



10,000 Lines Later: When a Tool Became a Compiler

Rob Durst | February 12, 2026 | University of Utah



Intro

- Site Reliability Engineer by 🌞
- Programming Language enthusiast by 🌙
- Based out of Salt Lake City, Utah 🏔️
- I do have a yorkie & he does sometimes code...

@rob_d on





Origin Story

SLO Adoption Guide

 By Rob Durst  4 min  65  Add a reaction

SLA vs. SLO – Quick Definitions

SLA (Service Level Agreement):

A **formal, contractual commitment** made to a customer. If we fail to meet an SLA (e.g., 99.9% uptime per month), there may be **financial penalties or clawbacks**. These are defined by legal or sales teams during contract negotiations and vary by customer.

SLO (Service Level Objective):

An **internal reliability target** we set for ourselves—e.g., 99.95% uptime, <1% error rate, 250ms latency. SLOs help us stay well within our SLA thresholds and act as early warning signals before we breach customer commitments.

TL;DR:

- **SLA = Legal commitment to the customer**
- **SLO = Internal goal to help us stay reliable and avoid SLA breaches**



Origin Story

SLO Adoption Guide

By Rob Durst 4 min 65 Add a reaction

SLA vs. SLO – Quick Definitions

SLA (Service Level Agreement):

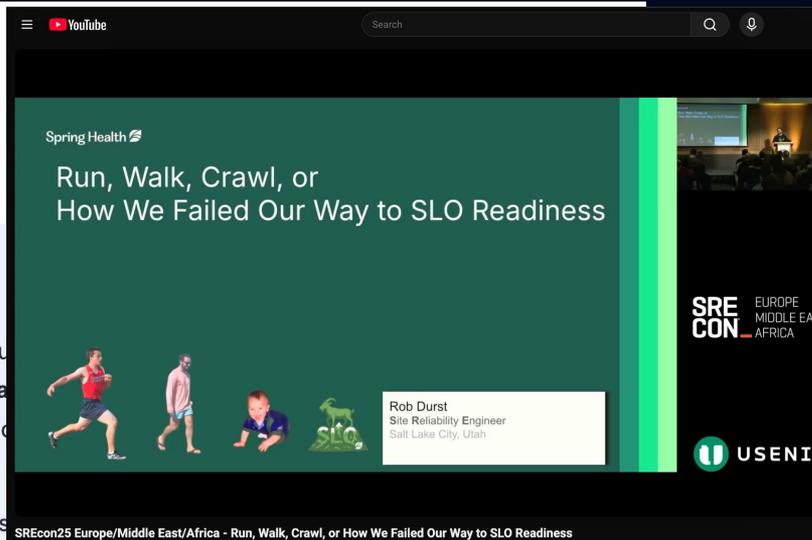
A **formal, contractual commitment** made to a customer (e.g., 99.9% uptime per month), there may be **financial penalties** imposed by legal or sales teams during contract negotiations.

SLO (Service Level Objective):

An **internal reliability target** we set for ourselves (e.g., 250ms latency). SLOs help us stay well within our SLA thresholds and act as early warning signals before we breach customer commitments.

TL;DR:

- **SLA = Legal commitment to the customer**
- **SLO = Internal goal to help us stay reliable and avoid SLA breaches**



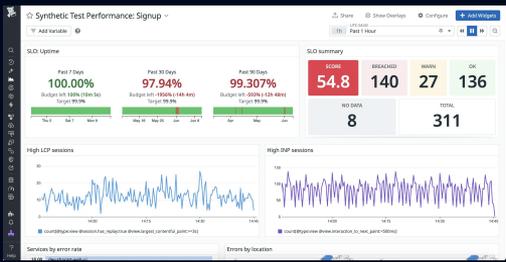


Origin Story



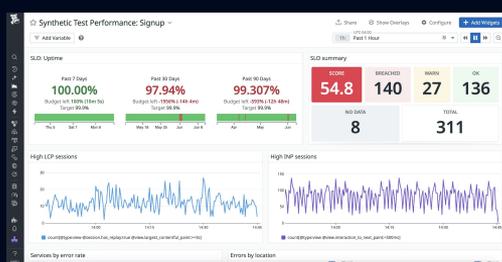
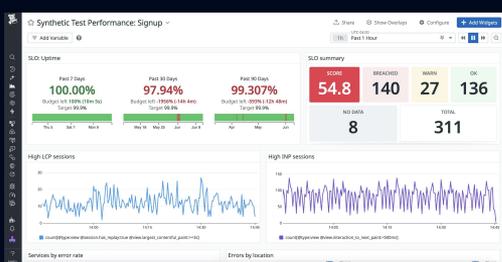
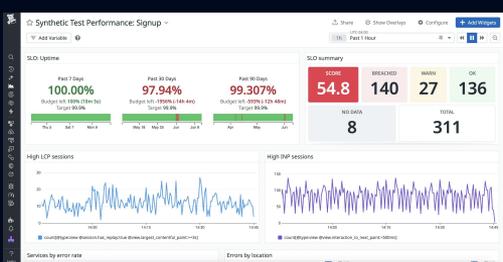
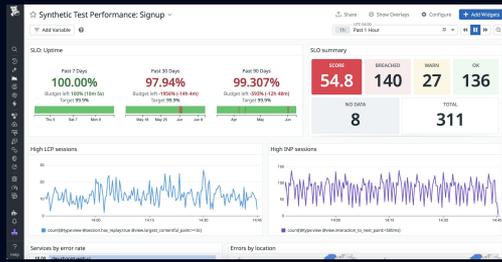
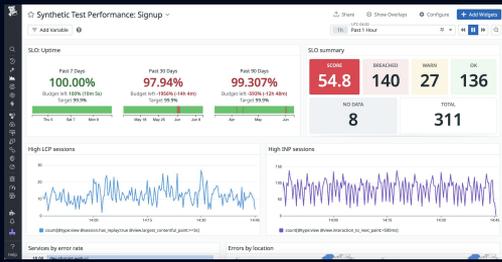
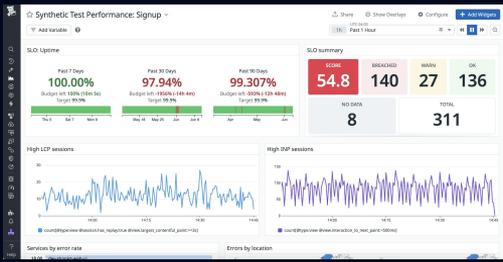
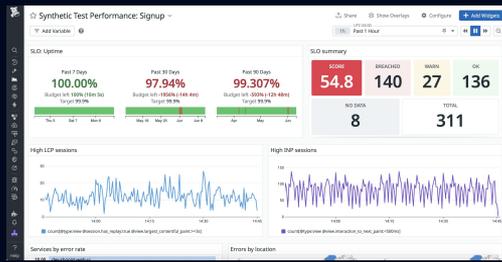
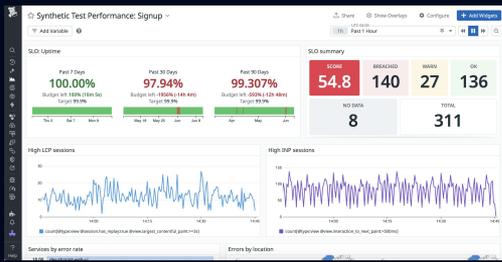
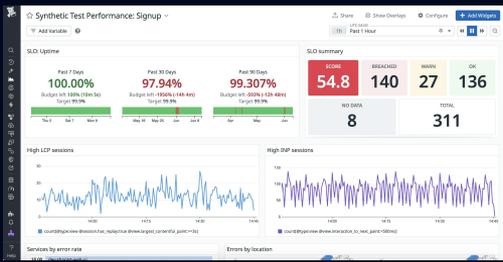


