

10,000 Lines Later:
When a Tool Became a Compiler
(and I Became a Gleamlin)

Rob Durst | February 21, 2026 | Gleam Gathering

Some Philosophy

What makes someone a Gleamlin?

What is Gleam?

A Bit About Me

[Day Job]: Site Reliability Engineer 

[Home]: Salt Lake City, Utah

[Fun Fact]: I do have a yorkie who does sometimes code...



@rob_d

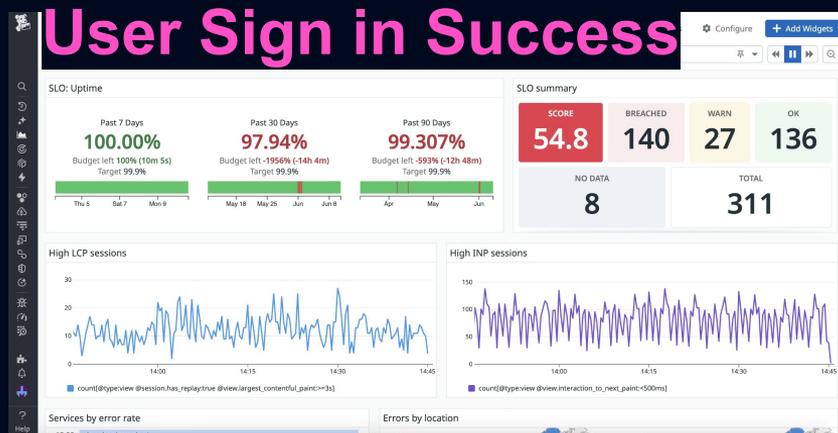


Disclaimer

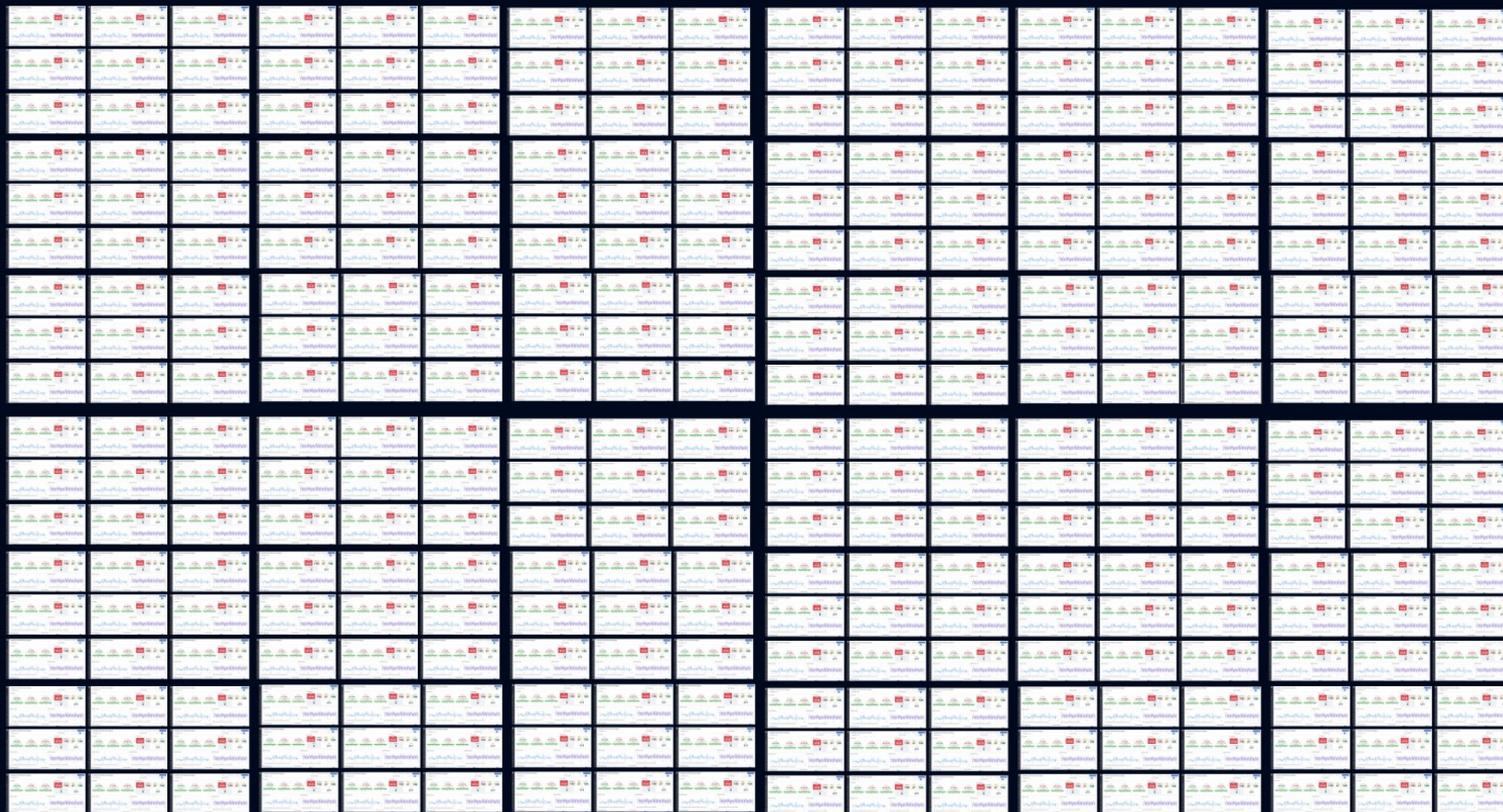
*This is a talk about the evolution of a tool.
Service Level Objectives (SLOs) are just
the language domain.*

