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

Rob Durst | February 12, 2026 | University of Utah

Systems thinking, without the thinking.

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

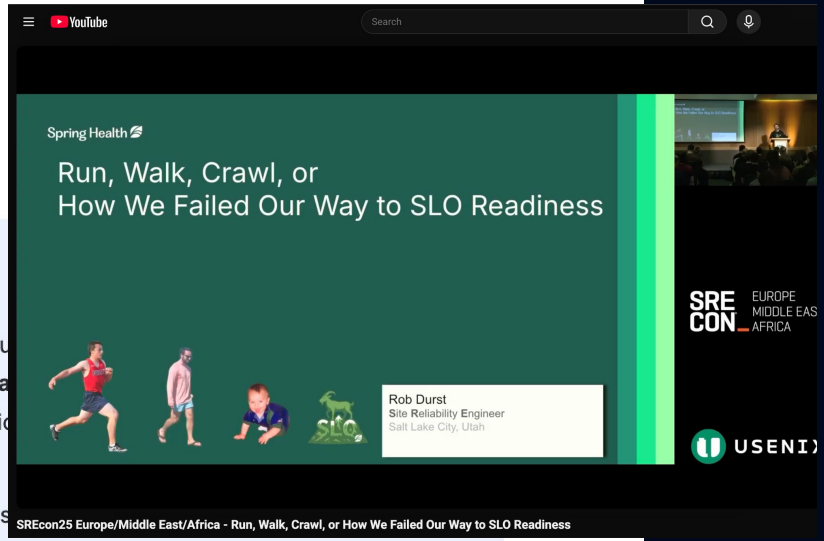
SLA vs. SLO – Quick Definitions

SLA (Service Level Agreement):
A **formal, contractual commitment** made to a cu
99.9% uptime per month), there may be **financia**
by legal or sales teams during contract negotiatio

SLO (Service Level Objective):
An **internal reliability target** we set for ourselves
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**