Origin Story



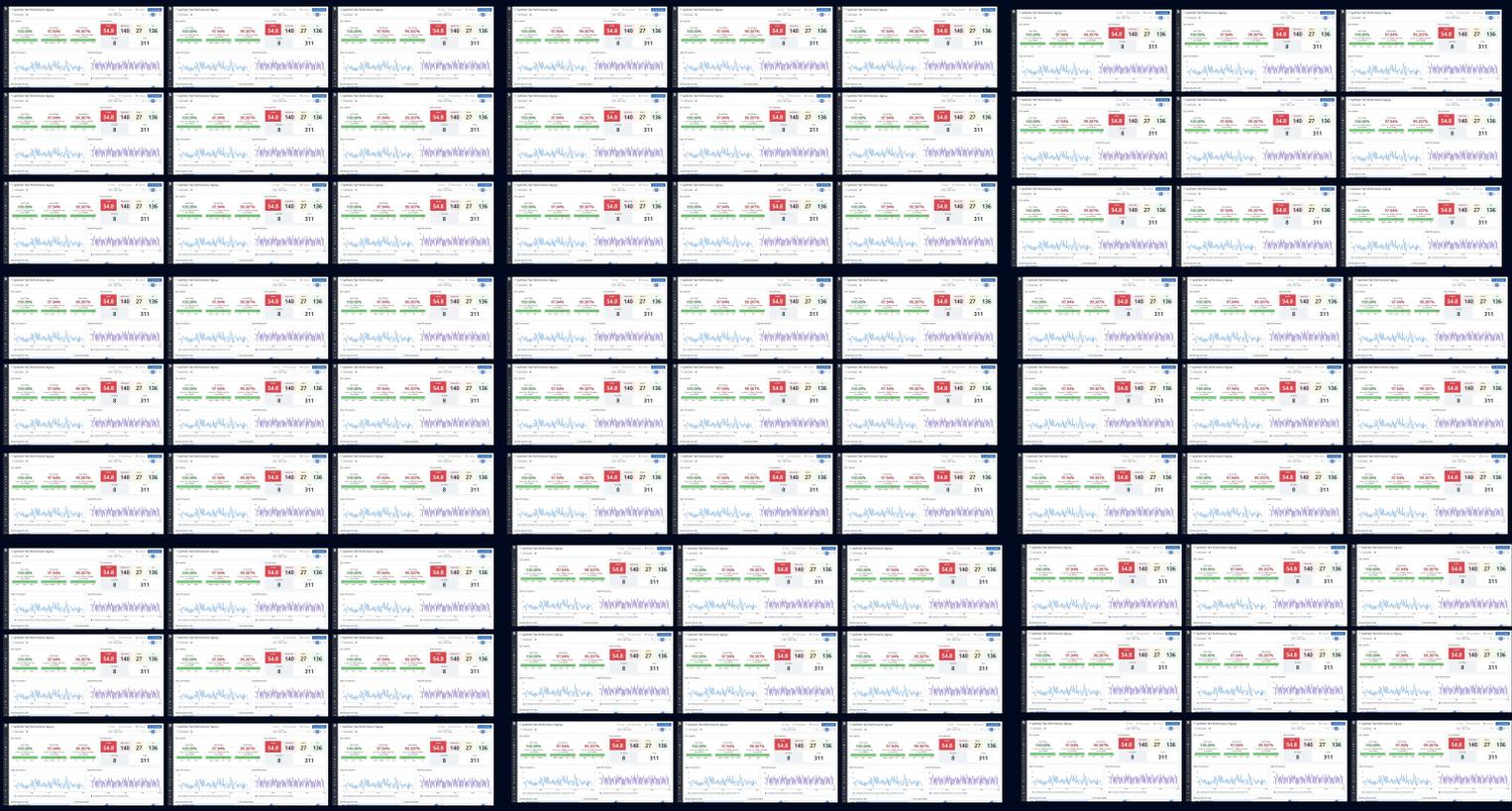


Origin Story





Origin Story





Origin Story



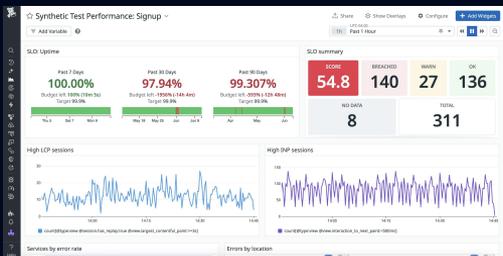


Origin Story





Origin Story

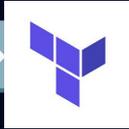


MAIN.TF

```
1 resource "datadog_monitor" "cpumonitor" {
2   name = "cpu monitor"
3   type = "metric alert"
4   message = "CPU usage alert"
5   query = "avg(last_1m):avg:system.cpu.system{*} by {host} > 60"
6   monitor_thresholds = {
7     ok = 20
8     warning = 50
9     critical = 60
10  }
11 }
```



Origin Story



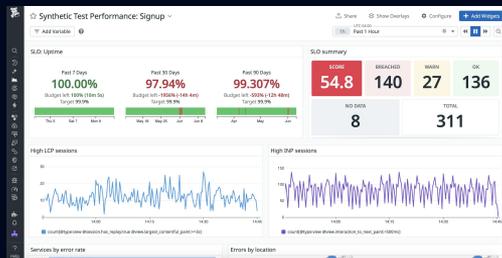


Origin Story

→ ~ generate_slos slos_src/slos

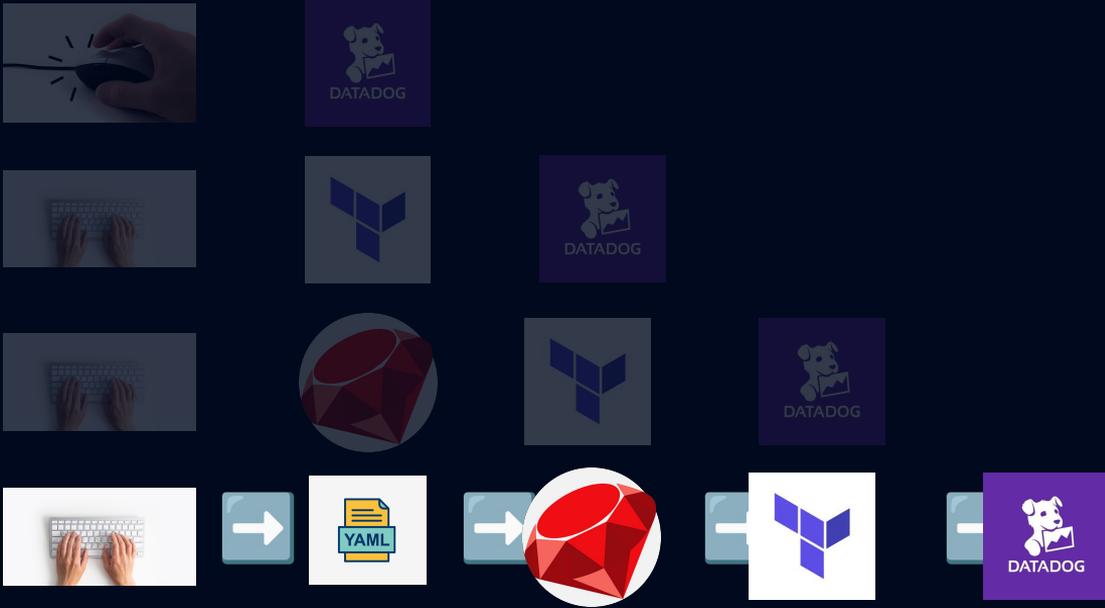
MAIN.TF

```
1 resource "datadog_monitor" "cpumonitor" {
2   name = "cpu monitor"
3   type = "metric alert"
4   message = "CPU usage alert"
5   query = "avg(last_1m):avg:system.cpu.system{*} by {host} > 60"
6   monitor_thresholds = {
7     ok = 20
8     warning = 50
9     critical = 60
10  }
11 }
```





Origin Story





Origin Story

New SLOs

Add dev utils SLO

+26 -1 ■■■■■■

New Functionality

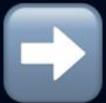
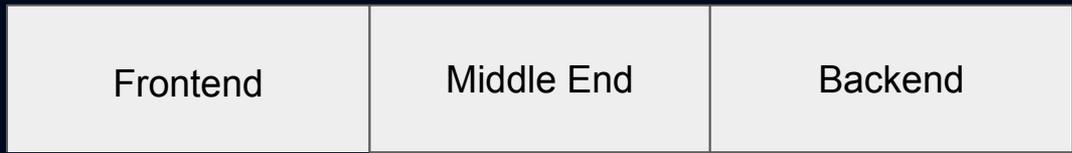
- lib
- config
 - ingestor.rb
- data
 - services.yml
 - slo_types.yml
 - teams.yml
- ez_slo/terraform_generato...
 - auth_success_rate_gene...
 - base_generator.rb
- models
 - slo.rb
 - slo_type.rb
- spec
 - models
 - slo_spec.rb
 - terraform_generators/slo
 - auth_success_rate_gene...

Commit d5c8bd4

11 files changed +656 -59 lines changed



It's a Compiler Problem





How can we...

