RFC 677 REVIEW: Unifying the code insights experience data generation experience

Recommender: Joel Kwartler
Date: Apr 27, 2022
Status: WIP
Decider: @project lead: Chris Warwick
Input providers: Leo Papaloizos, Coury Clark, Chris Warwick, Vova Kulikov, Cristina Birkel, Justin Boyson, Alicja Suska, Felix Becker
Approvers (please review by EOD YYYY-MM-DD): Alicja Suska, Joel Kwartler
Approvals:
Team: Code Insights

Summary

Search-based Code Insights run either over an explicit list of repos or all repos. If the former, the data is generated by frontend just-in-time calls to the API. If the latter, the data is generated on a delay in the backend. These methods have different costs and benefits. We need to unify them to unblock developer velocity so we don't have to build two versions of new features, and also to get the benefits of each across all insights while removing some of these costs.

Background

When we built code Insights we started with just-in-time, frontend-only insights. We then built a backend to handle the scale of all repos and greater numbers of datapoints. These methods have different costs and benefits.

Problem

The differences in these data engines create the following problems:

1) Hard to build new features for all users (filters only exist on backend-persisted insights, even though a user won't understand why and there's not logical reason – just a technical one – why you can't filter insights over 10 explicit repos down to just 1-2 repos)
2) Hard to debug customer issues (have to split which data engine it is)

3) Users have a slow loading experience (insights that aren't persisted take longer to load on large dashboards) – this includes CEs, who have articulated this as a pain point for demos

Specifically, we want to solve the following problems (in priority order):

1. Unlock live preview for all insights, even those running over all repositories
2. Cache/load insights that have run before faster, so the overall loading time of a dashboard page before you see value is lower
3. You can see and use the filter panel on all insights
4. You can continue to collect future + past data on all insights (rather than re-running 7 points in time from today; run the original number of points and then continue collecting snapshots)

Proposal

This is the **product-level** section of this proposal. The goal is to get to a set of **Decisions for Engineering** that engineers can then lead.

When we complete this initiative, our goals are the following. "All insights" refers to "all code insights that are search or compute based" (so all current types).

It's okay if we need to iteratively roll out to first search-based and then compute-based, or vice versa, but ultimately we want to finish updating all types by the end of this work and during Q2.

I was intentional in the below point to use slightly awkward phrasing to avoid implying or anchoring us to specific technical solutions. These are problems we are trying to solve, and perhaps there are other solutions beyond "Make it do what the backend/frontend insight type does" we'll want to explore, so I avoided phrasing things like that.

WIP **Product Decisions for Engineering** (Goals we want to achieve):

1. **All insights data loads equally as fast as the fastest insights' data loads now.**
    a. This is a non-engineering-prescriptive way of saying, for example, that datapoints are cached after creation such that loading dashboards loads the data from a cache rather than hitting our API, because backend-cached insights data is the fasted to load currently.
    b. But maybe that's not what we want to do to speed this up, and I don't care about the technical choices as long as the goal – speed up loading, which is very slow for just-in-time generated insights.

2. **You can always live preview results on the creation screen for any insights.**

    a. Maybe we accomplish this by streaming? Maybe we accomplish this with some frontend API hits still while we load the persisted on the backend? I don't care, but let's make this happen.
    b. "You can live preview" ≠ you can live preview the end state of the insight. If we do this via streaming, for example, it's fine if this livestreams progressively more complete results (but still takes hours to finish)
    c. A requirement is that live preview **always runs** – you shouldn't be blocked from live previewing because there are other insights loading in the background somewhere.
    d. Stretch goal: Ideally, you don't then "lose" your live preview:
        i. If you see live preview data and then create the insight and get taken to the dashboard page with it, it's not ideal if the insight "starts over from nothing".
        ii. It's okay if this is a stretch goal – from my POV this seems technically hard

3. **"Small insights" (by repo count) load faster than "large insights"**
    a. Right now, all persisted insights load in parallel and slow the total load time of any down. If you create an insight over a single repo, it would be a failure of this effort if we end up creating a system that loads insights we used to load live over the course of hours.

4. **You can use our filter panels on all insights**.
    a. This feels like something we'll get for "free" but noting it anyway.

