

RFC 497 CLOSED: Restructuring CI Experience

This RFC served as an artifact to support work across FY22Q3/Q4 but it has drifted into being a tracking document which isn't the purpose of an RFC. Therefore it's now closed.

This RFC is a part of [RFC 495 WIP: CI reframed principles and vision](#).

Also see [RFC 496 WIP: Continuous integration observability](#) for improvements targeting CI metrics and visibility.

Recommender: JH Chabran

Input providers: Robert Lin

Status: [REVIEW](#)

Non goals: This RFC does not address observability, direct reliability of testing or with what tools we use to write the scripts. (It focuses instead on the internals and easing the contribution path to this area of the codebase by making it more maintainable).

Requested reviewers:

Approvals:

Tracking issue: [#25348](#)

Problems

CI Pipeline is a blackbox


With the exception of a handful of engineers that have been historically involved in the CI process, interactions with the CI are often a source of frustration: *What is being built and why? It fails, what do I do? This step should not run here, how can I fix that? What is the CI running exactly? I'm not sure I'll be able to reproduce things locally, ...* are recurring questions.

The result is that it's really hard for engineers to fix problems, whether it's failing tests or flakes.

`/dev/ci/**` is composed of many shell scripts that have no documentation, no context and are performing actions that are either domain related or stack related. Are they entry points or individual helpers? (both). `/dev/*.sh` is composed of many shell undocumented scripts which are in similar state.

Desired Outcomes

For each outcome, we've listed the scenarios from

 RFC 495 WIP: CI reframed principles and vision that are addressed.

1. As an engineer working in any team, I can run a failing step (with the exception of infrastructure failures) on my local machine. The step name on Buildkite is enough for me to identify which command I should run locally. If I need help, it is explicit for me who I should ping (owners are explicit).
 - a. *Scenario: On my PR, checks are red and I am able to reproduce it locally, but it's coming from tests which are not related to my changes.*
 - b. *Scenario: On my PR, checks are red and I am unable to reproduce it locally. It works on my machine.*
 - c. *On my PR, checks are red and I believe that it's a flake. I retried it (one or many times) and it turned green.*
 - d. *On my PR, checks are red and I believe that it's an infrastructure flake. How can I make sure that's the case?*
2. As an engineer making changes in a domain, I know exactly where to add a new step definition, along with supporting scripts and documentation. Corollary, as an engineer debugging a failing step, I know exactly where to find the relevant code, documentation, and who is the owner.
 - a. *Scenario: In my PR, I want to add a new step to the pipeline that relates to my domain.*
 - b. *Scenario: In my PR, I want to remove checks that shouldn't be running and are slowing it down*
3. As an engineer looking at my build on Buildkite, I immediately see what caused a step to fail by skimming over the build and what actions I need to take. I can still browse the full logs to investigate if I need to.
 - a. *Scenario: On my PR, checks are red and I am able to reproduce it locally, but it's coming from tests which are not related to my changes.*
 - b. *Scenario: On my PR, checks are red and I am unable to reproduce it locally. It works on my machine.*
 - c. *On my PR, checks are red and I believe that it's a flake. I retried it (one or many times) and it turned green.*
 - d. *On my PR, checks are red and I believe that it's an infrastructure flake. How can I make sure that's the case?*

Proposal

Legend:  Implemented |  Planned |  Under consideration









We establish a set of core principles and a set of common scenarios that defines our vision for the CI. We can then work toward making those scenarios real iteratively as we go. This work is meant to be tackled as a background task.

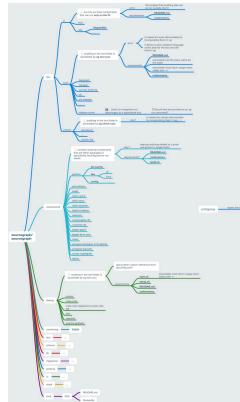
Actions


Reorganize the folder structure


Addresses: DO2

Success criteria

- ❖ Must be future-proof: if should not restrict our ability to adjust to future constraints
 - ❖ Must make sense at first glance to a new engineer
 - ❖ Must unambiguously answer the question of "where should this go?"
-
-  Add a linter for the testing sub folders (check the structure proposal below) that that checks for README and CODEOWNERS/CODENOTIFY (See also [RFC 551 APPROVED: Documenting ownership with OWNERS files](#))
 -  iterative steps to reorganize the structure if we agree on the proposed structure below. [#28287](#)
 -  Iter 1: dev/tools, dev/checks, dev/ci
 -  add linters for step1
 -  add sg wrapping for step1
 -  Iter 2: /testing
 -  Existing linter steps should call the scripts in the /check/*.sh to eliminate redundancy.
 -  Document existing folder structure on the high level to answer the question: "where should this go?" [#28288](#)



