RFC 122 APPROVED: Code intel LSIF delivery

Editor: eric@sourcegraph.com, michael@sourcegraph.com
Status: APPROVED
Required reviewers: Nick, Christina, Quinn
Approvals: Nick, Quinn, Christina

Problem

For many users, the benefits of precise code intelligence are not immediately obvious. For a developer browsing unfamiliar code, results from basic (search-based) code intelligence are often sufficient to handle their immediate need. The limitations of search-based intelligence do not show up until they attempt more complex tasks. Precise code intelligence requires an indexing step, however, and it takes non-trivial effort to set up the LSIF indexers that provide the data for those queries—including changes to the build process. The combination of these two factors impedes adoption, even by customers who are willing to try the features.

For that reason, we have so far not seen much uptake of LSIF indexing. As a result, we lack good signals for what aspects of the product to work on. Because precise code intelligence requires *non-zero* configuration, in comparison to zero-configuration for search-based results, we have to overcome a basic bootstrapping problem: We need to reduce that friction to the point where customers can easily try and experience the benefits of precise code intelligence. We can then use their feedback to inform what improvements to make.

Before we ask a lot of effort from a Sourcegraph evangelist, we want to ensure that the process of generating an LSIF index and uploading it to a Sourcegraph instance is easy enough, and produces results of sufficient value, to justify the extra effort.

Proposal

To bootstrap adoption of precise code intelligence, we propose to index a selected corpus of “interesting” open-source repositories for Go, TypeScript, and JavaScript, and expose the data from those indexes on the public-facing sourcegraph.com instance. In this proposal, we have divided the work into three milestones.

After achieving **Milestone 3**, defined below, we want to gain enough confidence in the process that the following three delivery plans can be resumed (bringing precise code intelligence for Go, TypeScript/JavaScript, and Scala to existing and prospective customers).

- https://github.com/sourcegraph/customer/issues/7
- https://github.com/sourcegraph/customer/issues/9
- https://github.com/sourcegraph/customer/issues/10

Milestone 1: Manual indexing

We are able to generate and upload LSIF indexes for **all of** the repositories in the lists below. The process does not need to be automated, but must be reproducible with manual instructions, and the output needs to be generated and uploaded successfully, with the correct and expected results given in the UI via code intel extensions.

Milestone 2: Automated indexing

We are able to generate LSIF indexes for at least 75% of the same repositories, using the same CI that they use (given that it's sufficiently replicable). This may require temporarily forking repositories to update a build environment to add an indexing step. We should aim for a large spread of CI systems (Travis, GitHub actions, CircleCI, Buildkite, etc). This may require making additional accounts or compute resources on GCP.

Forks should be created in a new *sourcegraph-codeintel-showcase* organization. To keep forks up-to-date, we should use the GitHub action https://github.com/wei/pull.

Milestone 3: Re-integration

- We will reach out to the maintainers of the open-source repositories we have indexed, show them the results of indexing, and ask them if they are willing to consider running our indexing steps as part of their CI. For the projects that are willing, we will make pull requests to the same repositories to set up the indexing steps we created in the previous milestone, so that we can keep the indexes up-to-date without maintaining a fork.

  An important consideration in this step is to be clear that our goal is to show off the capabilities and sustainability of precise code intelligence, rather than a sales pitch.

- We generate detailed instructions to reproduce the results from M1 and M2, so that a customer developer could reproduce our results with their own instance using a workflow that is similar enough to the idioms used for building and deploying their code "normally" not to be disruptive. The level of detail should be sufficient that:

  - There is very little room for error if someone follows the instructions carefully, and
  - We can use the instructions as-written as a script for one or more screencasts to demonstrate the results.

Definition of success

We will use the following concrete outcomes to measure successful completion of this work:

1. Up-to-date precise code intelligence results on sourcegraph.com for all or most of the repositories in our target list (see below).
2. Detailed instructions for how to reproduce the results from sourcegraph.com in a customer's own instance.
3. An interesting subset of the PRs from Milestone 3 are merged, and/or we are able to run updates for those repositories with our own resources.