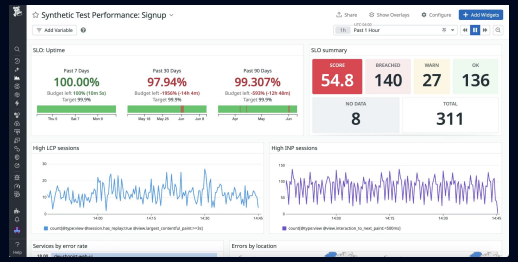
BRICKELL RESEARCH

Origin Story

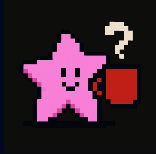
➡️

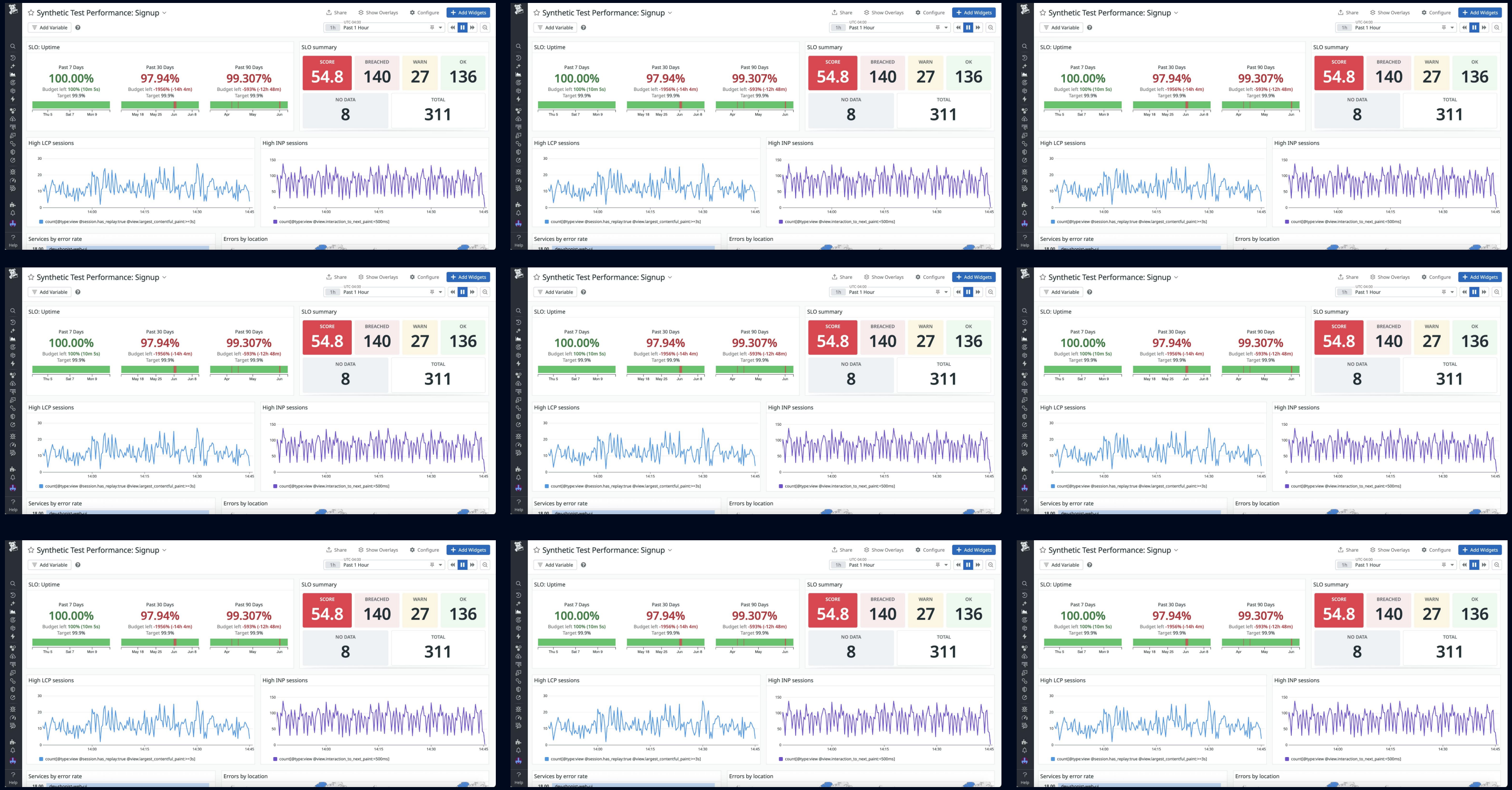
DATADOG

Origin Story



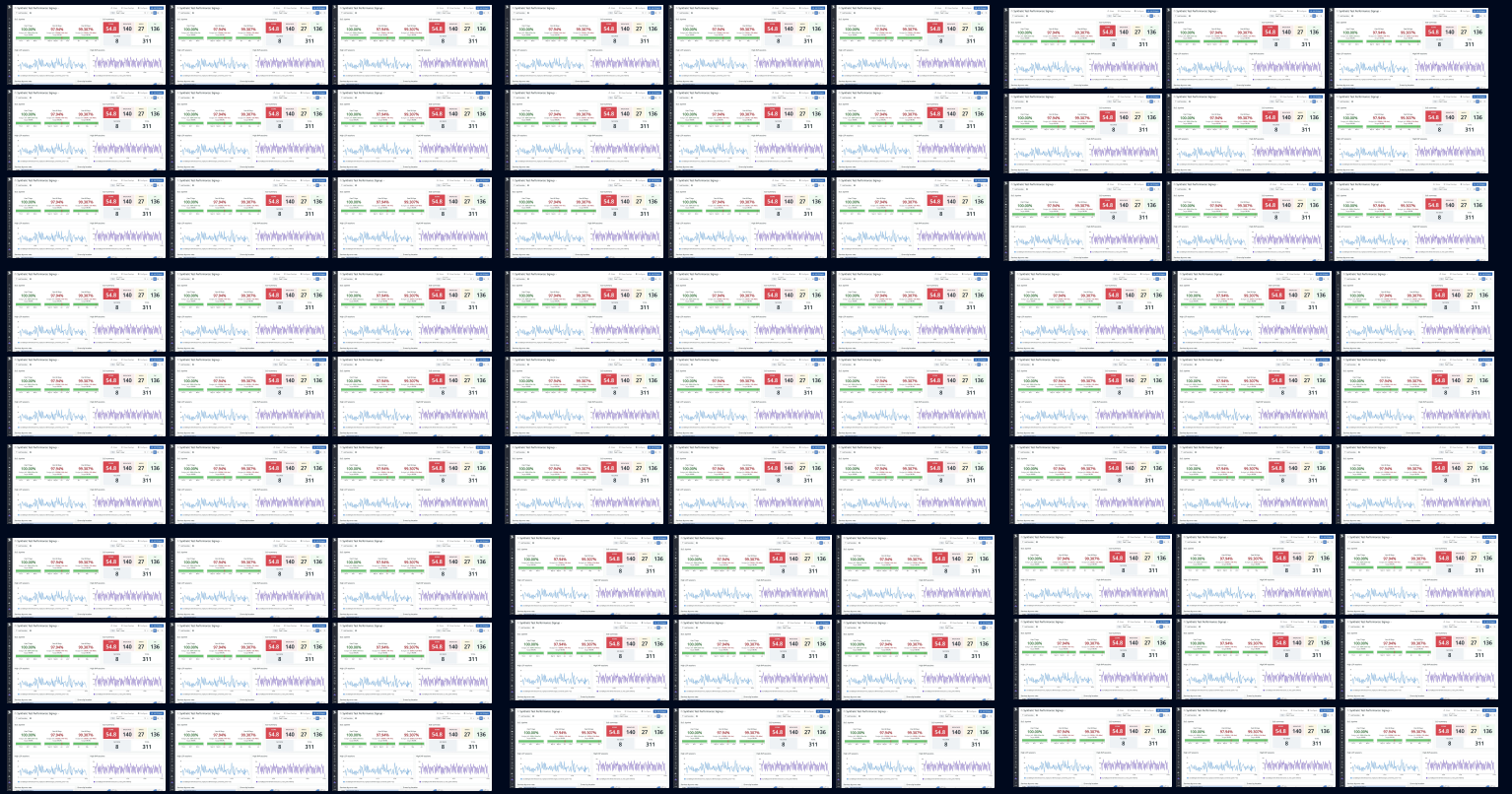
Origin Story





BRICKELL RESEARCH

Origin Story



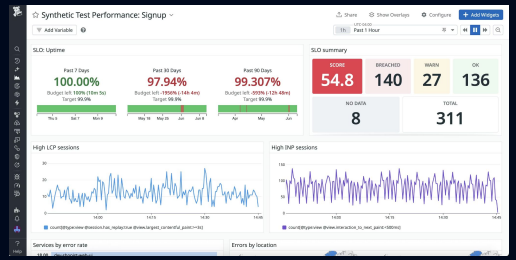
BRICKELL RESEARCH

Origin Story

BRICKELL RESEARCH

Origin Story

Origin Story

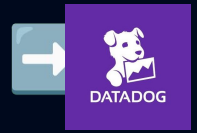


MAIN.TF

```
resource "datadog_monitor" "cpumonitor" {
  name = "cpu monitor"
  type = "metric alert"
  message = "CPU usage alert"
  query = "avg(last_1m):avg:system.cpu.system{*} by {host} > 60"
  monitor_thresholds = {
    ok = 20
    warning = 50
    critical = 60
  }
}
```

BRICKELL RESEARCH

Origin Story

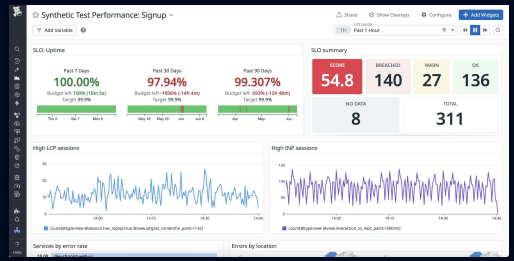


Origin Story

```
➜  ~ generate_slos slos_src/slos
```

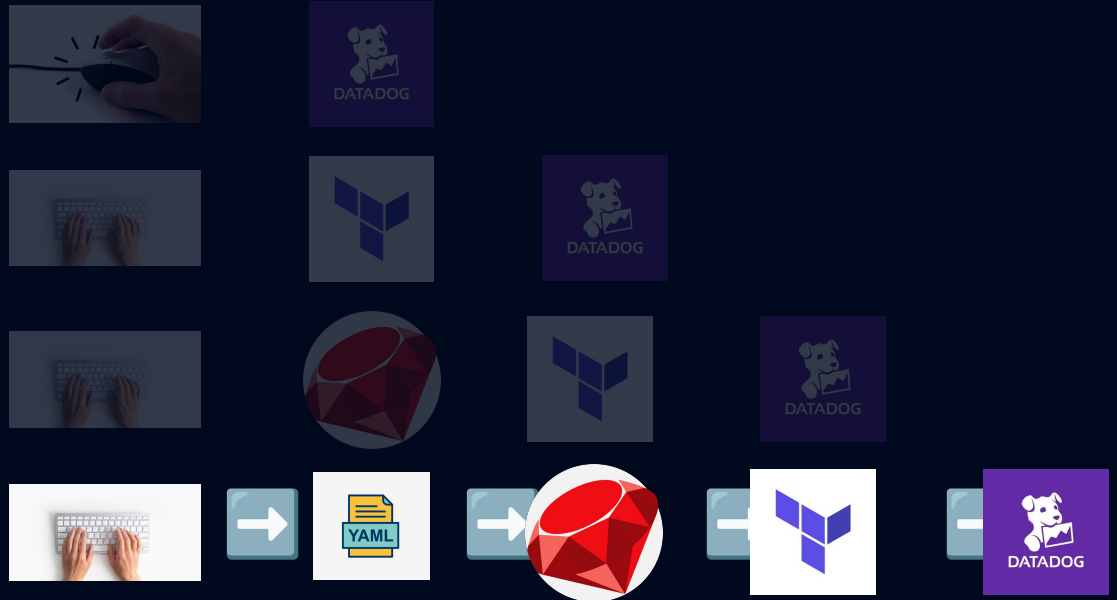
MAIN.TF

```
resource "datadog_monitor" "cpumonitor" {
  name = "cpu monitor"
  type = "metric alert"
  message = "CPU usage alert"
  query = "avg(last_1m):avg:system.cpu.system{*} by {host} > 60"
  monitor_thresholds = {
    ok = 20
    warning = 50
    critical = 60
  }
}
```