- [User] Simplify user interface
- [Maintainer] Reduce maintenance burden
- [Process] Enable tooling to create a pit of success



Init Commit

```
Initial commit
main ··· v4.3.7 ··· v0.0.1 0 parents commit 406bc13

Filter files... 3 files changed +731 -0 lines changed Search within code

.gitignore
LICENSE
README.md

... @@ -0,0 +1,56 @@

1 +.gen
2 +.rbc
3 +.config
4 +/coverage/
5 +/InstalledFiles
6 +/pkg/
7 +/spec/reports/
8 +/spec/examples.txt
9 +/test/tmp/
10 +/test/version_tmp/
11 +/tmp/
12 +
13 + # Used by dotenv library to load environment variables.
14 + # .env
15 +
16 + # Ignore Byebug command history file.
17 + .byebug_history
18 +
19 + ## Specific to RubyMotion:
20 + .data
21 + .rep_history
22 + build/
23 + .bridgesupport
24 + build-iPhoneOS/
25 + build-iPhoneSimulator/
26 +
27 + ## Specific to RubyMotion (use of CocoaPods):
28 + #
29 + # We recommend against adding the Pods directory to your .gitignore. However
30 + # you should judge for yourself, the pros and cons are mentioned at:
31 + # https://guides.cocoapods.org/using/using-cocoapods.html#should-i-check-the-pods-directory-into-source-control
32 + #
33 + # vendor/Pods/
34 +
```



What Language?

Sorbet - Ruby + Gradual Typing



Typed Racket



Rust



Haskell

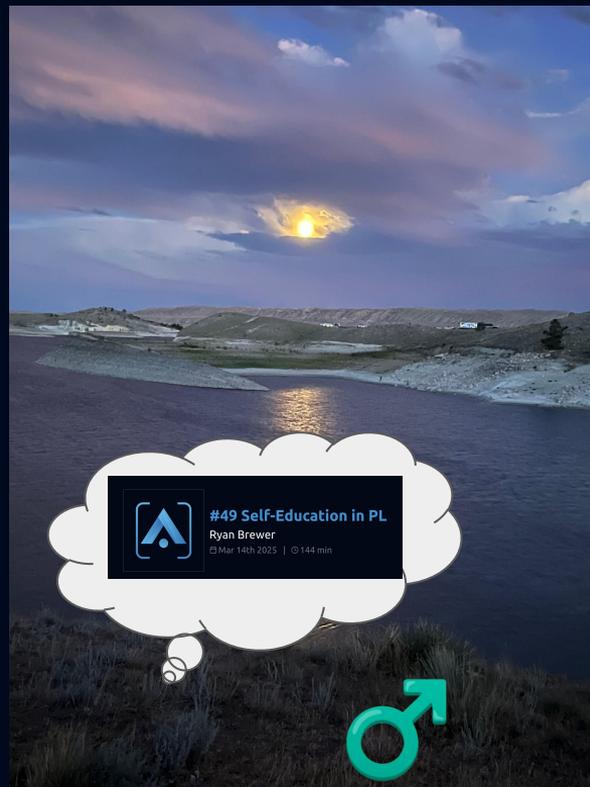


Go





Gleam?



 #49 Self-Education in PL
Ryan Brewer
Mar 14h 2025 | 144 min





Gleam.

Have your say in the [Gleam Developer Survey](#).



[News](#) [Community](#) [Sponsor](#)

[Packages](#) [Docs](#) [Code](#)



Gleam is a **friendly**
language for building
type-safe systems that
scale!

[Try Gleam](#)

The power of a type system, the expressiveness of functional programming, and the reliability of the highly concurrent, fault tolerant Erlang runtime, with a familiar and modern syntax.

```
import gleam/io

pub fn main() {
  io.println("hello, friend!")
}
```

Kindly supported by

aws



Gleam!

Gleam Language Tour gleam.run 5/10

Basics

Hello world

Modules

Unqualified imports

Type checking

Ints

Floats

Number formats

Equality

Strings

Bools

Assignments

Discard patterns

Type annotations

Type imports

Type classes

Blocks

Lists

Constants

Functions

Functions

Higher order functions

Anonymous functions

Function capture

Generic functions

Pipelines

Labelled arguments

Label shorthand syntax

Documentation comments

Deprecations

Flow control

Case expressions

Variable patterns

String patterns

List patterns

Recursion

Tail calls

List recursion

Multiple subjects

Alternative patterns

Pattern aliases

Guards

Data types

Tuples

Custom types

Records

Record accessors

Record pattern matching

Record updates

Generic custom types

Nil

Results

Basics

Hello world

Here is a tiny program that prints out the text "Hello, Joel". We'll explain how it works shortly.

In a normal Gleam project this program would be run using the command `gleam run` on the command line, but here in this tour the program is compiled and run inside your web browser, allowing you to try Gleam without installing anything on your computer.

Try changing the text being printed to `Hello, Mike!` and see what happens.

```
import gleam/io

pub fn main() {
  io.println("Hello, Joel!")
}
```

[Run code snippet](#)

Modules

Gleam code is organized into units called *modules*. A module is a bunch of definitions (of types, functions, etc.) that seem to belong together. For example, the `gleam/io` module contains a variety of functions for printing, like `println`.

All gleam code is in some module or other, whose name comes from the name of the file it's in. For example, `gleam/io` is in a file called `io.gleam` in a directory called `gleam`.

For code in one module to access code in another module, we import it using the `import` keyword, and the name used to refer to it is the last part of the module name. For example, the `gleam/io` module is referred to as `io` once imported.

The `as` keyword can be used to refer to a module by a different name. See how the `gleam/string` module is referred to as `text` here.

Comments in Gleam start with `//` and continue to the end of the line. Comments go on the line before the item they are about, not after.

```
import gleam/io
import gleam/string as text

pub fn main() {
  // Use a function from the gleam/io module
  io.println("Hello, Mike!")

  // Use a function from the gleam/string module
  io.println(text.reverse("Hello, Joel!"))
}
```

[Run code snippet](#)

Unqualified imports

Normally functions from other modules are used in a *qualified* fashion, meaning the name used to refer the module goes before function name with a dot between them. For example, `io.println("Hello!")`.

It is also possible to specify a list of functions to import from a module in an *unqualified* fashion, meaning the function name can be used without the module *qualifier* (the name and the dot) before it.

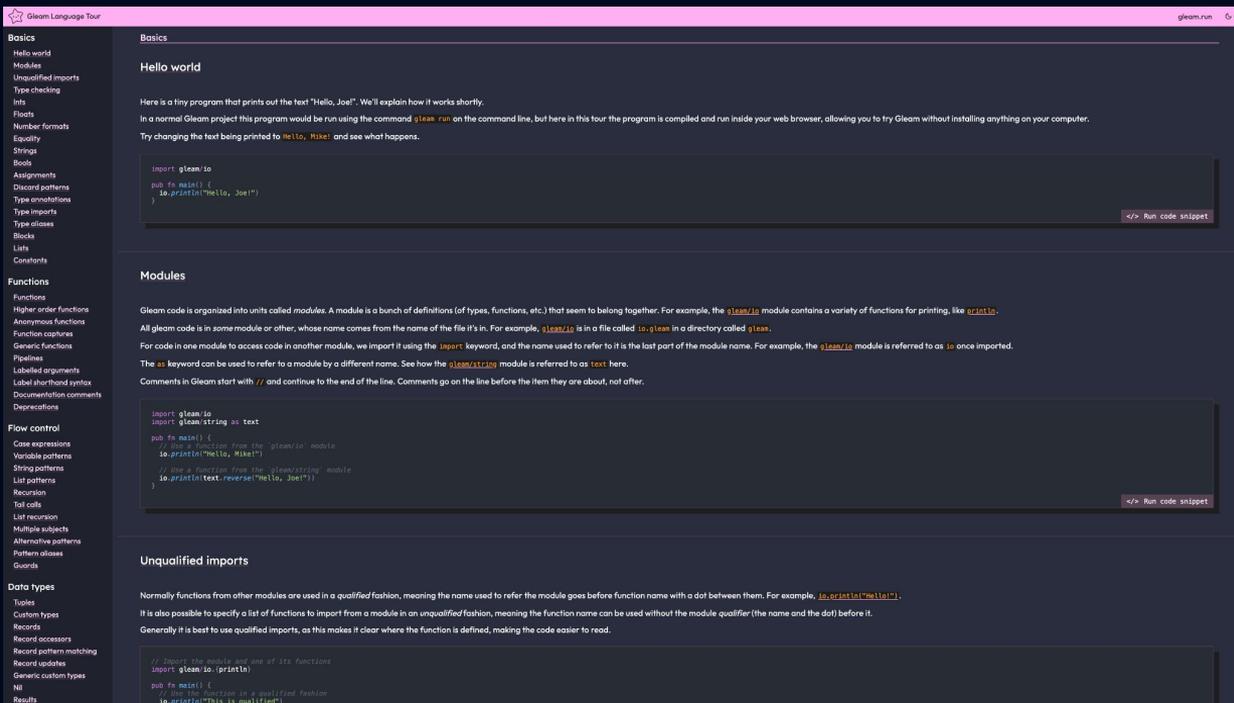
Generally it is best to use qualified imports, as this makes it clear where the function is defined, making the code easier to read.

```
// Import the module as one of its functions
import gleam/io { println }

pub fn main() {
  // Use the function in a qualified fashion
  io.println("This is qualified!")
}
```



Gleam!