Service Level Objectives

“99.9% of users who try to sign in with valid credentials succeed.”



ClickOps



We Built some Tooling

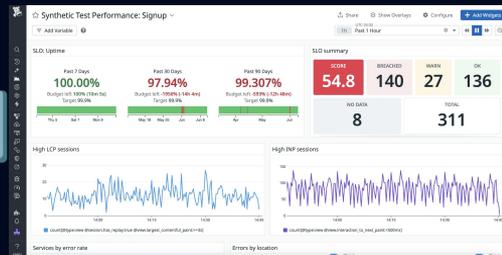
Specification

```
## Sign In
- view: "/sign_in"
  slis:
    - type: "FE_Renders"
      target: 99.9
    - type: "LCP_Latency"
      target: 99.99
      duration_threshold_seconds: 3.5
```

Compiled Artifact

```
# Monitor-Based SLO
# Create a new Datadog service level objective
resource "datadog_service_level_objective" "bar" {
  name      = "Example Monitor SLO"
  type      = "monitor"
  description = "My custom monitor SLO"
  monitor_ids = [1, 2, 3]
```

Manifestation



So We Accidentally Built a Compiler...

- [user] Simplify user interface
- [dev] Reduce maintenance burden
- [compiler] Enable tooling to create a pit of success



Let's Do it Right: *Introducing Caffeine*

“A compiler for generating reliability artifacts from service expectation definitions.”



Gleam?



#49 Self-Education in PL

Ryan Brewer

📅 Mar 14th 2025 | ⌚ 144 min



One Week to Feature Parity



Progress 0%

YAML

glaml Public

Watch 1

Fork 4

Starred 7

main 1 Branch 6 Tags

Go to file

Add file

Code

katekyy remove deprecations ✓

0ce4dc last year 11 Commits

github/workflows	update workflows	last year
src	remove deprecations	last year
test	rewrite	last year
.gitignore	remove deprecations	last year
LICENSE	initial	2 years ago
README.md	update README after rewrite	last year
develop.sh	initial	2 years ago
gleam.toml	remove deprecations	last year
manifest.toml	remove deprecations	last year

About

A Glean wrapper around yamerl that enables your app to read YAML.

- Readme
- MIT license
- Activity
- 7 stars
- 1 watching
- 4 forks

Report repository

Releases

6 tags

Contributors 2

katekyy

Ok, I Miss RSpec

```
pub fn is_last_char_test() {
  describe("is_last_char", fn() {
    it("should return true for empty string at index 0", fn() {
      is_last_char("", 0)
      |> gleeunit.be_true
    })
  })
}
```



rob_d ❤️ LUCY 11/6/25, 7:40 AM

Did some skimming this morning - looks like `gleeunit` is more or less the defacto test library? Found some interesting alternatives (i.e. `glacier` and another that does some behavior driven type describes), but the 20 or so repos I looked at with significant codebases (possibly need a greater corpus of repos to look at...) all use `gleeunit`.

