the npm deploy to finish and bump the package versions on the PR in the main repo.

Sometimes CI won’t pass on the main repo, as something was overlooked in the `superset-ui` PR and wasn’t caught until Cypress tests caught the problem on the “bump PR”. In this case steps 3-6 have to be repeated before proceeding.

With over 100 npm releases in the last year alone (!), this additional step causes substantial overhead to the development process.

It is also worth noting that the dual repository structure has led to some duplicated code. This is easier to avoid when the code is centralized in one repo.

2. Instability

Over the years npm linking has suddenly stopped working a dozen or so times, usually due to issues related to shared/conflicting dependencies, the Webpack config, or simply due to unknown errors during linking.

One reason for the instability is most likely due to the massive amount of shared dependencies used in the project, which causes unexpected behavior and requires careful fine-tuning to related configuration files when linking packages together. Fixing these is often time-consuming and blocks all development of `superset-ui`. We are currently in a similar situation making linking `@superset-ui/core` very unstable, which is most likely at least caused by the Webpack v5 and npm v7 upgrades. Research has also revealed that many other projects are struggling with similar problems, indicating that this issue is not isolated to Superset, but a general problem affecting major multi-repo projects.

While there are several workarounds that make it possible to temporarily fix these problems, this is both a source of engineering waste and a hindrance to developer inclusivity, as only a handful of developers are able to consistently make their dev environments work correctly.

3. Cherry picking

Every time a PR is merged on `superset-ui`, a new npm release has to be made to pull it into the main repo. When a new release is published, it bumps the version number of all affected monorepo packages. If, for instance, the core packages are changed, ALL plugins are bumped due to their dependencies having been updated.

Bringing this change into the main repo is mostly a simple chore involving updating the package and lock files. However, this workflow makes it almost impossible to cherry-pick critical fixes from `superset-ui` after a cut has been made, since the current architecture doesn’t support mixing different versions of `@superset-ui/core` (all plugins pin exact versions of the core packages). Therefore, cherry-picking PRs from `superset-ui` would require pulling in ALL changes that have been made on `superset-ui` since the cut, which usually isn’t an option due to fixes and features being merged all the time. The same applies to reverting PRs, which is equally difficult in the current setting.

4. Apache compliance

“The Apache way” mandates that all Apache projects should follow the same process for releases (72 hour vote etc). While the `apache-superset` GitHub organization isn’t strictly speaking an Apache project, in practice it’s a core component of the Apache Superset project. So far nobody from the ASF has raised this issue, but the prudent course of action would be to move all core functionality from `superset-ui` into the main repo to avoid potential legal conflicts with ASF policies. Moving the code to the main repo shouldn’t be an issue, as the `superset-ui` repos are all Apache 2.0 licensed. However, as part of the migration, ASF license compliance of dependencies on `superset-ui` should be verified using a tool such as `license-checker`.

Benefits of monorepo architecture

By switching to a monorepo architecture, the issues described above would be solved in the following ways:

- Changes that previously required multiple PRs could be contained in a single PR.
- By centralizing all dependencies in one monorepo, we would avoid issues caused by shared/peer dependencies.
- Cherry-picking PRs involving changes to `superset-ui` into release branches would be possible.
- The developer experience would be much simpler, and would totally remove the need to do npm linking.
- By removing the need for intermediate npm releases of `superset-ui` between Apache releases, the formal ASF release process requirements would be satisfied. This would also add clarity to which changes are breaking/features/fixes, as the ASF release version number follows semantic versioning much more strictly than `superset-ui` currently does.

The migration would require practically no code changes, as the core libraries would be aliased in the Webpack config to point to their respective source directories. Also, as we would no longer need to do interim npm releases, as we could synchronize the `superset-ui` releases with the official Apache releases. For example, when a release for 1.4.0 would go out, we would also publish all `@superset-ui` packages to npm with the same version. In addition to making the release process more ASF compliant, it would also be more transparent for plugin developers, as they could pin the versions to their internally used version of Superset. It would also be possible to deploy a canary release when PRs are merged, which would make it possible to do plugin development that matches the current master branch.

One potential issue that was brought up by moving to a monorepo is the added burden of moving `superset-ui` CI tasks to the main repo (CI on the main repo is often congested). However, bumping the package versions on the main repo requires a separate PR, which in itself requires running CI on the main repo. The net effect of having a single PR being opened on the main repo that takes slightly longer vs having two separate PRs running serially for the same functional changes will likely result in less time wasted waiting for CI and thus increased code velocity.

Requirements

Switching to a monorepo architecture only requires adding `lerna` as a new dependency on the main repo (MIT license). A large portion of the monorepo configurations from `superset-ui` could probably be reused.

Since `superset-frontend` is using npm, and `superset-ui` is using yarn, we should also consider migrating to a single toolchain to simplify the build process. It should now be easier to migrate the monorepo to npm, as npm@7 added support for workspaces. This change would be in line with SIP-14 which replaced yarn with npm on the main repo. However, migrating a major tool in the toolchain is always risky, so chances are we may run into surprises. In that case it might be worth considering migrating `superset-frontend` to yarn, too.