At this time, we will not gate completion of this work on producing screencasts, since the more immediate goal is to bootstrap precise code indexing for the customers whose delivery plans are already pending. We believe screencasts may help with subsequent customer contacts, however, and the "detailed instructions" mentioned above should be complete and accurate enough that we can continue in that direction later.

The following repos are owned by Eric, Michael, and Garo.

Target Go repositories

Project	**Gen cmd**	**Gen time**	**Output size (raw)**	**Notes**	**DotCom**	**CI**	**Forked CI**
aws/aws-sdk-go	lsif-go	2.5m	1314 MiB	640 packages on 1928 Go source files, 1.18 M LoC		Travis	Travis; DotCom
etcd-io/etcd	lsif-go	29s	149MiB		upload	Travis and Semaphore	Travis; GH Actions; DotCom.
gohugoio/hugo	lsif-go	32s			upload	Travis	Travis;

						(CircleCI for releases only)	DotCom
golang/go				Unusual project structure		(custom)	GH Actions; DotCom. N.B.: Upload processing works; results not linking correctly in UI.
grafana/grafana	lsif-go	1m16s			upload	Circle	Go working, TS wip
helm/helm	lsif-go	22s			upload	Circle	Working
kubernetes/kubernetes	lsif-go	2m	750MiB	Very large repository. Vendor and staging omitted.		(custom)	GH Actions; DotCom
moby/moby	lsif-go	55s		No go.mod file	upload	None?	Actions; DotCom
prometheus/prometheus	lsif-go	24s			upload	Circle	Circle; DotCom

Target TypeScript repositories

Project	Generation cmd	Generation time	Notes	DotCom	CI	Forked CI
ReactiveX/RxJS	npm i && lsif-tsc -p .	33s		upload	Circle	Circle; DotCo

						m
ReactiveX/IxJS	npm i && lsif-tsc -p .	7s		upload	Appveyor (failing)	using GH actions
grafana/grafana	yarn && lsif-tsc -p .	55s	nvm install/ run 12	upload	Circle	
elastic/kibana	yarn kbn bootstrap && lsif-tsc -p .	1m32s	nvm install/ run 10	upload	Jenkins (private), uses GH Actions for other things	Using GH actions
sindresorhus/got	npm i && lsif-tsc -p .	8s		upload	Travis	Travis; DotCom
angular/angular					Circle	Circle; DotCom
Microsoft/TypeScript			Unusual project structure		Travis	Travis; DotCom
facebook/jest	yarn && lsif-tsc -p .	13s		upload	Circle	Circle

Target JavaScript repositories

Project	Generation cmd	Generation time	Notes	DotCom	CI	Forked CI
expressjs/express	npm i && lsif-tsc lib/**/*.js test/**/*.js benchmarks/**/*.js --AllowJs	5s		upload	Travis	Travis

	--checkJs					
facebook/react	yarn && lsif-tsc packages/**/*.js --AllowJs --checkJs	54s		upload	CircleCI	Circle
lodash/lodash	npm i && lsif-tsc *.js test/*.js --AllowJs --checkJs	9s		upload	None?	Actions; DotCom
moment/moment	npm i && lsif-tsc **/*.js --AllowJs --checkJs	48s		upload	Travis	Travis; DotCom

Scripting Tactics

List Go packages mentioned in a dump:

```
jq -r
'select(.type=="vertex")|select(.label=="packageInformation").name' < dump.lsif
```

List all Go package names mentioned by Go source files in the repository:

```
find . -type f -not -path './.git*' -name '*.go' \
  -exec grep -m1 ^package {} \; \
| cut -c9- | cut -d/ -f1 | tr -d ' ' \
| sort | uniq -c | sort -rn \
| tee packages.txt
```

Milestone 2 Notes

For CircleCI, we can create *orbs*, which are like how GitHub actions are packaged. You publish a YAML definition of the task, and can import it by name and version.

Sample definition: https://github.com/go-nacelle/website/blob/master/.circleci/docs.orb.yml

Sample use: https://github.com/go-nacelle/service/blob/master/.circleci/config.yml

Running error logs (should be addressed and issues made before success)