I like my `describes` and `its` coming from Ruby land. My brain is wired to consume tests setup as so vs. a more "linear" or "single level" set of test functions.

Was wondering if this is a me thing (my style/bias) or somewhat interesting. Like I said found a package that does this but wasn't 100% working for nesting (which ofc might be a user error) so now I have a wrapper I wrote to at least format my tests like rspec without anything more than just syntactic sugar over `gleeunit` for now lol (edited)



Papipo ❤️ LUCY 11/6/25, 7:47 AM

nesting obscures tests IMO. If you really need complex setups, I'd rather use helpers and call them explicitly within the test

also, try to have a pure core because testing pure functions is 🍷

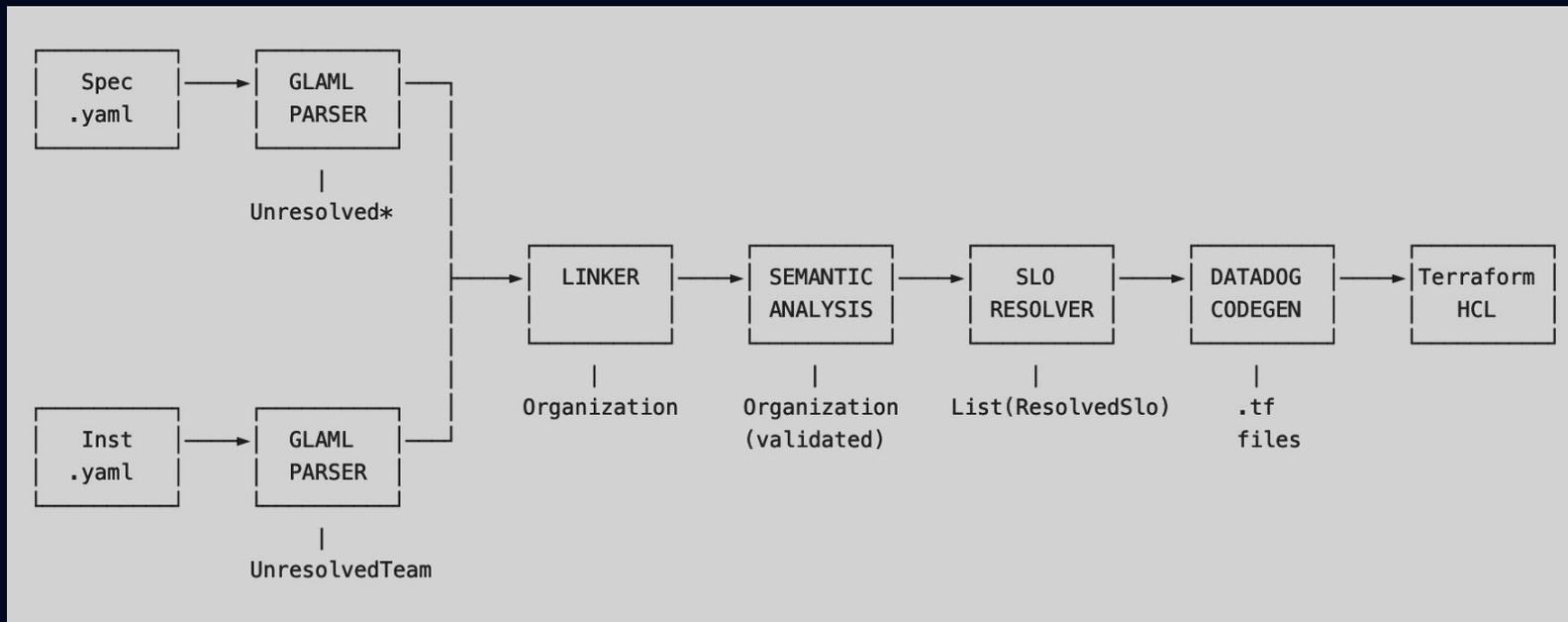
I also come from ruby, you probably need a lot of brain unwiring, but it'll be worth it