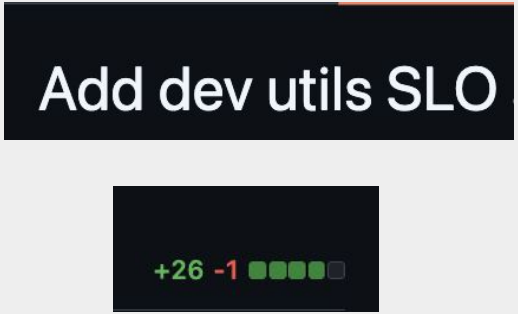
BRICKELL RESEARCH

Origin Story

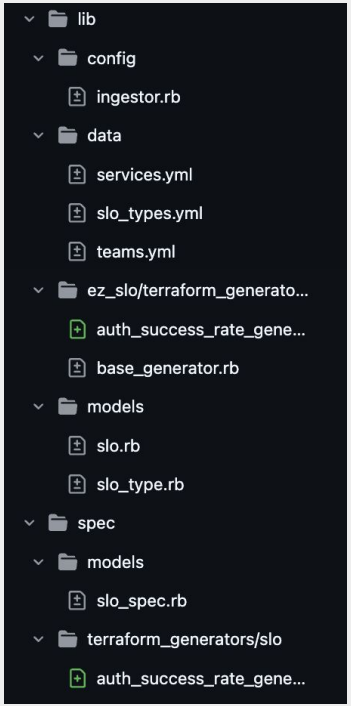


Origin Story

New SLOs



New Functionality



It’s a Compiler Problem

Frontend	Middle End	Backend





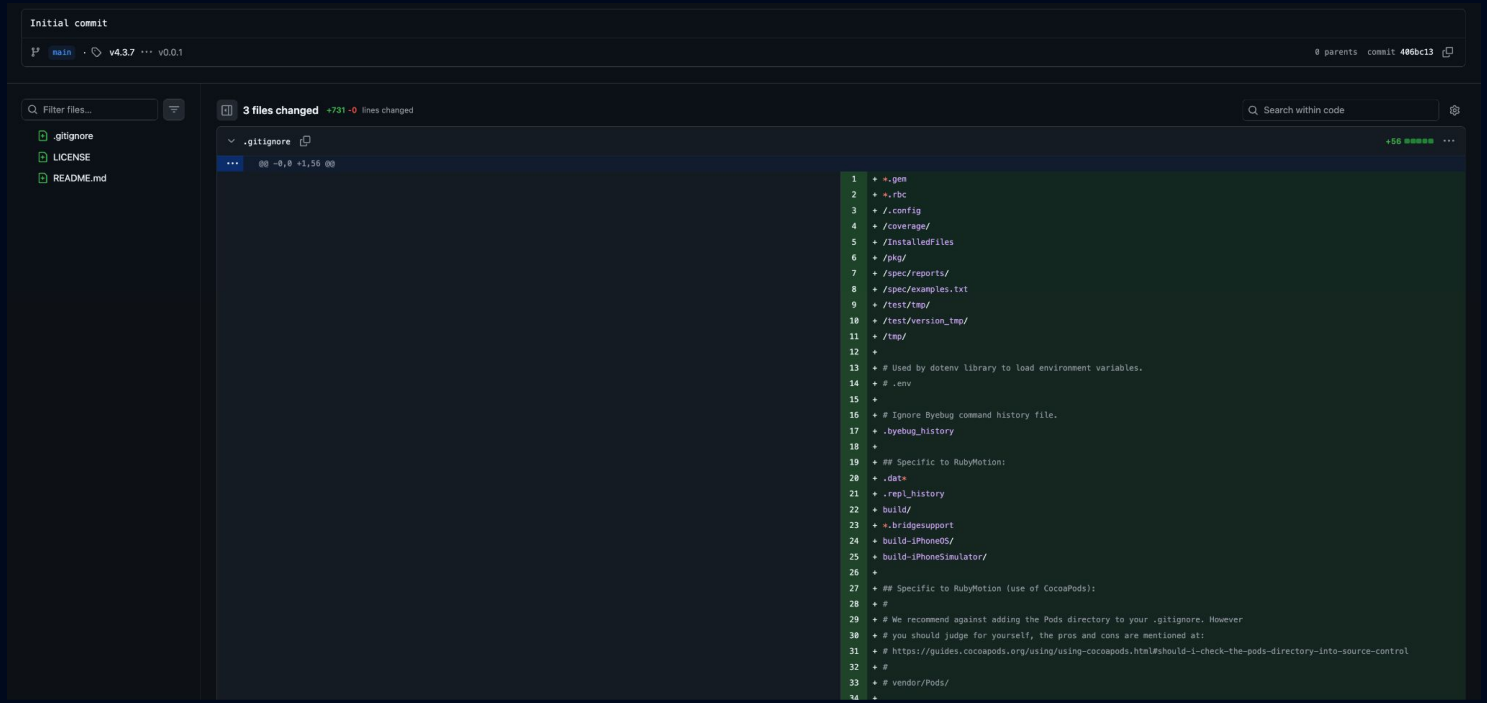



BRICKELL RESEARCH

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...
.gitignore
LICENSE
README.md
3 files changed +731 -0 lines changed
Search within code
.gitignore
+56
@@ -0,0 +1,56 @@
+ *.gem
+ *.rbc
+ /.config
+ /coverage/
+ /InstalledFiles
+ /pkg/
+ /spec/reports/
+ /spec/examples.txt
+ /test/tmp/
+ /test/version_tmp/
+ /tmp/
+
+ # Used by dotenv library to load environment variables.
+ # .env
+
+ # Ignore Byebug command history file.
+ .byebug_history
+
+ ## Specific to RubyMotion:
+ .dat*
+ .repl_history
+ build/
+ *.bridgesupport
+ build-iPhoneOS/
+ build-iPhoneSimulator/
+
+ ## Specific to RubyMotion (use of CocoaPods):
+ #
+ # We recommend against adding the Pods directory to your .gitignore. However
+ # you should judge for yourself, the pros and cons are mentioned at:
+ # https://guides.cocoapods.org/using/using-cocoapods.html#should-i-check-the-pods-directory-into-source-control
+ #
+ # vendor/Pods/

BRICKELL RESEARCH

What Language?