The screenshot shows the 'Gleam Language Tour' website. The left sidebar contains a navigation menu with categories like Basics, Functions, Flow control, Data types, and Results. The main content area is titled 'Basics' and has a sub-section 'Hello world'. It explains that Gleam is a tiny program that prints 'Hello, Joel!' and can be run in a browser. A code snippet shows the basic Gleam code for printing a string. Below this, the 'Modules' section explains that Gleam code is organized into modules and how to import and use them. A second code snippet shows how to import a module and use its functions. Finally, the 'Unqualified imports' section explains how to use functions from other modules without the module name. A third code snippet shows unqualified imports.

Gleam Language Tour gleam.run 5/10

Basics

Hello world

Here is a tiny program that prints out the text "Hello, Joel!". We'll explain how it works shortly.

In a normal Gleam project this program would be run using the command `gleam run` on the command line, but here in this tour the program is compiled and run inside your web browser, allowing you to try Gleam without installing anything on your computer.

Try changing the text being printed to `Hello, Mike!` and see what happens.

```
import gleam/io

pub fn main() {
  io.println("Hello, Joel!")
}
```

[Run code snippet](#)

Modules

Gleam code is organized into units called *modules*. A module is a bunch of definitions (of types, functions, etc.) that seem to belong together. For example, the `gleam/io` module contains a variety of functions for printing, like `println`.

All gleam code is in some module or other, whose name comes from the name of the file it's in. For example, `gleam/io` is in a file called `io.gleam` in a directory called `gleam`.

For code in one module to access code in another module, we import it using the `import` keyword, and the name used to refer to it is the last part of the module name. For example, the `gleam/io` module is referred to as `io` once imported.

The `as` keyword can be used to refer to a module by a different name. See how the `gleam/string` module is referred to as `text` here.

Comments in Gleam start with `//` and continue to the end of the line. Comments go on the line before the item they are about, not after.

```
import gleam/io
import gleam/string as text

pub fn main() {
  // Use a function from the gleam/io module
  io.println("Hello, Mike!")

  // Use a function from the gleam/string module
  io.println(text.reverse("Hello, Joel!"))
}
```

[Run code snippet](#)

Unqualified imports

Normally functions from other modules are used in a *qualified* fashion, meaning the name used to refer the module goes before function name with a dot between them. For example, `io.println("Hello!")`.

It is also possible to specify a list of functions to import from a module in an *unqualified* fashion, meaning the function name can be used without the module *qualifier* (the name and the dot) before it. Generally it is best to use qualified imports, as this makes it clear where the function is defined, making the code easier to read.

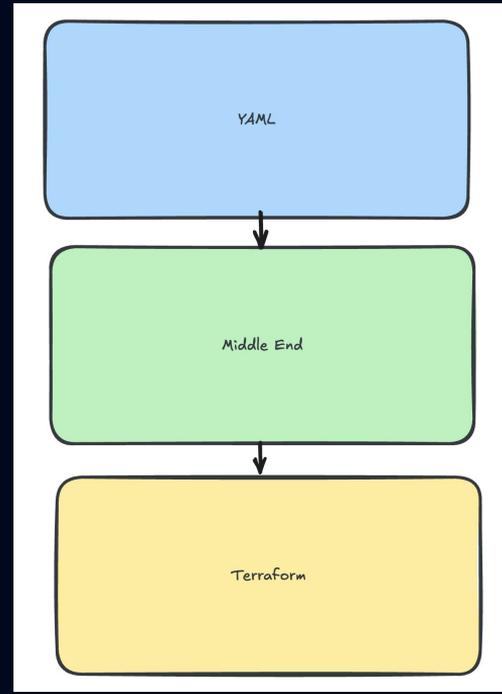
```
// Import the functions you want to use from one of its modules
import gleam/io.{println}

pub fn main() {
  // Use the function in an unqualified fashion
  io.println("This is qualified!")
}
```

[Approachability]: *Gleam is intentionally small!*



One Week to Parity





YAML

glaml

hex v3.0.2 hex docs

Glaml is a simple Gleam wrapper around yamler that enables your app to read YAML.

```
import glaml

// ...

let assert Ok([doc]) = glaml.parse_string("
stars: 7
this-is-nil:
jobs:
  - being a cat
")

glaml.select_sugar(glaml.document_root(doc), "jobs.#0")
// -> Ok(NodeStr("being a cat"))
```

Further documentation can be found at <https://hexdocs.pm/glaml>.

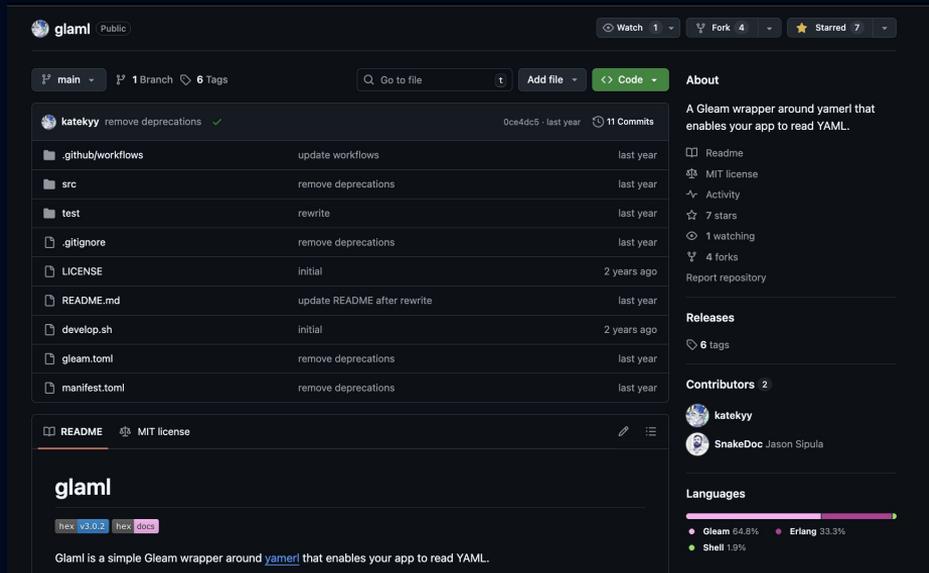
Installation

You can add `glaml` to your project by simply running the command below.

```
gleam add glaml
```

Development

```
./develop.sh # Copy Erlang headers (for lsp)
gleam test # Run the tests
```



The screenshot shows the GitHub repository for `glaml`. The repository is public and owned by `katekyy`. It has 11 commits, 4 forks, and 7 stars. The repository description is: "A Gleam wrapper around yamler that enables your app to read YAML." The repository contains several files and folders, including `.github/workflows`, `src`, `test`, `.gitignore`, `LICENSE`, `README.md`, `develop.sh`, `gleam.toml`, and `manifest.toml`. The README file is selected, showing the repository name, version (v3.0.2), and documentation link (hex docs). The README content is: "Glaml is a simple Gleam wrapper around [yamler](#) that enables your app to read YAML." The right sidebar shows repository statistics: 7 stars, 1 watching, 4 forks, and 2 contributors. The languages section shows the following distribution: Gleam 64.8%, Erlang 33.3%, and Shell 1.9%.



YAML

glaml

hex v3.0.2 hex docs

Glaml is a simple Gleam wrapper around yamler that enables your app to read YAML.

```
import glaml

// ...

let assert Ok([doc]) = glaml.parse_string("
stars: 7
this-is-nil:
jobs:
  - being a cat
")

glaml.select_sugar(glaml.document_root(doc), "jobs.#0")
// -> Ok(NodeStr("being a cat"))
```

Further documentation can be found at <https://hexdocs.pm/glaml>.