Table Driven Testing

```
pub fn extract_paren_content_test() {  
  [  
    #("happy path - simple parens", "(String)", Ok("String")),  
    #("nested parens", "(Dict(String, Integer))", Ok("Dict(String, Integer)")),  
    #("prefix before parens", "List(String)", Ok("String")),  
    #("no parens returns Error", "no parens", Error(Nil)),  
    #("content after closing paren returns Error", "(String) { x | x in { a } }", Error(Nil)),  
  ]  
  |> test_helpers.array_based_test_executor_1(  
    parsing_utils.extract_paren_content,  
  )  
}
```

Checkpoint: a Compiler in a week



Parse Don't Validate

ALEXIS KING

[HOME](#) [ABOUT ME](#)

Parse, don't validate

2019-11-05 • [functional programming](#), [haskell](#), [types](#)

Historically, I've struggled to find a concise, simple way to explain what it means to practice type-driven design. Too often, when someone asks me "How did you come up with this approach?" I find I can't give them a satisfying answer. I know it didn't just come to me in a vision—I have an iterative design process that doesn't require plucking the "right" approach out of thin air—yet I haven't been very successful in communicating that process to others.

However, about a month ago, I was [reflecting on Twitter](#) about the differences I experienced parsing JSON in statically- and dynamically-typed languages, and finally, I realized what I was looking for. Now I have a single, snappy slogan that encapsulates what type-driven design means to me, and better yet, it's only three words long:

Parse, don't validate.

The essence of type-driven design

Alright, I'll confess: unless you already know what type-driven design is, my catchy slogan probably doesn't mean all that much to you. Fortunately, that's what the

Extraction Pattern

Specification

```
3  
4   threshold: Float  
5
```

Compiler

```
1 // Extract the actual float value using pattern matching  
2 let threshold_value = try_extract_float(threshold_node)  
3  
4 fn try_extract_float(node: glaml.Node) -> Float {  
5     case node {  
6         glaml.NodeFloat(value) -> value  
7         _ -> panic as "Expected node to be a float"  
8     }  
9 }
```

A Fork: YAY - “yet another YAML”

A combinatorial explosion of extractors: $5 \times 4 \rightarrow 20$

Pattern	String	Int	Float	Bool
Required	<code>extract_string</code>	<code>extract_int</code>	<code>extract_float</code>	<code>extract_bool</code>
Optional	<code>extract_optional_string</code>	<code>extract_optional_int</code>	<code>extract_optional_float</code>	<code>extract_optional_bool</code>
With default	<code>extract_string_or</code>	<code>extract_int_or</code>	<code>extract_float_or</code>	<code>extract_bool_or</code>
List of	<code>extract_string_list</code>	<code>extract_int_list</code>	<code>extract_float_list</code>	<code>extract_bool_list</code>
Map of	<code>extract_string_map</code>	<code>extract_int_map</code>	<code>extract_float_map</code>	<code>extract_bool_map</code>

JSON?

★ json Public

♥ Sponsoring

👁 Unwatch 4

🍴 Fork 18

★ Starred 141

main

1 Branch 17 Tags

Go to file

Add file

Code

lpil v3.1.0 ✓

d303aa 3 months ago

116 Commits

.github/workflows

Update to use new JavaScript API

3 months ago

src

v3.1.0

3 months ago

test

Remove deprecated dynamic decoders

9 months ago

.gitignore

Convert to use Gleam build tool

5 years ago

CHANGELOG.md

v3.1.0

3 months ago

LICENCE

Licence

4 years ago

README.md

v3.0.0

9 months ago

gleam.toml

v3.1.0

3 months ago

manifest.toml

Remove deprecated dynamic decoders

9 months ago

README

Code of conduct

Apache-2.0 license

Security

json 

About

Work with JSON in Gleam!