Sorbet - Ruby + Gradual Typing

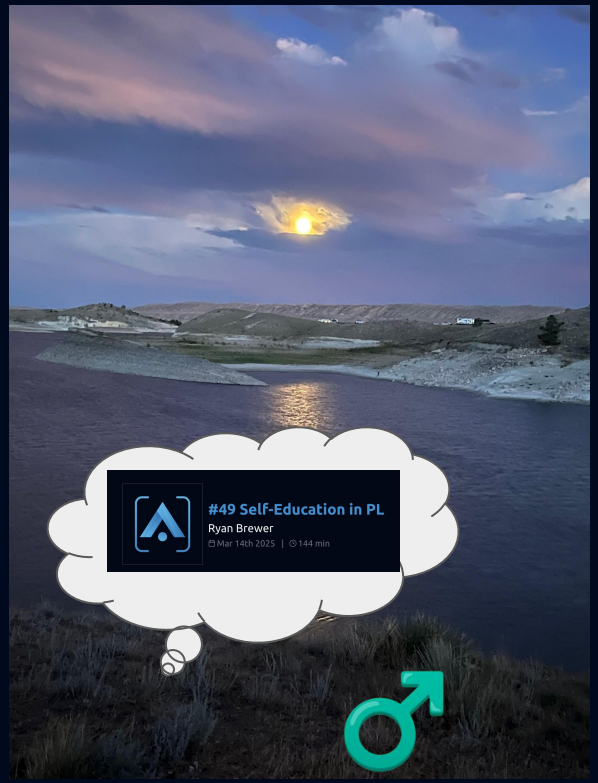
Typed Racket

Rust

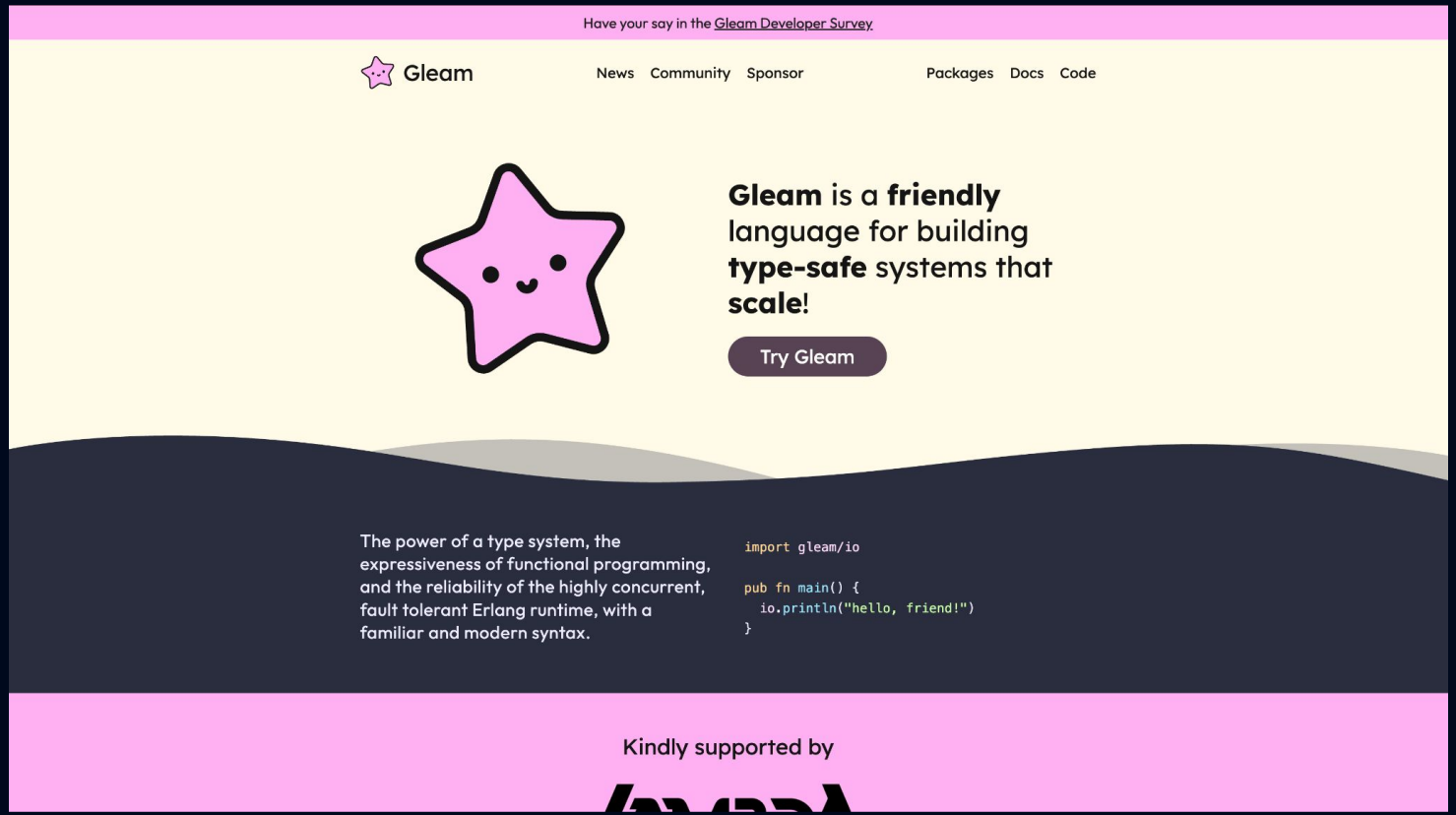
Haskell

Go

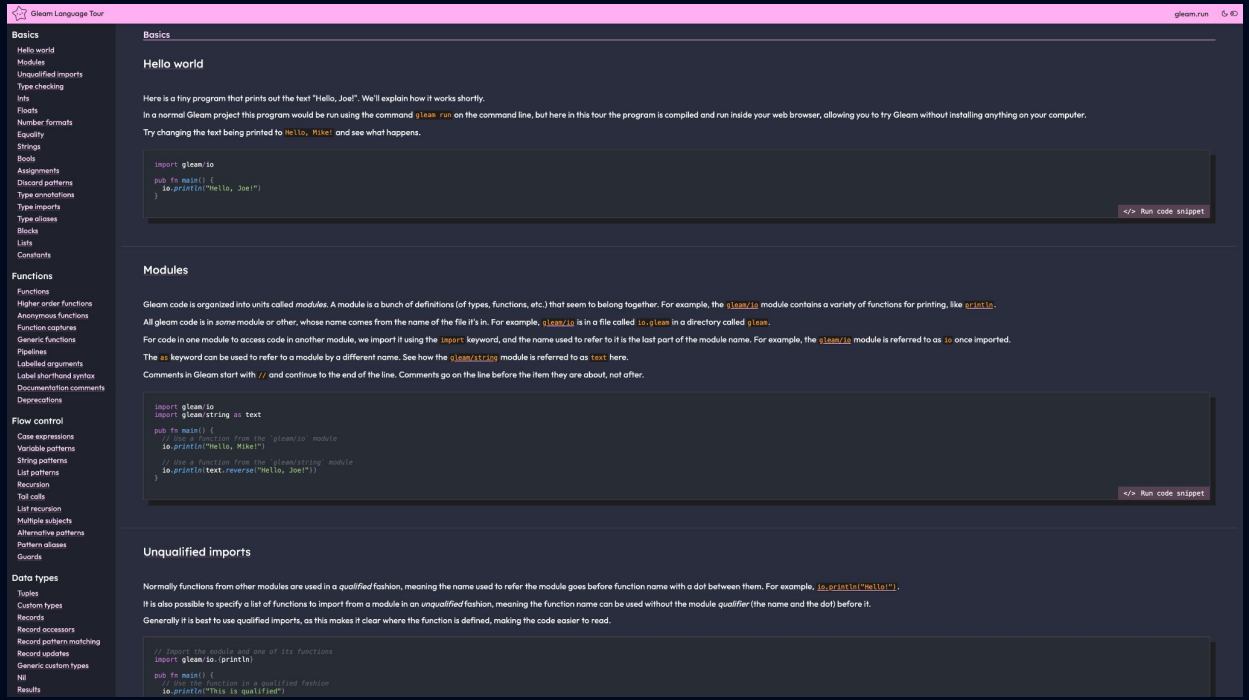
Gleam?



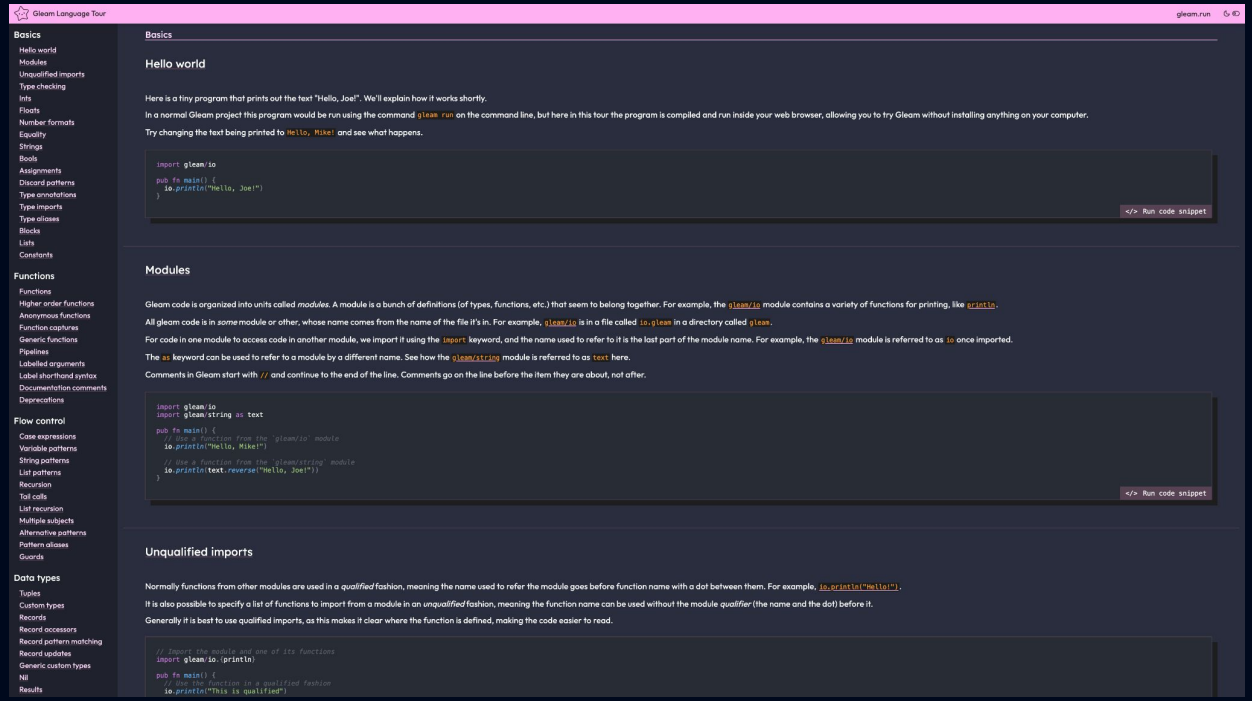
Gleam.


Have your say in the Gleam Developer Survey
Gleam
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

Gleam!

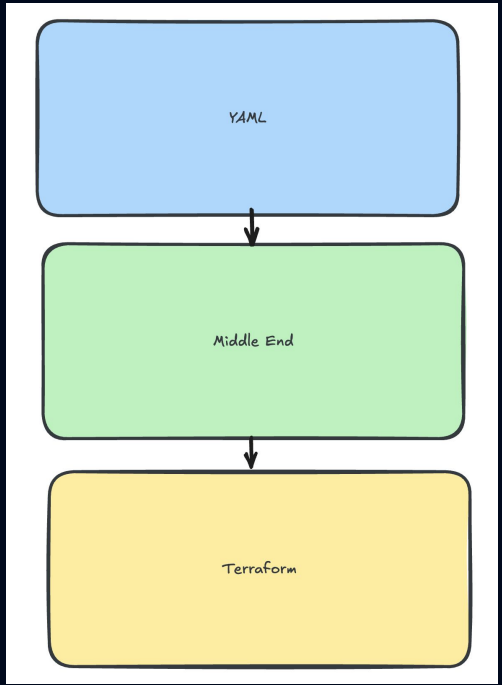

Gleam Language Tour
gleam.run
Basics
Hello world
Modules
Unqualified imports
Type checking
Ints
Floats
Number formats
Equality
Strings
Bools
Assignments
Discard patterns
Type annotations
Type imports
Type aliases
Blocks
Lists
Constants
Functions
Functions
Higher order functions
Anonymous functions
Function captures
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, Joe!". 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, Joe!")
}
</> 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, Joe!"))
}
</> 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 and one of its functions
import gleam/io.{println}
pub fn main() {
// Use the function in a qualified fashion
io.println("This is qualified")

Gleam!



[Approachability]: *Gleam is intentionally small!*

One Week to Parity



YAML

glaml

hex v3.0.2 hex docs

Glaml is a simple Gleam wrapper around yamerl 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.