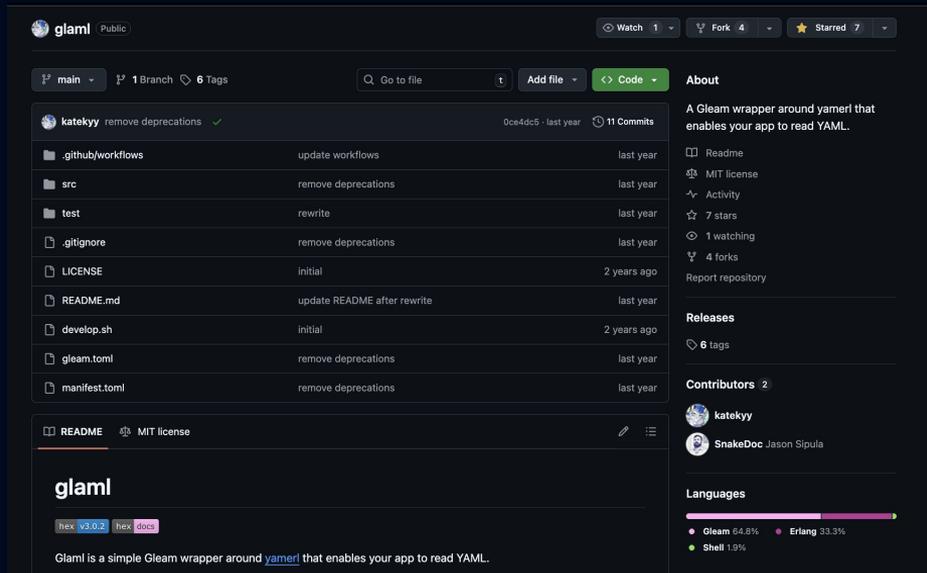
Installation

You can add `glaml` to your project by simply running the command below.

```
gleam add glaml
```

Development

```
./develop.sh # Copy Erlang headers (for lsp)
gleam test # Run the tests
```



Public

Watch 1 Fork 4 Starred 7

main 1 Branch 6 Tags

Go to file Add file Code

about

A Gleam wrapper around yamler that enables your app to read YAML.

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

README MIT license

glaml

hex v3.0.2 hex docs

Glaml is a simple Gleam wrapper around [yamler](#) 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 SnakeDoc Jason Sipula

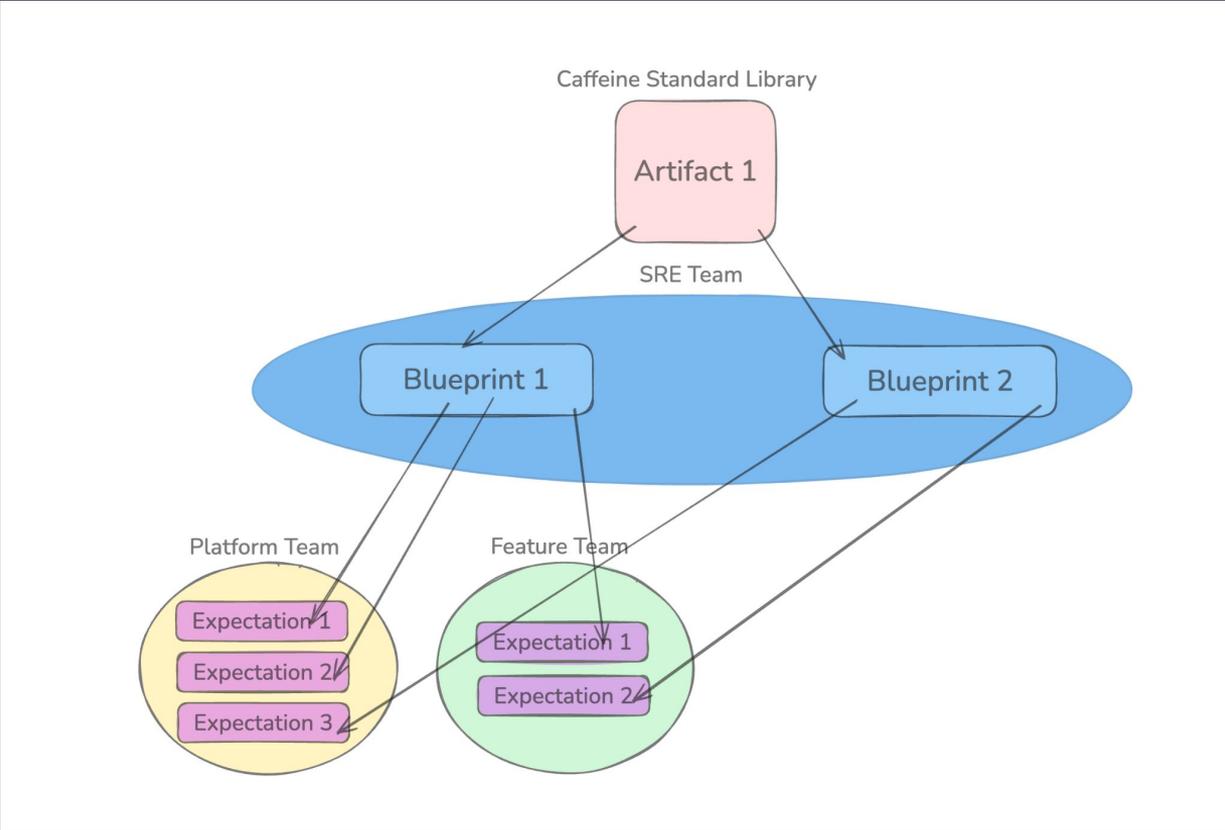
Languages

Gleam	64.8%
Erlang	33.3%
Shell	1.9%

[Ecosystem Insight]: ★ + commits + activity != maturity



How Caffeine Works





How Caffeine Works

Artifact

- Name: “SLO”
- Requires:
 - Threshold
 - Signals
 - Evaluation

Result

Threshold: _____
Signals: _____
Evaluation: _____



How Caffeine Works

Blueprint

```
name: Some_Blueprint
artifact: SLO
provides:
  signals:
    good: "successes{ $$http_endpoint->endpoint$$ }"
    total: "total{ $$http_endpoint->endpoint$$ }"
  evaluation: "good / total"
requires: [ endpoint ]
```

Result

Threshold: _____

Signals:

- **good:** "successes{ \$\$http_endpoint->endpoint\$\$ }"
- **total:** "total{ \$\$http_endpoint->endpoint\$\$ }"

Evaluation: "good / total"

Endpoint: _____



How Caffeine Works

Expectation

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

Result

Threshold: 99.9%

Signals:

- good: "successes{ \$\$http_endpoint->endpoint\$\$ }"
- total: "total{ \$\$http_endpoint->endpoint\$\$ }"

Evaluation: "good / total"

Endpoint: /sign-in



How Caffeine Works

Terraform

```
resource "datadog_service_level_objective" "tour_service_Some_Expectation" {
  name = "Some_Expectation"
  tags = [
    "org:tour",
    "team:demo",
    "service:service",
    "blueprint:Some_Blueprint",
    "expectation:Some_Expectation",
    "artifact:SLO",
    "endpoint:/sign-in",
  ]
  type = "metric"

  query {
    denominator = "total{http_endpoint:/sign-in}"
    numerator = "success{http_endpoint:/sign-in}"
  }

  thresholds {
    target = 99.9
    timeframe = "30d"
  }
}
```

Result

Threshold: 99.9%

Signals:

- good: "successes{ http_endpoint: /sign-in }"
- total: "total{ http_endpoint: /sign-in }"

Evaluation: "good / total"



How Caffeine Works

SLOs > Create SLO

1 Define your SLO measurement

Select how to measure your SLO

By Count

Measures reliability as a ratio of good / total events.

By Monitor Uptime

Measures the uptime of your monitors.

By Time Slices

Measure reliability using a custom uptime definition.

Define your SLI

Select the metric you would like to measure. Import From Monitor

a `trace.rack.request` </>

from `service:web-store, env:prod,`

`resource_name:shoppingcartcontroller_checkout` ✕

p90 by `(everything)` Σ

+ Add Query + Add Formula

Uptime is when the value is `<=` `seconds`

over `5 minute` time slices ?