hexdocs.pm/gleam_json/

json gleam

Readme

Apache-2.0 license

Code of conduct

Security policy

Activity

Custom properties

141 stars

4 watching

18 forks

Report repository

Releases 17

v3.1.0 Latest
on Nov 8, 2025

+ 16 releases

Decoders

[Before] Extractor

```
4 fn try_extract_float(node: glaml.Node) -> Float {
5     case node {
6         glaml.NodeFloat(value) -> value
7         _ -> panic as "Expected node to be a float"
8     }
9 }
```

[After] Decoder

```
pub fn decode_float(input: Dynamic) -> Result(Float, String) {
    decode.run(input, decode.float)
    |> result.replace_error("Unable to decode: " <> string.inspect(input))
}
```

A Makeover (sort of)

(YAML) Specification

```
name: Some_Expectation
blueprint: Some_Blueprint
provides:
  threshold: 99.9
  endpoint: /sign-in
```

(JSON) Specification

```
{
  "name": "Some_Expectation",
  "blueprint": "Some_Blueprint",
  "provides": {
    "threshold": 99.9,
    "endpoint": "/sign-in"
  }
}
```

Example Type System Addition

```
pub type AcceptedTypes {  
  PrimitiveType(PrimitiveTypes)  
  CollectionType(CollectionTypes(AcceptedTypes))  
  ModifierType(ModifierTypes(AcceptedTypes))  
  RefinementType(RefinementTypes(AcceptedTypes))  
  RecordType(dict.Dict(String, AcceptedTypes))  
  SomeNewType(String)  
}
```

Exhaustiveness Checks as a Checklist

CLI (7 errors returned)

```
[+ caffeine_lang git:(main) × gleam build
Compiling caffeine_lang
error: Inexhaustive patterns
/Users/rdurst/BrickellResearch/caffeine/caffeine_lang/src/caffeine_lang/types.gleam:323:3
323 |     case accepted_type {
324 |       PrimitiveType(primitive_type) -> primitive_type_to_string(primitive_type)
325 |       CollectionType(collection_type) ->
326 |         collection_type_to_string(collection_type)
330 |       RecordType(fields) -> record_type_to_string(fields)
331 |     }
```

This case expression does not have a pattern for all possible values. If it is run on one of the values without a pattern then it will crash.

The missing patterns are:

```
SomeNewType(_)
error: Inexhaustive patterns
/Users/rdurst/BrickellResearch/caffeine/caffeine_lang/src/caffeine_lang/types.gleam:888:3
888 |     case typ {
889 |       PrimitiveType(primitive) ->
890 |         validate_primitive_default_value(primitive, value)
891 |       RefinementType(refinement) ->
984 |       RecordType(_) -> Error(Nil)
985 |     }
```

IDE (7 sections underlined)

```
pub fn validate_value(
  accepted_type: AcceptedTypes,
  val: Value,
) -> Result(Value, List(ValidationError)) {
  case accepted_type {
    PrimitiveType(primitive) -> validate_primitive_value(primitive, val)
    CollectionType(collection) -> validate_collection_value(collection, val)
    ModifierType(modifier) -> validate_modifier_value(modifier, val)
    RefinementType(refinement) -> validate_refinement_value(refinement, val)
    RecordType(fields) -> validate_record_value(fields, val)
  }
}
```

A Glow Up: Docs Next to Code

BLUEPRINT

blueprints.caffeine

```
Blueprints for "SLO"
* "Simple_SLO":
  Requires { service_name: String }
  Provides {
    vendor: "datadog",
    indicators: {
      numerator: "sum:successes{$$service->service_name$$}",
      denominator: "sum:total{$$service->service_name$$}"
    },
    evaluation: "numerator / denominator"
  }
}
```

EXPECTATION

tour/demo/service.caffeine

```
Expectations for "Simple_SLO"
## This SLO is useful because we
## need to be mindful of how often
## hello world succeeds.
* "My First SLO":
  Provides {
    service_name: "hello-world",
    threshold: 99.9,
  }
## So while everyone "hello worlds"
## not everyone will fuzzbuzz. Is
## this a right of passage, idk?
## It is unsupported in Caffeine...
* "My Second SLO":
  Provides {
    service_name: "fuzzbuzz",
    threshold: 99.99,
  }
}
```