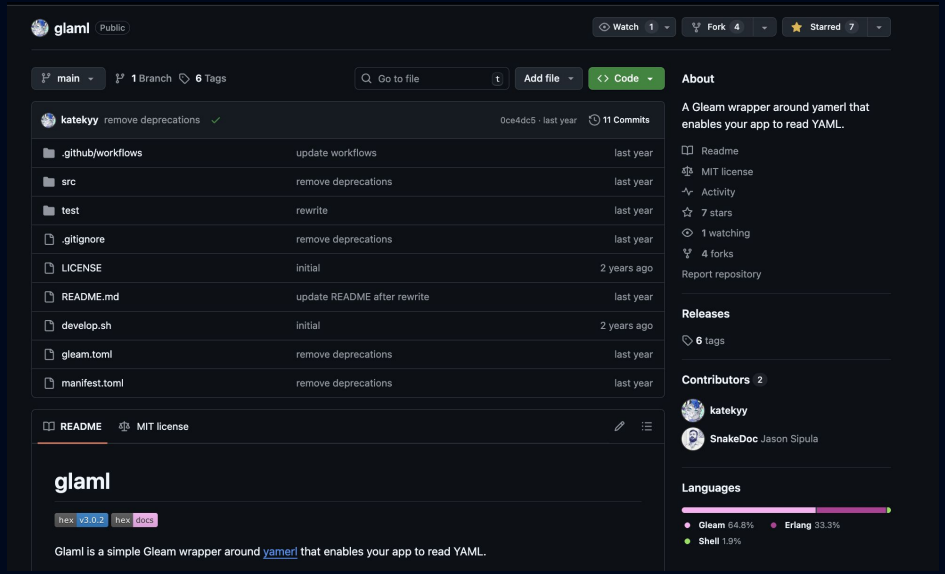
Instalation

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
```



YAML

glaml

hex v3.0.2 hex docs

Glaml is a simple Gleam wrapper around yamerl 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.

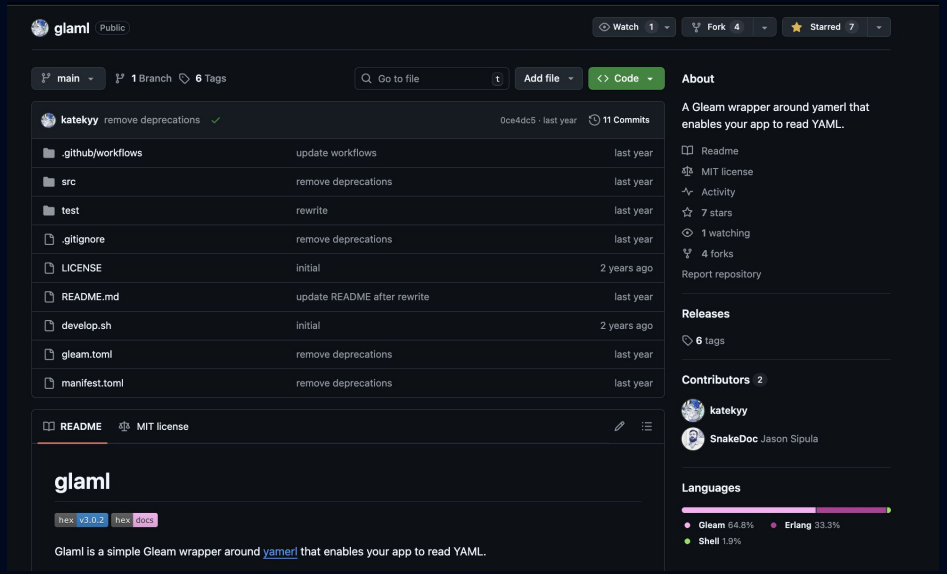
Instalation

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

```
gleam add glaml
```

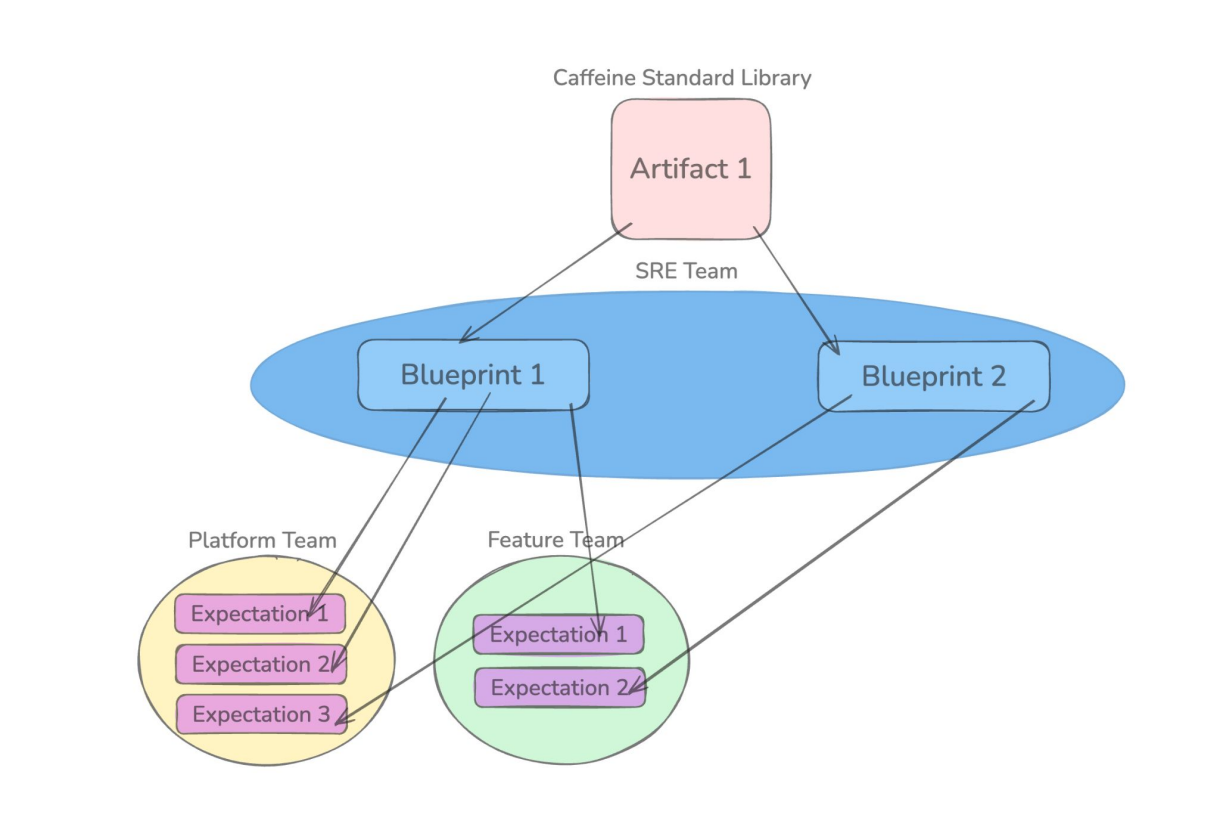
Development

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



[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
```

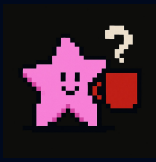
Result

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

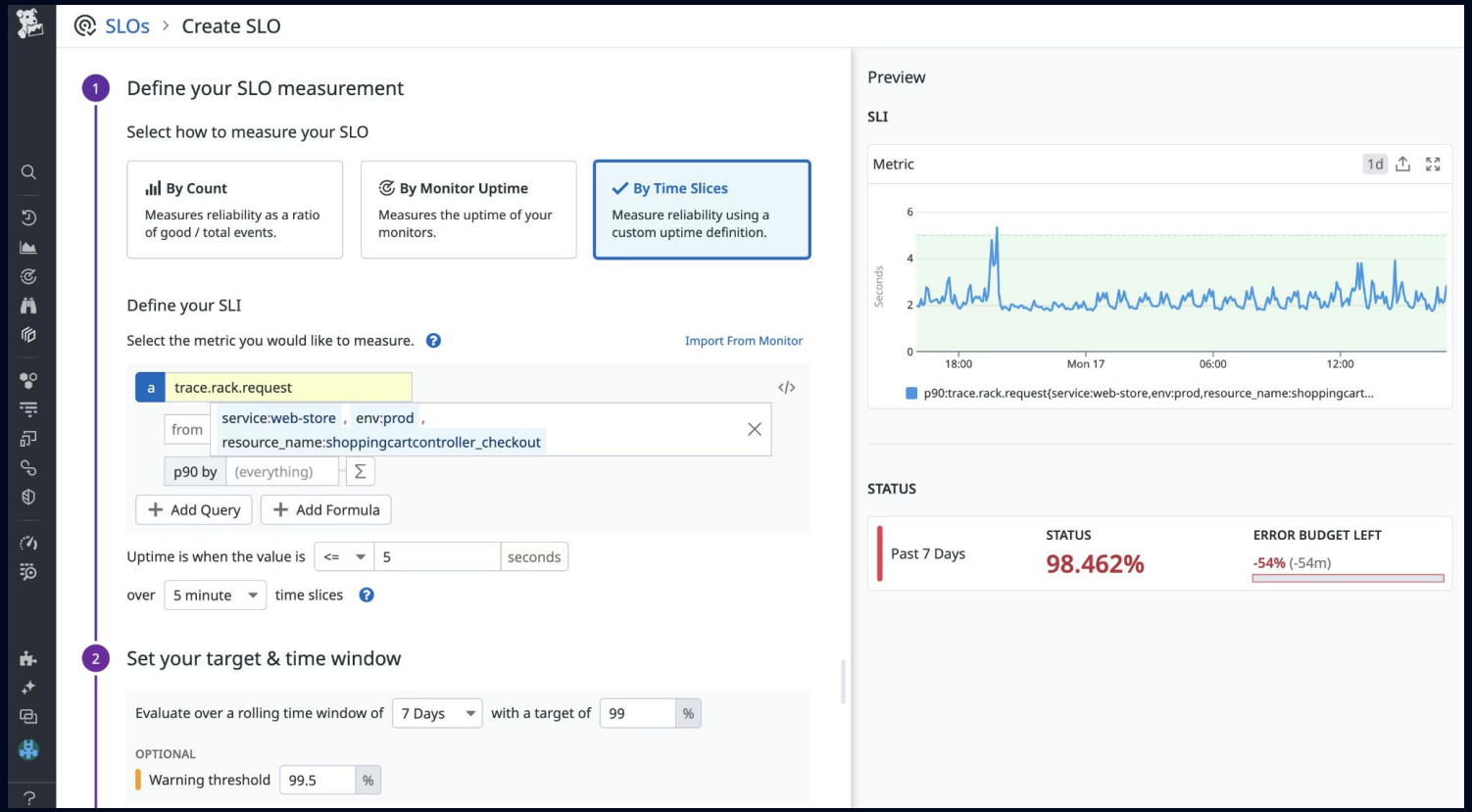
Threshold: 99.9%

Signals:

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

Evaluation: “good / total”

How Caffeine Works



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.