2 Set your target & time window

Evaluate over a rolling time window of `7 Days` with a target of `99` %

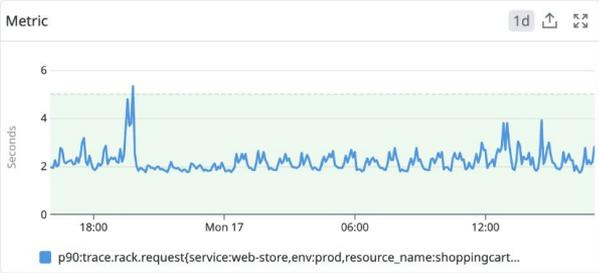
OPTIONAL

Warning threshold `99.5` %

Preview

SLI

Metric 1d



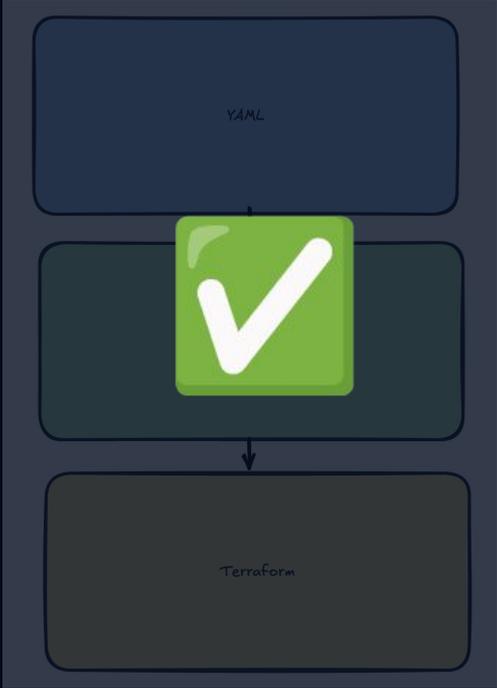
p90:trace.rack.request(service:web-store,env:prod,resource_name:shoppingcart...

STATUS

Past 7 Days	STATUS 98.462%	ERROR BUDGET LEFT -54% (-54m)
-------------	---	--



One Week to Parity





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

Parse Don't Validate

ALEXIS KING

HOME ABOUT ME

Parse, don't validate

2019-11-05 • functional programming, haskell, types

...on, I've figured out a few things. One of them is that I like 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

“Use a data structure that makes illegal states unrepresentable”

“Push the burden of proof upward as far as possible, but no further”



Type System

Primitive

- Boolean \equiv Bool
- Integer \equiv Int
- Float \equiv Float
- String \equiv String

Collection

- List(T) \equiv List(T)
- Dict(String, T) \equiv Dict(String, T)

Structured

- Record({String, T}) \equiv Dict(String, T)

Modifier

- Optional(T) \equiv T
- Defaulted(T, x) \equiv T
where x is a value of type T

Refinement

- OneOf(T, {xs}) \equiv T
where T is Integer, Float, or String and xs is a set of member values all of type T
- InclusiveRange(T, (l:h)) \equiv T
where T is Integer or Float and l and h are both of type T



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



YAY - “yet another YAML”

- A subset of all extractors...

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?



gleam_json v3.1.0 Search + K

Pages

- README
- Links
 - Website
 - Sponsor
 - Repository
 - Hex
- Modules
 - gleam/json

json

Work with JSON in Gleam!

Installation

```
gleam add gleam_json@3
```

Encoding

```
import myapp.{type Cat}
import gleam/json

pub fn cat_to_json(cat: Cat) -> String {
  json.object([
    #("name", json.string(cat.name)),
    #("lives", json.int(cat.lives)),
    #("flaws", json.null()),
    #("nicknames", json.array(cat.nicknames, of: json.string)),
  ])
  |> json.to_string
}
```

json Public Sponsoring Unwatch 4 Fork 18 Starred 141

main 1 Branch 17 Tags Go to file Add file Code About

ipili v3.1.0	d909aa9 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

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



JSON?

gleam_json v3.1.0 Search 𐀀 + K

Pages

- README
- Links
- Website
- Sponsor
- Repository
- Hex

Modules

- gleam/json

json 🐈

Work with JSON in Gleam!

Installation

```
gleam add gleam_json@3
```

Encoding

```
import myapp.{type Cat}
import gleam/json

pub fn cat_to_json(cat: Cat) -> String {
  json.object([
    #("name", json.string(cat.name)),
    #("lives", json.int(cat.lives)),
    #("flaws", json.null()),
    #("nicknames", json.array(cat.nicknames, of: json.string)),
  ])
  |> json.to_string
}
```

json Public Sponsoring Unwatch 4 Fork 18 Starred 141

main 1 Branch 17 Tags Go to file Add file Code About

ipili v3.1.0	d909aa9 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

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

[Ecosystem Insight]: common use cases yield more mature packages



Decoders

Examples

Dynamic data may come from various sources and so many different syntaxes could be used to describe or construct them. In these examples a pseudocode syntax is used to describe the data.

Simple types

This module defines decoders for simple data types such as `string`, `int`, `float`, `bit_array`, and `bool`.

```
// Data:  
// "Hello, Joe!"  
  
let result = decode.run(data, decode.string)  
assert result == Ok("Hello, Joe!")
```



Decoders

Examples

Dynamic data may come from various sources and so many different syntaxes could be used to describe or construct them. In these examples a pseudocode syntax is used to describe the data.

Simple types

This module defines decoders for simple data types such as `string`, `int`, `float`, `bit_array`, and `bool`.

```
// Data:  
// "Hello, Joe!"  
  
let result = decode.run(data, decode.string)  
assert result == Ok("Hello, Joe!")
```

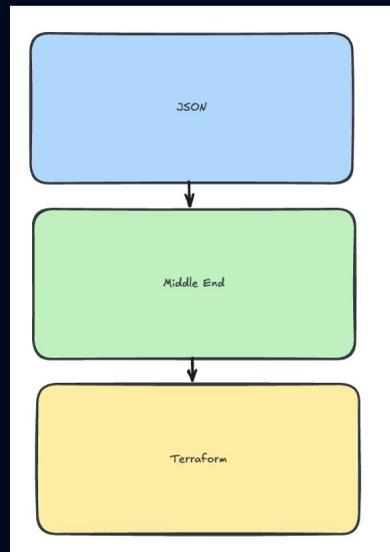
[Micropattern]: *decoders are awesome for handling untyped boundaries*



A Makeover (sort of)

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

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



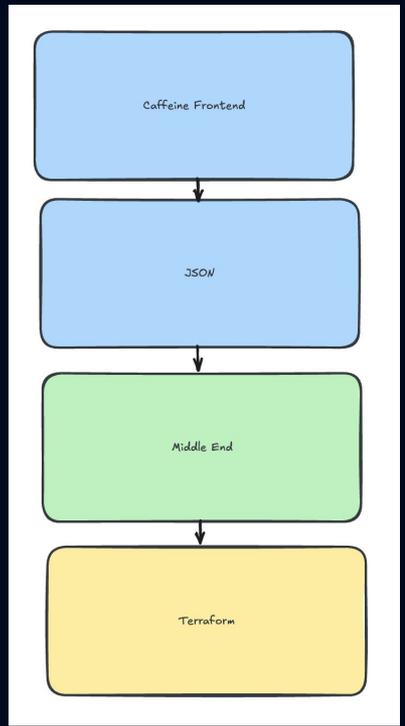
A Glow Up

```
BLUEPRINT                                blueprints.caffeine
Blueprints for "SLO"
* "api_availability":
  Requires {
    status: Boolean,
    env: String
  }

  Provides {
    evaluation: "numerator / denominator",
    indicators: {
      numerator: "sum:http.requests{{$env->env$$}",
      denominator: "sum:http.requests{{$env->env$$}"
    },
    vendor: "datadog"
  }
}

EXPECTATION                                tour/demo/service.caffeine
Expectations for "api_availability"
* "checkout_availability":
  Provides {
    env: "production",
    status: true,
    threshold: 99.95,
    window_in_days: 30
  }

* "payment_availability":
  Provides {
    env: "production",
    status: true,
    threshold: 99.0,
    window_in_days: 30
  }
}
```





Gleam Detour - Result.try

```
pub fn try(  
  result: Result(a, e),  
  apply fun: fn(a) -> Result(b, e),  
) -> Result(b, e)
```

“Updates” an `Ok` result by passing its value to a function that yields a result, and returning the yielded result. (This may “replace” the `Ok` with an `Error`.)

If the input is an `Error` rather than an `Ok`, the function is not called and the original `Error` is returned.

This function is the equivalent of calling `map` followed by `flatten`, and it is useful for chaining together multiple functions that may fail.



Gleam Detour - Use

Gleam's `use` expression is for calling functions that take a callback as an argument without increasing the indentation of the code.

All the code below the `use` becomes an anonymous function that is passed as a final argument to the function on the right hand side of the `<-`, and the assigned variables become the arguments to the anonymous function.