Checkpoint: a Language



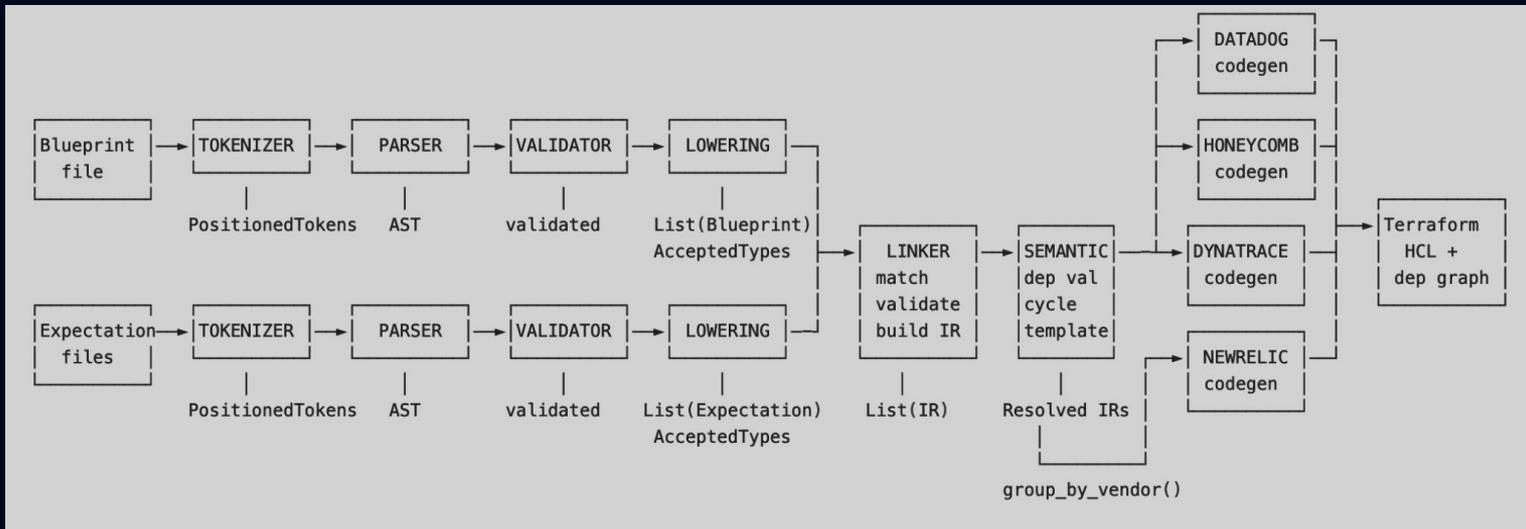
[user] Simplify user interface



[dev] Reduce maintenance burden



[compiler] Enable tooling to create a pit of success



Packaging Caffeine

1. **gleam export erlang-shipment or gleescript**
Requires Erlang on the target.
2. **Docker**
Works, but awkward for a CLI.
3. **Nix**
*Nix is great - just a me problem?**
4. **Deno**
Target JS → Deno Compile → Binary

Shoutout to *gleative* and *garnet* both of which take this same approach!

**Check out “A Gleam Burrito”!*

Checkpoint: a Distributable Language

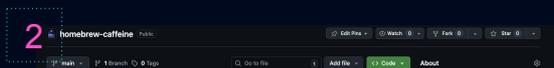


Our CI/CD



Github Releases

Asset	SHA-256	Size	Time
caffeine-4.3.1-checksums.sha256	sha256:978a8354555d8a...	493 Bytes	3 days ago
caffeine-4.3.1-linux-arm64.tar.gz	sha256:31f5361753f68e...	39.8 MB	3 days ago
caffeine-4.3.1-linux-x64.tar.gz	sha256:45b1981c878ed2...	37 MB	3 days ago
caffeine-4.3.1-macos-arm64.tar.gz	sha256:53d0f36d79598f...	46.4 MB	3 days ago
caffeine-4.3.1-macos-x64.tar.gz	sha256:5edf2e49d3866...	43.3 MB	3 days ago
caffeine-4.3.1-windows-x64.zip	sha256:aef8163b486561...	83.2 MB	3 days ago
Source code (zip)			3 days ago
Source code (tar.gz)			3 days ago



Homebrew Tap

Installation

```
brew tap robikallresearch/caffeine
brew install caffeine_lang
```

What is Caffeine Lang?