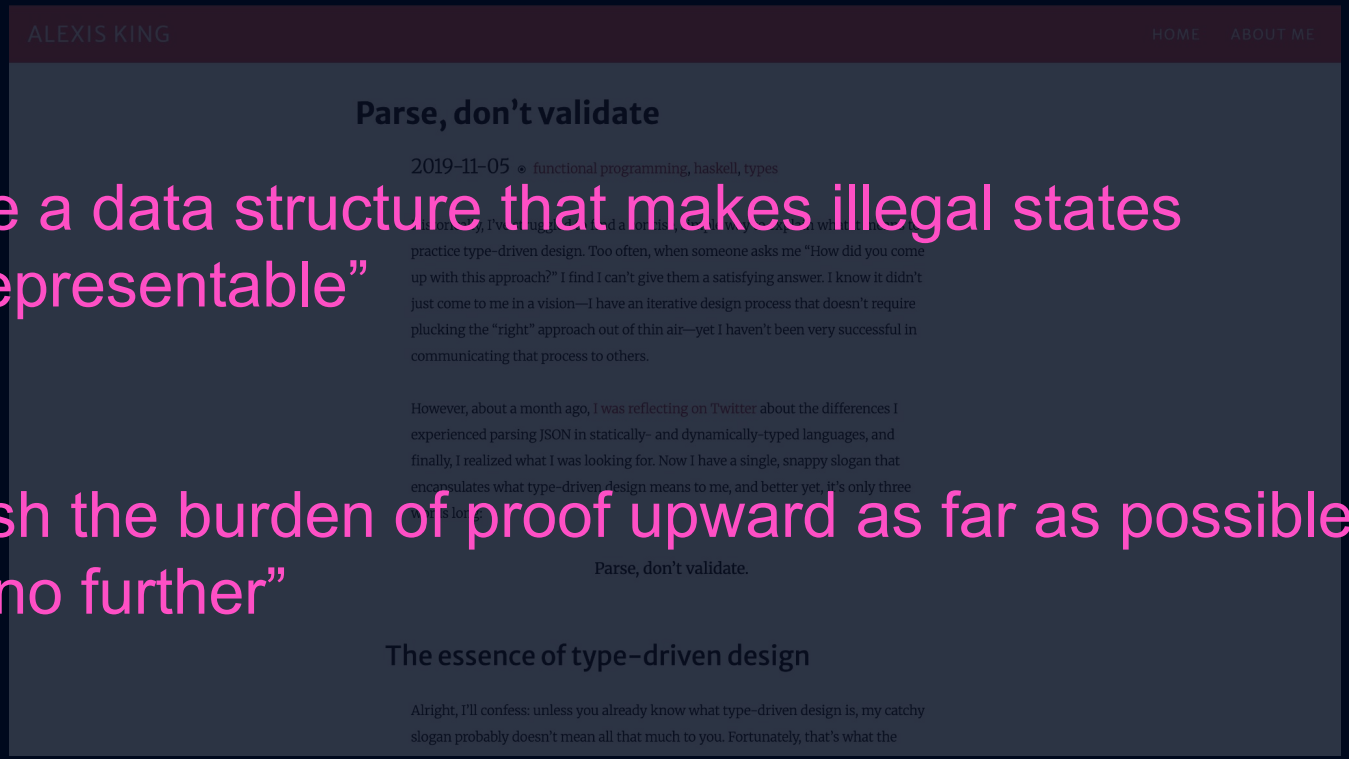
However, about a month ago, I was reflecting on Twitter about the differences I experienced parsing JSON in statically- and dynamically-typed languages, and finally, I realized what I was looking for. Now I have a single, snappy slogan that encapsulates what type-driven design means to me, and better 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



“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 〓 Bool
- Integer 〓 Int
- Float 〓 Float
- String 〓 String

Collection

- List(T) 〓 List(T)
- Dict(String, T) 〓 Dict(String, T)

Structured

- Record({String, T}) 〓 Dict(String, T)

Modifier

- Optional(T) 〓 T
- Defaulted(T, x) 〓 T

  where x is a value of type T

Refinement

- OneOf(T, {xs}) 〓 T

  where T is Integer, Float, or String and xs is a set of member values all of type T
- InclusiveRange(T, (l:h)) 〓 T

  where T is Integer or Float and l and h are both of type T

Caffeine Type 〓 Gleam Type

Extraction Pattern

Specification

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

Compiler

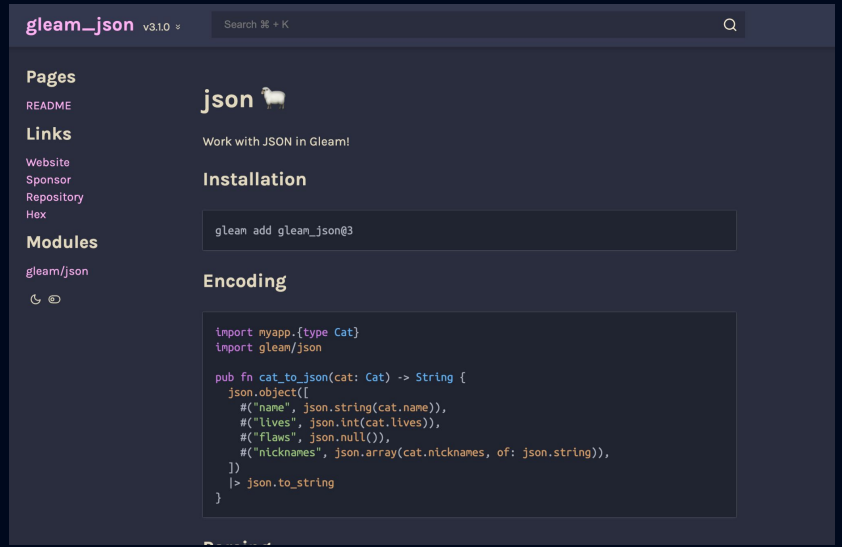
```
1   // Extract the actual float value using pattern matching
2   let threshold_value = try_extract_float(threshold_node)
3
4   fn try_extract_float(node: glaml.Node) -> Float {
5     case node {
6       glaml.NodeFloat(value) -> value
7       _ -> panic as "Expected node to be a float"
8     }
9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

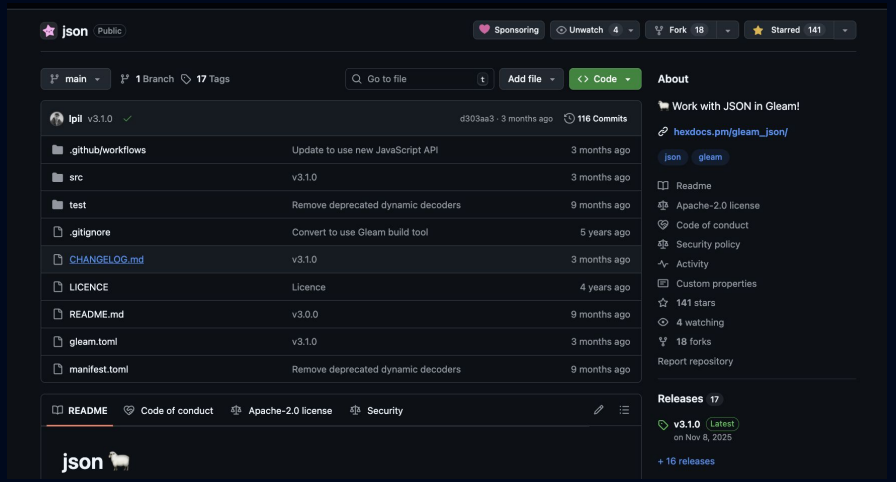
YAY - *“yet another YAML”*

- A subset of all extractors…

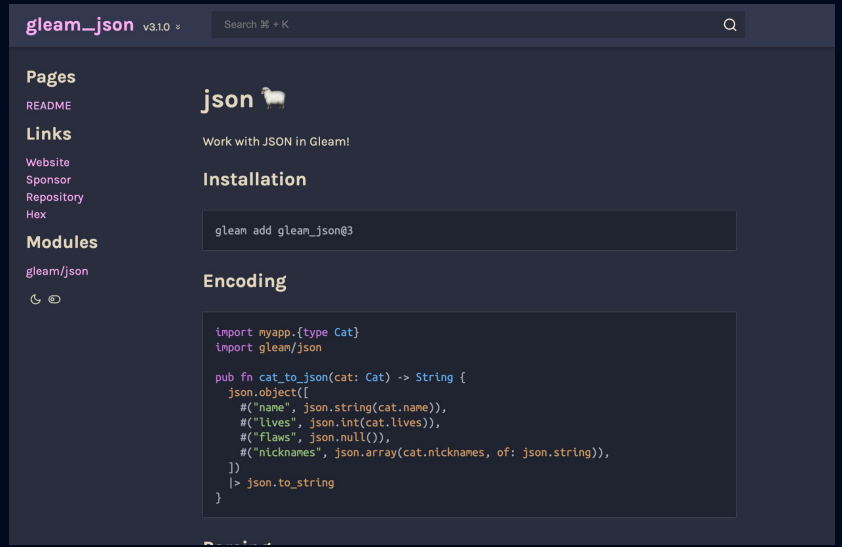
Pattern	String	Int	Float	Bool
Required	extract_string	extract_int	extract_float	extract_bool
Optional	extract_optional_string	extract_optional_int	extract_optional_float	extract_optional_bool
With default	extract_string_or	extract_int_or	extract_float_or	extract_bool_or
List of	extract_string_list	extract_int_list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

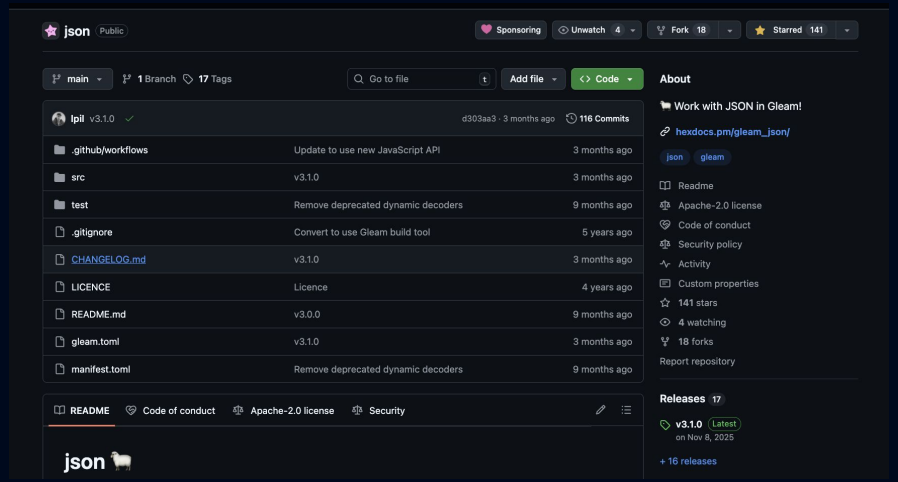
JSON?


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
lpil v3.1.0
d303aa3 · 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

JSON?





[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!")
```