Since two repos are on different major versions of storybook (`superset-ui` is on version 5.3.18; the main repo is on 6.3.8), we need to upgrade storybook to the same version as on the main repo before migration to resolve any compatibility conflicts that may arise during migartion.

It was also pointed out in the SIP that we don't have admin access to the main repo, which means that any new CI workflows or npm release automation tasks would need to be coordinated with Apache Infra. These could, however, be done in parallel with the migration work, so wouldn't necessarily be blocking tasks for completing the code migration.

Options

The following three options have been discussed:

Option 1: No changes

In order to maintain the status quo, we would need to fix the current issues blocking npm linking. While probably doable with a few days of work, we would be stuck with the current challenges described above.

Option 2: Partial/dual monorepo

Another option would be to only move the core packages to the main repo, leaving all plugins in the `superset-ui` repo, which would remove the need for npm linking the core packages. This would save some time by limiting the scope of the migration. However, since the core packages are direct dependencies of all viz plugins, it would create a circular dependency, as `superset-frontend` would simultaneously be referencing the new and the old core packages via its own and the plugin dependencies. And even if this were made to work somehow, it would still leave the majority of the original issues unaddressed:

- Need for multiple PRs when making changes to chart plugins
- Difficulty cherry-picking PRs involving changes to charts
- The compliance issues would become worse, as we would need to make interim npm releases from the main repo for the core packages in between Apache releases

Option 3: Full monorepo

The final option is to make a full monorepo migration, as outlined in the original SIP. This would create four new directories under `superset-frontend/src/superset-ui` as follows:

`/superset-frontend/src/superset-ui/core` (alias: `@superset-ui/core`)

`/superset-frontend/src/superset-ui/chart-controls` (alias: `@superset-ui/chart-controls`)

`/superset-frontend/src/superset-ui/chart-generator` (Yeoman generator for plugins)

`/superset-frontend/src/superset-ui/plugins` (all plugins that ship with Superset)

The motivation for keeping the `superset-ui` packages/plugins under a subdirectory under `src` is to make it clear that they relate to the `@superset-ui` npm organization, and to make the initial migration as simple as possible with minimal changes to the original monorepo configuration. This structure can later be refined as needed in a follow-up PR to minimize risk for regressions in the initial PR.

We might want to consider leaving out the legacy/deprecated charts, but this would introduce a similar catch-22 as outlined in Option 2. So omitting the old charts would most likely require removing them as a dependency on master branch.

Doing a full monorepo migration will be more work than the other options and carries with it some risk in the form of potential regressions and unexpected issues that may surface during the migration. However, this work can be considered an investment into the future productivity and stability of the product, and is expected to yield a net positive effect on developer productivity fairly quickly (the required work should be offset in 3-6 months by increased development velocity), along with a better developer experience.

Recommendation

A summary of the pros/cons of the available options:

Legend: ❌ Unaddressed ☑️ Partially resolved ✔️ Fully resolved

Aa Issue	1 - No changes	2 - Partial monorepo	3 - Full monorepo
1. Shared/duplicated context	❌	☑️	✔️
2. Instability	❌	☑️	✔️
3. Cherry picking	❌	☑️	✔️
4. Apache compliance	❌	❌	✔️

Issue	1 - No changes	2 - Partial monorepo	3 - Full monorepo
Summary	Minimal effort to maintain status quo, but doesn't solve the observed problems associated with the multirepo architecture.	Partially solves the issues for the core libraries, but doesn't address any issues affecting the plugins or Apache compliance (in fact they may become worse).	Solves all observed problems, but is more resource intensive to carry out and introduces the risk of regressions in the short term.

Due to the problems identified with options 1 and 2, the recommendation is to go with the full monorepo approach described in option 3.

Since doing a full migration at once could block `superset-ui` development for a long time, the work should be done incrementally, if possible.

Phase 1 - Migrate core packages to main repo

The core packages should be migrated first, as they are the lowest level dependency, being required by `superset-frontend`, all plugins and the storybook. This would make it possible to make changes to the core packages that only affect `superset-frontend`, but would exclude changes that require updating the plugins. The lerna workflow would be integrated with the regular Apache release process, meaning that the version number would match the version of the Superset release.

In addition to migrating the packages to the main repo, the main CI workflows and storybooks for the core packages would be migrated, along with creating a workflow for deploying the canary release. We should also make sure we retain the current code coverage requirements for the core libraries.

Phase 2 - Migrate plugins to main repo

In the second phase, the default Superset plugins should be migrated to the monorepo. All plugins that ship with vanilla Superset should be included to make it easy to make changes to plugins that can be cherry-picked to releases. After this CI and storybooks would be migrated, similar to the core packages.

Phase 3 - Final cleanup

As a final phase, the old `superset-ui` repo would be archived. In addition, there is a long tail of docs that need to be updated, including:

- `CONTRBUTING.md` document on `apache/superset`
- The Hello World blog post
- Any affected Preset documentation
- Do further cleanup + harmonization work on the Storybook on the main repo, potentially introducing Vercel or similar workflow that's currently used on `superset-ui`.