 Full picture of the proposed structure: <https://imgur.com/a/h7LlgNe>

 Please bear in mind that this is a brainstorm on the structure, meaning that it includes additional ideas regarding sg and structure that can be tackled independently in later iterations if deemed useful. We could migrate tools and testing first and leave components out for later for example.

Also, codeowners should be read as codenotiy in the diagram.

 Steps are runnable locally

Addresses: DO2

- Wrap buildkite-agent build to locally run a step
 - Discovery: [#28285](#)
 - <https://buildkite.com/docs/agent/v3/cli-bootstrap#description> indicates that this should be feasible.
- Introduce an `sg` command to output commands ran in failed steps. Something like `sg ci showFailed \$buildId`. If implementation of the previous point takes a significant amount of time having a list of commands to run locally to repro failed CI step would be a good temporary compromise [#28286](#)

? Steps are maintainable and discoverable

Addresses: DO1, DO3

- ? A pipeline step has an explicit set of owners.
- ? Every script being called by a step has an attached README and explicit owners (codenotify).
 - README must include mandatory env variables.
- ? Dependencies (release container for example are made explicit)
 - How: display the dependencies in the `sg ci preview` along with the owners as defined by the codenotify file.
- Checks can individually be called with `sg check` [#28281](#)

? Step output is actionable

Addresses: DO3

- **TODO** POC the formatted output and insert a screenshot here
- Highlight the failure reasons for that particular step in an annotation attached to the build.
- Soft failures and cancellations are highlighted in an annotation [#28283](#)
 - See <https://buildkite.com/docs/agent/v3#exit-codes>.
- ? Remove any red text from the logs unless it's an explicit failure.
- ? Display step dependencies at the beginning of the logs or in metadata
- ? Each command performing a test related action is wrapped in fold (--- or +++)

Core Principles

CI is discoverable and simple to understand

The pipeline holds an agglomeration of checks that are necessary for us to ensure we ship with an acceptable quality to our customers. As engineers, it provides us feedback on our work and it should be as clear as possible.

To make an analogy with management: nobody likes it when their manager sandwiches unactionable negative feedback with positive feedback and then bolts in another meeting, leaving you to guess what is the problem exactly. It's no different for the CI.

Discoverability implies two things:

- You can see what are the available actions while you are using the tool in question.

- You can learn about the tool without having to explicitly focus on doing so. In other terms, you learn about it as you go.

Discoverability does not absolve anyone from writing documentation, but instead complements it and exposes users to it: "they know what they don't know".

Therefore, in the context of the CI, it translates into:

- You can see what are the available tests to give you confidence in your changes and you can zoom onto these to understand how they work.
- You can read how other steps are written to understand how to add a new one or how to modify them in order to adjust what is tested.
- **The two above statements enable you to be autonomous to make changes on the CI pipeline within your own domain.**

CI steps are first class citizens and have owners

Linking a step to what scripts are run should not be a tedious task that requires squinting the eyes at a myriad of log lines. As an engineer working on a particular domain, adding or updating a step should be straightforward to me, even if I'm not familiar with the CI.

CI is predictable, *then* fast

What is being run, how and when must be very clear. Otherwise, it introduces friction that prevents new engineers from debugging existing steps, which puts pressure on the engineers who have been around longer.

Scratchpad

- **Robert Lin** I might also advocate for migrating a lot of these bash scripts to Deno, which is just as easy to set up and run and introduces all the normal programming goodness for manipulating strings like this. Alternatively, perhaps turning this lint script into an `sg` command and aggregating output there?
 - JH Chabran I did not miss your suggestion or your comment on Slack regarding this, I need to take a step back and think before opening this box :)
 - JH Chabran Also, I think this can be addressed in a further iteration, to avoid having too many changes at once.