BRICKELL RESEARCH

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`.

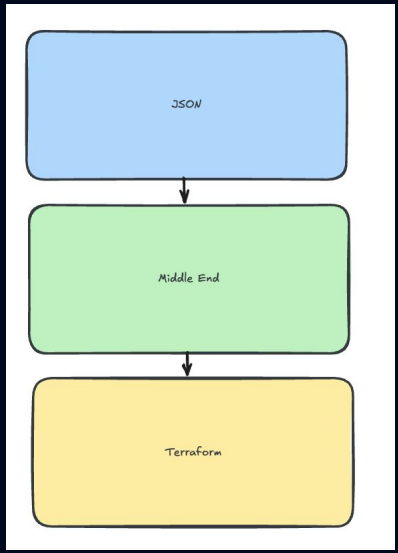
```
// Data:
// "Hello, Joe!"

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

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

A Makeover (sort of)

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

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



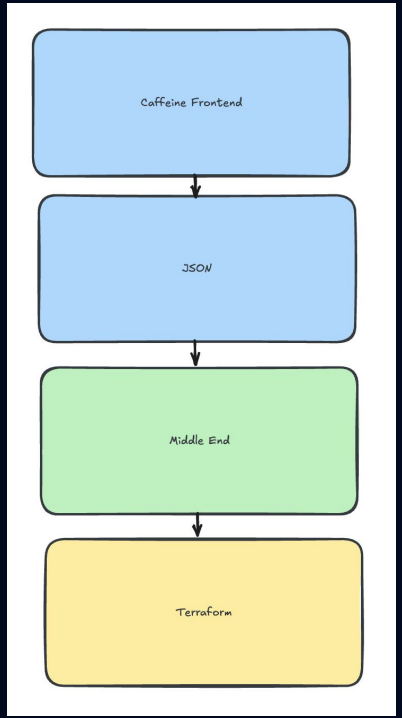
A Glow Up

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

```gleam
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)
  })
}
```

```gleam
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

```gleam
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,
  ))
}
```