Caffeine Lang is a programming language. Visit [our repository](#) for more information.

Available Formulae

- caffeine_lang - The Caffeine programming language

Updating

To update to the latest version:

```
brew update
brew upgrade caffeine_lang
```

Releases

No releases published

Packages

No packages published

Contributors

- github-actions[bot]
- robertDumit · Rob Dumit

Languages

Ruby 100.0%



Website Tour

```
def build do
  # This is the default behavior for most projects.
  # You can also use `:publish` instead of `:publish_to_github` if you want to publish to GitHub.
  publish_to_github
end

def test do
  # This is the default behavior for most projects.
  # You can also use `:publish` instead of `:publish_to_github` if you want to publish to GitHub.
  publish_to_github
end
```



Release Notifications

Big #156

- Has publish step

Refactoring

- Further shift left checks to adhere to parse don't validate philosophy
- Remove existing json phase vestigial code & tests

Robikall Research
Creator of Caffeine Lang

Unsubscribe

Checkpoint: a Language Ecosystem

Command Line Interface

```
• (3.11.9) → slos git:(main) caffeine
A compiler for generating reliability artifacts from service expectation
definitions.
Version: 4.4.1

USAGE:
  caffeine ( artifacts | compile | format | lsp | types | validate ) [ ARGS ]

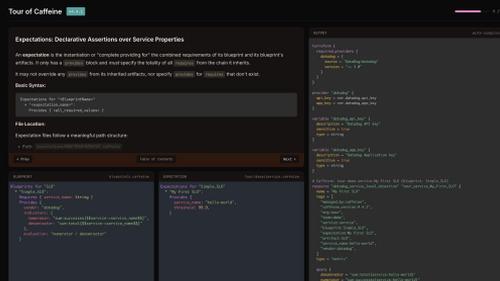
SUBCOMMANDS:
  artifacts      List available artifacts from the standard library
  compile        Compile .caffeine blueprints and expectations to
                 output
  format         Format .caffeine files
  lsp            Start the Language Server Protocol server
  types          Show the type system reference with all supported
                 types
  validate       Validate .caffeine blueprints and expectations without
                 writing output
```

Language Server

```
  cicd_success_rate — Blueprint item
  Extends: _simple_good_over_bad_env

  ## Requires: 2 fields | Provides: 1 fields (y circle)
  * "cicd_success_rate" extends [_simple_good_over_bad_env]:
  Requires { repo_name: String, job_name: List(String) }
  Provides {
    indicators: {
      numerator: "sum:circleci.completed_builds.count{${repo_name}->repo_n
      denominator: "sum:circleci.completed_builds.count{${repo_name}->repo_
    }
  }
```

Language Tour



Notifications



All Written in Gleam!

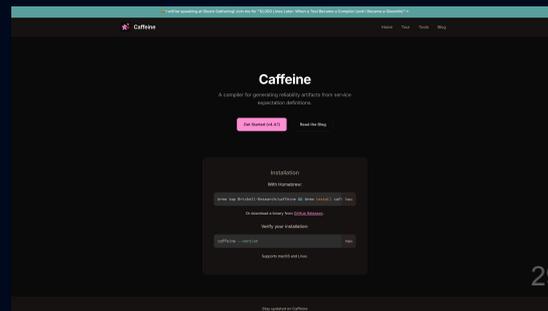
Compiler



caffeine_lang

Public

Website (*lustre-ssg*)



I Had Become a Gleamlin



Gleam is...

1. [Approachable] Small enough *to learn*
2. [Micropatterns] Functional enough *to model a compiler*
3. [Ecosystem] Practical enough *to build*
4. [Productionization] Robust enough *to ship*

A Special Thanks

glaml



glint



ansi



stdlib

Contributors 147



+ 133 contributors

filepath



argv



envoy



simplifile



json



gleeunit



~10,000 lines later we ❤️ Gleam



caffeine-lang.run



github.com/Brickell-Research/caffeine_lang



rob@brickellresearch.org