This code:

```
pub fn main() -> Nil {
  use a, b <- my_function
  next(a)
  next(b)
}
```

Expands into this code:

```
pub fn main() -> Nil {
  my_function(fn(a, b) {
    next(a)
    next(b)
  })
}
```

```
pub fn with_use() -> Result(String, Nil) {
  use username <- result.try(get_username())
  use password <- result.try(get_password())
  use greeting <- result.map(log_in(username, password))
  greeting <> ", " <> username
}

pub fn without_use() -> Result(String, Nil) {
  result.try(get_username(), fn(username) {
    result.try(get_password(), fn(password) {
      result.map(log_in(username, password), fn(greeting) {
        greeting <> ", " <> username
      })
    })
  })
}
```



Recursive Descent - Result.try + Use

```
fn parse_blueprint_item(  
  state: ParserState,  
  leading_comments: List(Comment),  
) -> Result(#[BlueprintItem, ParserState], ParserError) {  
  use state <- result.try(expect(state, token.SymbolStar, "*"))  
  use #(name, state) <- result.try(parse_string_literal(state))  
  use #(extends, state) <- result.try(parse_optional_extends(state))  
  use state <- result.try(expect(state, token.SymbolColon, ":"))  
  use state <- result.try(expect(state, token.KeywordRequires, "Requires"))  
  use #(requires, state) <- result.try(parse_type_struct(state))  
  use state <- result.try(expect(state, token.KeywordProvides, "Provides"))  
  use #(provides, state) <- result.try(parse_literal_struct(state))  
  Ok(#[  
    ast.BlueprintItem(name:, extends:, requires:, provides:, leading_comments:),  
    state,  
  ])  
}
```



Recursive Descent - Result.try + Use

```
fn parse_blueprint_item(  
  state: ParserState,  
  leading_comments: List(Comment),  
) -> Result(#[BlueprintItem, ParserState], ParserError) {  
  use state <- result.try(expect(state, token.SymbolStar, "*"))  
  use #(name, state) <- result.try(parse_string_literal(state))  
  use #(extends, state) <- result.try(parse_optional_extends(state))  
  use state <- result.try(expect(state, token.SymbolColon, ":"))  
  use state <- result.try(expect(state, token.KeywordRequires, "Requires"))  
  use #(requires, state) <- result.try(parse_type_struct(state))  
  use state <- result.try(expect(state, token.KeywordProvides, "Provides"))  
  use #(provides, state) <- result.try(parse_literal_struct(state))  
  Ok(#[  
    ast.BlueprintItem(name:, extends:, requires:, provides:, leading_comments:),  
    state,  
  ])  
}
```

[Micropattern]: *leverage use + result.try judiciously*



Own Frontend

BLUEPRINT

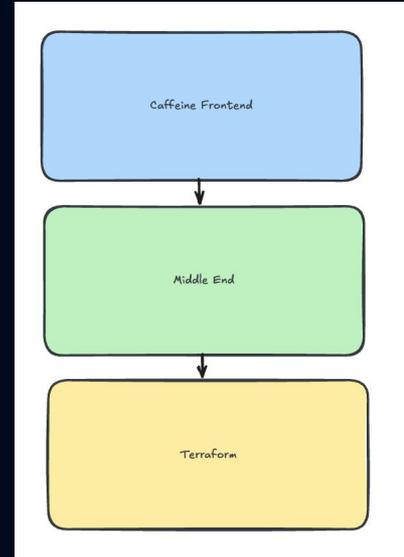
blueprints.caffeine

```
Blueprints for "SLO"  
* "api_availability":  
  Requires {  
    status: Boolean,  
    env: String  
  }  
  
  Provides {  
    evaluation: "numerator / denominator",  
    indicators: {  
      numerator: "sum:http.requests{{$env->env$$}",  
      denominator: "sum:http.requests{{$env->env$$}"  
    },  
    vendor: "datadog"  
  }  
}
```

EXPECTATION

tour/demo/service.caffeine

```
Expectations for "api_availability"  
* "checkout_availability":  
  Provides {  
    env: "production",  
    status: true,  
    threshold: 99.95,  
    window_in_days: 30  
  }  
  
* "payment_availability":  
  Provides {  
    env: "production",  
    status: true,  
    threshold: 99.0,  
    window_in_days: 30  
  }  
}
```





Ease of Extension

```
/// AcceptedTypes is a union of all the types that can be used as a "filter" over the set
/// of all possible values. This allows us to _type_ params and thus provide annotations
/// that the compiler can leverage to be a more useful guide towards the pit of success.
pub type AcceptedTypes {
  PrimitiveType(PrimitiveTypes)
  CollectionType(CollectionTypes(AcceptedTypes))
  ModifierType(ModifierTypes(AcceptedTypes))
  RefinementType(RefinementTypes(AcceptedTypes))
  RecordType(dict.Dict(String, AcceptedTypes))
}
```



Ease of Extension

```
/// AcceptedTypes is a union of all the types that can be used as a "filter" over the set
/// of all possible values. This allows us to _type_ params and thus provide annotations
/// that the compiler can leverage to be a more useful guide towards the pit of success.
pub type AcceptedTypes {
  PrimitiveType(PrimitiveTypes)
  CollectionType(CollectionTypes(AcceptedTypes))
  ModifierType(ModifierTypes(AcceptedTypes))
  RefinementType(RefinementTypes(AcceptedTypes))
  RecordType(dict.Dict(String, AcceptedTypes))
}
```

[Micropattern]: *pattern matching is your best friend*



Full Example

```
name: Some_Blueprint
artifact: SLO
provides:
  signals:
    good: "successes{ $$http_endpoint->endpoint$$ }"
    total: "total{ $$http_endpoint->endpoint$$ }"
  evaluation: "good / total"
requires: [ endpoint ]
```

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

```
Blueprints for "SLO"
* "Some_Blueprint":
  Requires { endpoint: String }
  Provides {
    indicators: {
      good: "success{ $$http_endpoint:endpoint$$}",
      total: "total{ $$http_endpoint:endpoint$$}"
    },
    evaluation: "good / total"
  }
```

```
Expectations for "Some_Blueprint"
* "Some_SLO":
  Provides { threshold: 99.9, endpoint: "/sign-in" }
```



Shipping Gleam - Problem

- gleam export erlang-shipment
- gleescript
- Docker
- Nix



Shipping Gleam - Solution

1. Ensure JS target works
2. Target JS build
3. Compile with Deno

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

gleative@0.1.0

Updated 1 year ago

Compile your gleam projects into a native executable using deno.

[Docs](#) [Hex](#)

garnet_tool@1.0.4

Updated 1 year ago

Compile gleam to single binary, via Deno and Bun.

[Docs](#) [Repo](#) [Hex](#)



Distributing Caffeine - Full Pipeline

Build and Release		Search logs	🔍	⚙️
>	Set up job		2s	
>	Checkout code		1s	
>	Setup BEAM (Erlang + Gleam)		7s	
>	Setup Deno		2s	
>	Cache Gleam dependencies		0s	
>	Build Gleam to JavaScript		1s	
>	Extract version from gleam.toml		0s	
>	Generate release notes		0s	
>	Check disk space		0s	
>	Compile binaries for all platforms		14s	
>	Create archives		27s	
>	Generate SHA256 checksums		0s	
>	Check if release exists		0s	
>	Create Release		5s	
>	Send release email		2s	
>	Publish to Hex		3s	
>	Update Homebrew formula		1s	
>	Build browser bundle		1s	
>	Update website compiler		0s	
>	Post Cache Gleam dependencies		1s	
>	Post Setup Deno		0s	
>	Post Checkout code		1s	
>	Complete job		0s	

Distributing Caffeine - Full Pipeline

1

3 days ago

 github-actions

 v4.3.1

 eb33625

Compare ▾

v4.3.1



What's New in Caffeine v4.3.1

Features

- More precise relations type for DependencyRelations in stdlib leveraging record type

Bug Fixes

- Harden LSP

Assets ⁸

 caffeine-4.3.1-checksums.sha256	sha256:978a0c58455a60...		493 Bytes	3 days ago
 caffeine-4.3.1-linux-arm64.tar.gz	sha256:3ff516f755f60e...		39.8 MB	3 days ago
 caffeine-4.3.1-linux-x64.tar.gz	sha256:450169c1878ed2...		37 MB	3 days ago
 caffeine-4.3.1-macos-arm64.tar.gz	sha256:63ddf3de279d0f...		46.4 MB	3 days ago
 caffeine-4.3.1-macos-x64.tar.gz	sha256:bad7d2af9d38d6...		43.3 MB	3 days ago
 caffeine-4.3.1-windows-x64.zip	sha256:aef01b3b406561...		53.2 MB	3 days ago
 Source code (zip)				3 days ago
 Source code (tar.gz)				3 days ago





Distributing Caffeine - Full Pipeline

2 **homebrew-caffeine** Public

main 1 Branch 0 Tags

github-actions[bot] Update caffeine_lang to 4.3.5 3c95402 · 3 days ago 63 Commits

Formula	Update caffeine_lang to 4.3.5	3 days ago
README.md	Update tap references to Brickell-Research organization	3 months ago

README

Homebrew Tap for Caffeine Lang

Installation

```
brew tap brickell-research/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
```

About

No description, website, or topics provided.