5. **All insights should show current – meaning <24 hours old – point-in-time values, and save ongoing snapshots at the existing persisted-insight snapshot frequency (whatever the user set as their interval). They should also all show the same number (ideally/default assumption: 12) of backfilled datapoints.**
    a. A fancy way of capturing the behavior we/users like about backend insights

6. **We no longer have to make tradeoffs in the code path/feature engineering/data results between insights over scoped lists of repos and insights over all repos *for the below features***
    a. (This is sort of the definition of unifying, but just to be clear)
    b. We don't have to consider "over some repos" and "over all repos" as different states in building new features like, for example:
        i. New filter types (when applicable to all insights)
        ii. Per-series repo lists (running a series or filtering a series individually over repos, not just the whole chart) – for search insights
        iii. New chart types – for all insight types, such as bar chart, heatmap chart, and calculating a diff rather than cumulative
        iv. New integration points with monitoring/notebooks/batch changes – for all insight types, specifically:

1. Embedding insights into notebooks
2. Setting up a monitor trigger to fire when an insight reaches a certain threshold, or on a specific time cadence
3. Setting up a monitor action to send relevant content (screenshot of insight still TBD) in the action for that trigger
4. Ability to go to a relevant batch change template – for all applicable insight types (capture group and search based – this seems easy now, just trying to list everything. It just needs to preserve/convey the search query)

v. Backfilling performance pings (for all insight types)

vi. Drilldown features / panels (early work in RFC 561 WIP: Code Insights - Improve Drilldown Capability)

Some future possible work/ideas that might be worth keeping in mind so we don't engineer ourselves into a corner:

1. Allowing per-series repo/context filters, or allowing different series on the graph to run over different sets of repos
2. Allowing you re-run an insight if the data seems incorrect, if we have a bug, or if you connect other repos.

Some things that are explicitly not requirements:

1. We don't need to run all insights over all repos – we still want to let users tell us if they care about only a specific group of repositories.

Update 2022-05-24 Design problem/decision:

When we unify insights, this means that insights running over some repos will:

1. Still show a live preview of 7 datapoints
2. But, after creation, have 12 datapoints (the 7 originally shown, right now, and 5 more further back in time as well)

Given customers are used to setting their granularity so the live preview total time covered matches what they need, should we make any design/UI communication updates to communicate that #2 is actually what will happen?

1. If we don't do anything: you want 1 year of data, you probably set granularity = 2 months, you live preview 1 year, but then you backfill 1.4 years / see 1.4 years whenever you view the insight
2. If we do communicate this: you want 1 year of data, perhaps you'd rather have 1 year via 1 month granularity, even if the live preview only shows 6 months

How can/should we update our live preview or UI communication to account for this?

Joel Kwartler is *against* adding 5 more backfilled points to live preview as this will double the time live preview takes to load. One possible option is to double the spacing of the live preview (so it collects total time range) but we'd have to communicate that when you save the insight, it'll fill in with more datapoints. Another option is to do nothing, and bonus, the user will get more backfill data than they were expecting.

Joel Kwartler my preference: I think the best option is a clean and simple design/communication solution in the UI that tells the user that information most effectively, but the second best option is potentially to do nothing.

Definition of success

This work is successful if we achieve the above-listed goals. Qualitative measures that these goals drove business impact are:

1. Customers no longer run into issues or questions between persisted-insight and just-in-time insight functionality differences
2. CEs comment/no longer worry about preloading dashboards in the background for just-in-time insights
3. Code insights feel "faster" (CE/product/user feedback) because you can always get live previews.

Update 2022-05-25 Live Preview For All Repos Options:

- Run the live preview over a small subset of repos. Initially I think we would need to ask the user to which repo(s) to use for the preview because if there are many we may not have the knowledge necessary to programmatically pick repos that are relevant to the query.
    - Benefits
        - Lowest engineering effort because it fits the same model of preview we currently use.
        - We already know the performance characteristics and likelihood of returning results to the user quickly. Size of the selected repo(s) would be the factor that would most impact speed.
    - Concerns
        - Initially additional input from the user is required

        - Unknown if a subset of repos will provide the user with enough detail to commit to creation of the insight.
        - Unknown if the user would understand that the preview is run only on a subset of repos.
        - Unknown if we could programmatically select relevant repos on the users behalf.
- Run live preview over all repos, and limit the number of data points lower than the current 7
    - Benefits
        - Most accurate representation of the insights data for the points returned
    - Concerns
        - Too resource intensive and low probability of returning meaningful results to the user quickly. Outside of the installs with only very few repos the number of queries needed to generate the data would quickly grow beyond the number that could be run.
- Run live preview over all repos, and stream results back to the UI
    - Benefits
        - Fastest time to putting initial data on the screen
    - Concerns
        - Too resource intensive, outside of the installs with only very few repos the number of queries needed to generate the data would quickly grow beyond the number that could be run.
        - At any given time before 100% completion it won't be clear what the data on the chart represents. Is this data point complete? Which repos have been counted so far?
        - High contention on the sourcegraph instance. This live preview would essentially block all other code insights.
        - High waste. This would take a lot of work to calculate, and the nature of live previews is ephemeral. We would likely be throwing a lot of these out.