- Readme
- Activity
- Custom properties
- 0 stars
- 0 watching
- 0 forks
- Audit log
- Report repository

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Contributors 2

- github-actions[bot]
- robertDurst Rob Durst

Languages

- Ruby 100.0%



Distributing Caffeine - Full Pipeline

3

Tour of Caffeine v4.3.5

1 / 12

Welcome to Caffeine

For the returning user or enthusiastic passer-by who wants to see the full picture right away, jump to the end (lesson 12) for a complete example. Use the table of contents button below to quickly jump between lessons.

Caffeine is a programming language for generating reliability artifacts from service expectation definitions.

We built Caffeine to aid reliability-conscious software operators in the era of highly dynamic systems, aligning with the views of Chad Fowler's Phoenix Architecture, Google's evolution of SRE, the systems thinking outlined by Donella Meadows and Nancy Leveson, the "test in prod" mindset of Charity Majors, and many others.

Our definition of Caffeine above is intentionally vague, affording us a wide domain beyond the initial SLO focused use case, capturing the broader vision of the language.

[What's new in Caffeine v4.3.5](#)

[< Prev](#)

[Table of Contents](#)

[Next >](#)

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"
* "My First SLO":
  Provides {
    service_name: "hello-world",
    threshold: 99.9,
  }
}
```

OUTPUT

auto-compiles

```
terraform {
  required_providers {
    datadog = {
      source = "Datadog/datadog"
      version = "~> 3.0"
    }
  }
}

provider "datadog" {
  api_key = var.datadog_api_key
  app_key = var.datadog_app_key
}

variable "datadog_api_key" {
  description = "Datadog API key"
  sensitive = true
  type = string
}

variable "datadog_app_key" {
  description = "Datadog Application key"
  sensitive = true
  type = string
}

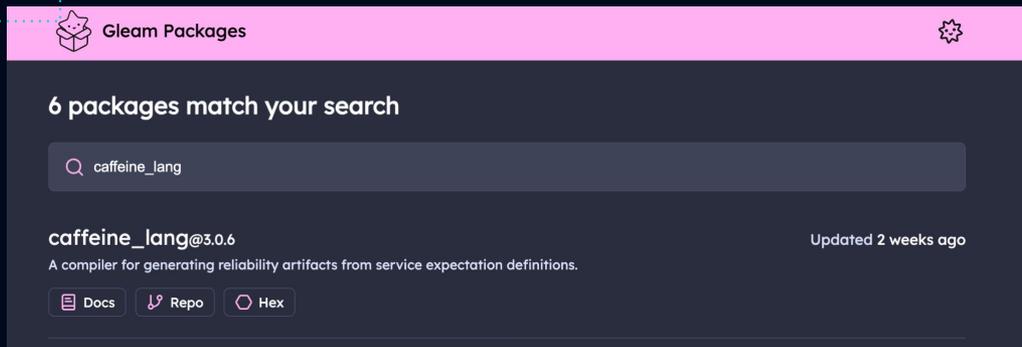
# Caffeine: tour.demo.service.My First SLO (blueprint: Simple_SLO)
resource "datadog_service_level_objective" "tour_service_my_first_slo" {
  name = "My First SLO"
  tags = [
    "managed_by:caffeine",
    "caffeine_version:4.3.5",
    "org:tour",
    "team:demo",
    "service:service",
    "blueprint:Simple_SLO",
    "expectations:My First SLO",
    "artifact:SLO",
    "service_name:hello-world",
    "vendor:datadog",
  ]
  type = "metric"

  query {
    denominator = "sum:total{service:hello-world}"
    numerator = "sum:successes{service:hello-world}"
  }
}
```



Distributing Caffeine - Full Pipeline

4

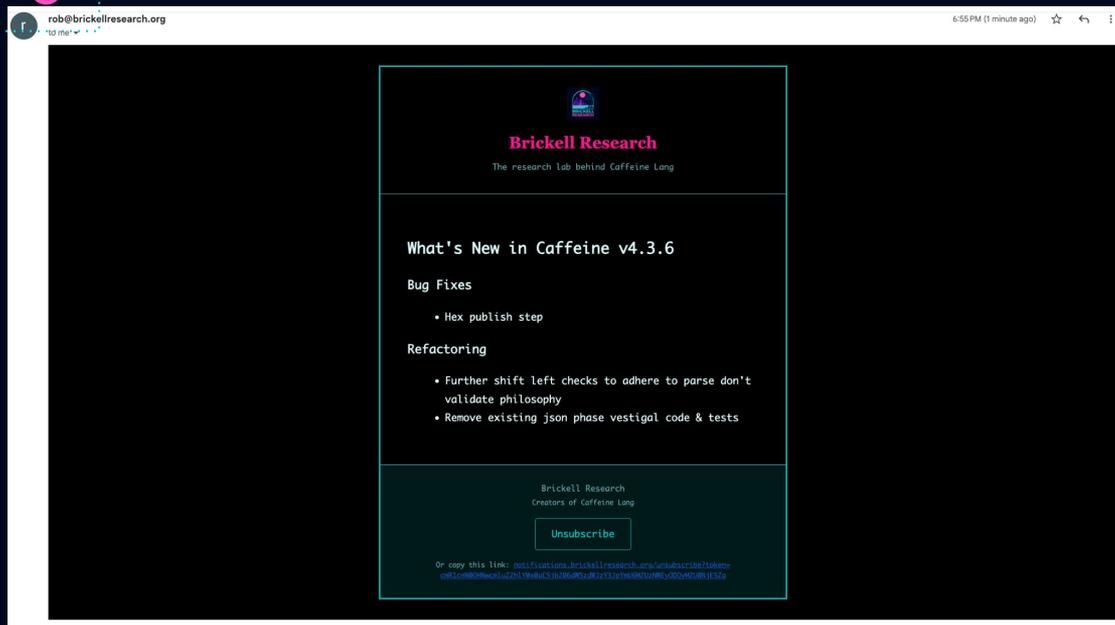


The screenshot shows the Gleam Packages search interface. At the top, there is a header with the Gleam logo and the text "Gleam Packages" on the left, and a gear icon on the right. Below the header, it says "6 packages match your search". A search bar contains the text "caffeine_lang". Below the search bar, the package "caffeine_lang@3.0.6" is listed, with the description "A compiler for generating reliability artifacts from service expectation definitions." and the text "Updated 2 weeks ago" on the right. At the bottom, there are three buttons: "Docs" with a document icon, "Repo" with a fork icon, and "Hex" with a hexagon icon.

Distributing Caffeine - Full Pipeline



5



rob@brickellresearch.org 6:55 PM (1 minute ago) ☆ ↶ ⋮


Brickell Research
The research lab behind Caffeine Lang

What's New in Caffeine v4.3.6

Bug Fixes

- Hex publish step

Refactoring

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

Brickell Research
Creators of Caffeine Lang

[Unsubscribe](#)

Or copy this link: notifications_brickellresearch_org/unsubscribe?token=ca51a0e029e6c3a42d379e46513b28685ca88213216169850a98645024502941e312

We Have a Programming Language!



🗣️ I will be speaking at Gleam Gathering! Join me for "10,000 Lines Later: When a Tool Became a Compiler (and I Became a Gleamlin)" →

 Caffeine

[Home](#) [Tour](#) [Tools](#) [Blog](#)

Caffeine

A compiler for generating reliability artifacts from service expectation definitions.

[Get Started \(v4.3.7\)](#)

[Read the Blog](#)

Installation

With Homebrew:

```
brew tap Brickell-Research/caffeine && brew install caf1 copy
```

Or download a binary from [GitHub Releases](#).

Verify your installation:

```
caffeine --version copy
```

Supports macOS and Linux.

Stay updated on Caffeine

[Subscribe](#)



Gleam is...

1. [Approachable] Small enough to learn



Gleam is...

1. [Approachable] Small enough to learn
2. [Micropatterns] Functional enough to model a compiler



Gleam is...

1. [Approachable] Small enough to learn
2. [Micropatterns] Functional enough to model a compiler
3. [Ecosystem] Practical enough to ship



A Special Thanks

glaml



glint



ansi



stdlib

Contributors 147



+ 133 contributors

filepath



argv



envoy



simplifile



json



gleeunit





We're Brickell Research and we ❤️ Gleam



caffeine-lang.run



github.com/Brickell-Research/caffeine_lang



rob@brickellresearch.org