BRICKELL RESEARCH

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*

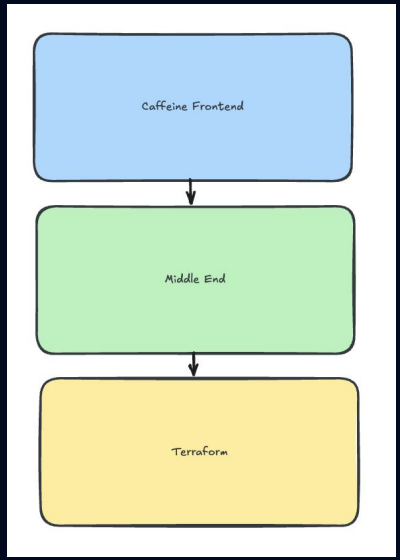
Own Frontend

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

```gleam
/// 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*

BRICKELL RESEARCH

Full Example

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

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

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

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

BRICKELL RESEARCH

Shipping Gleam - Problem

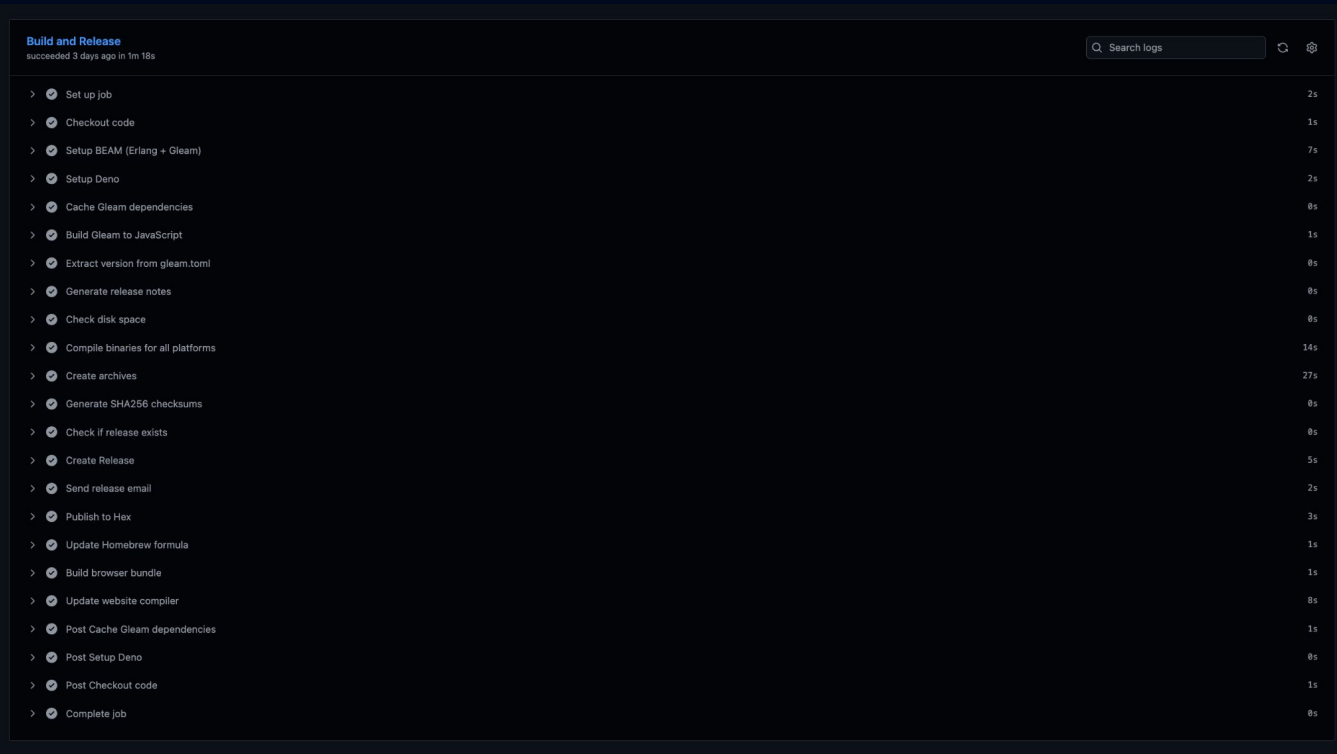
- gleam export erlang-shipment
- gleescript
- Docker
- Nix

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!

Distributing Caffeine - Full Pipeline



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

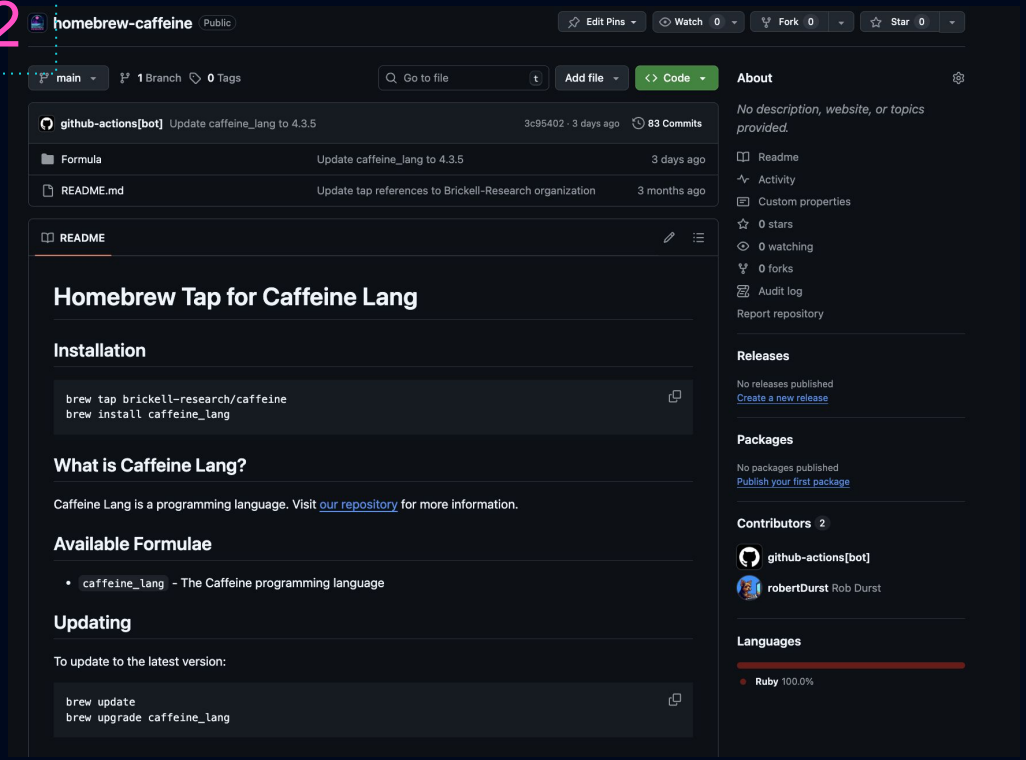
- Harden LSP

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

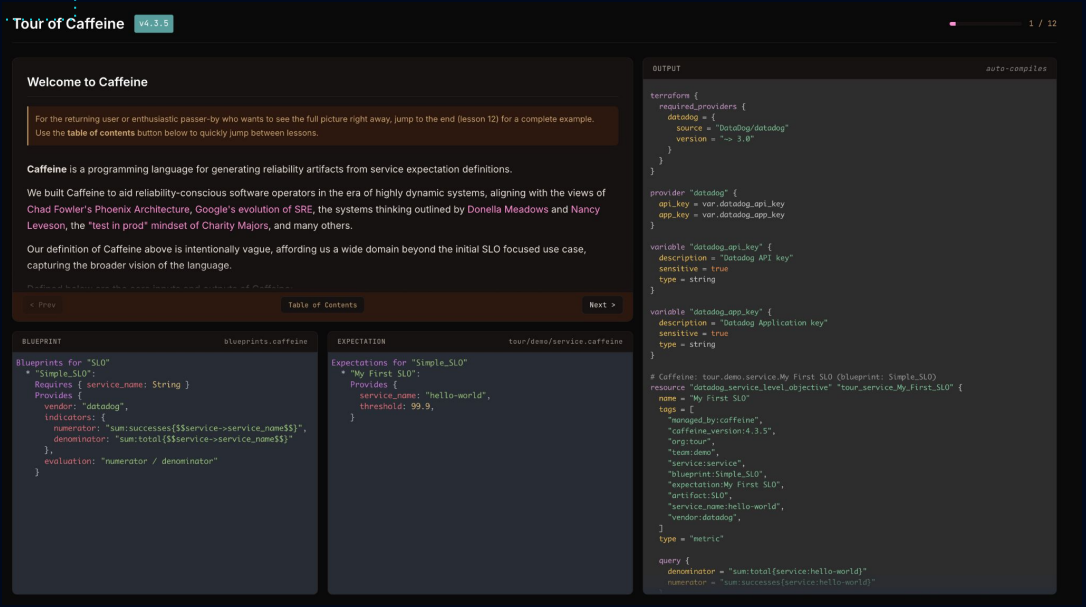
Distributing Caffeine - Full Pipeline

2



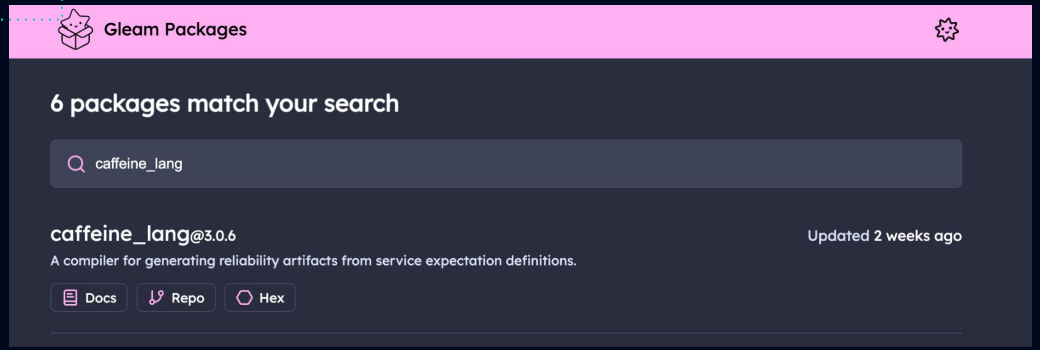
Distributing Caffeine - Full Pipeline

3



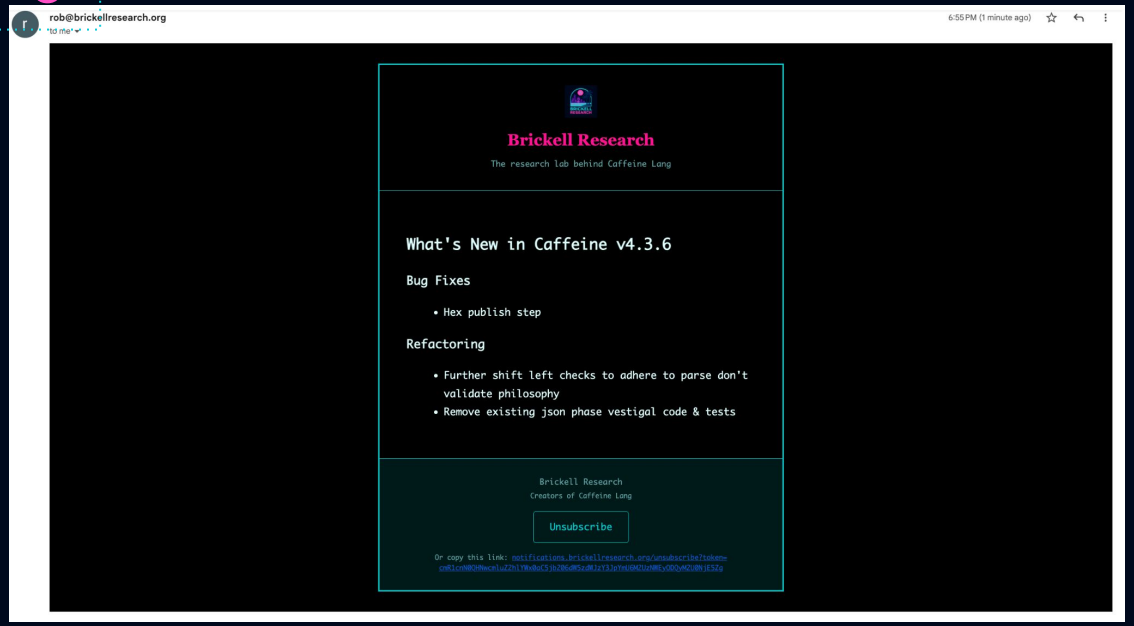
Distributing Caffeine - Full Pipeline

4

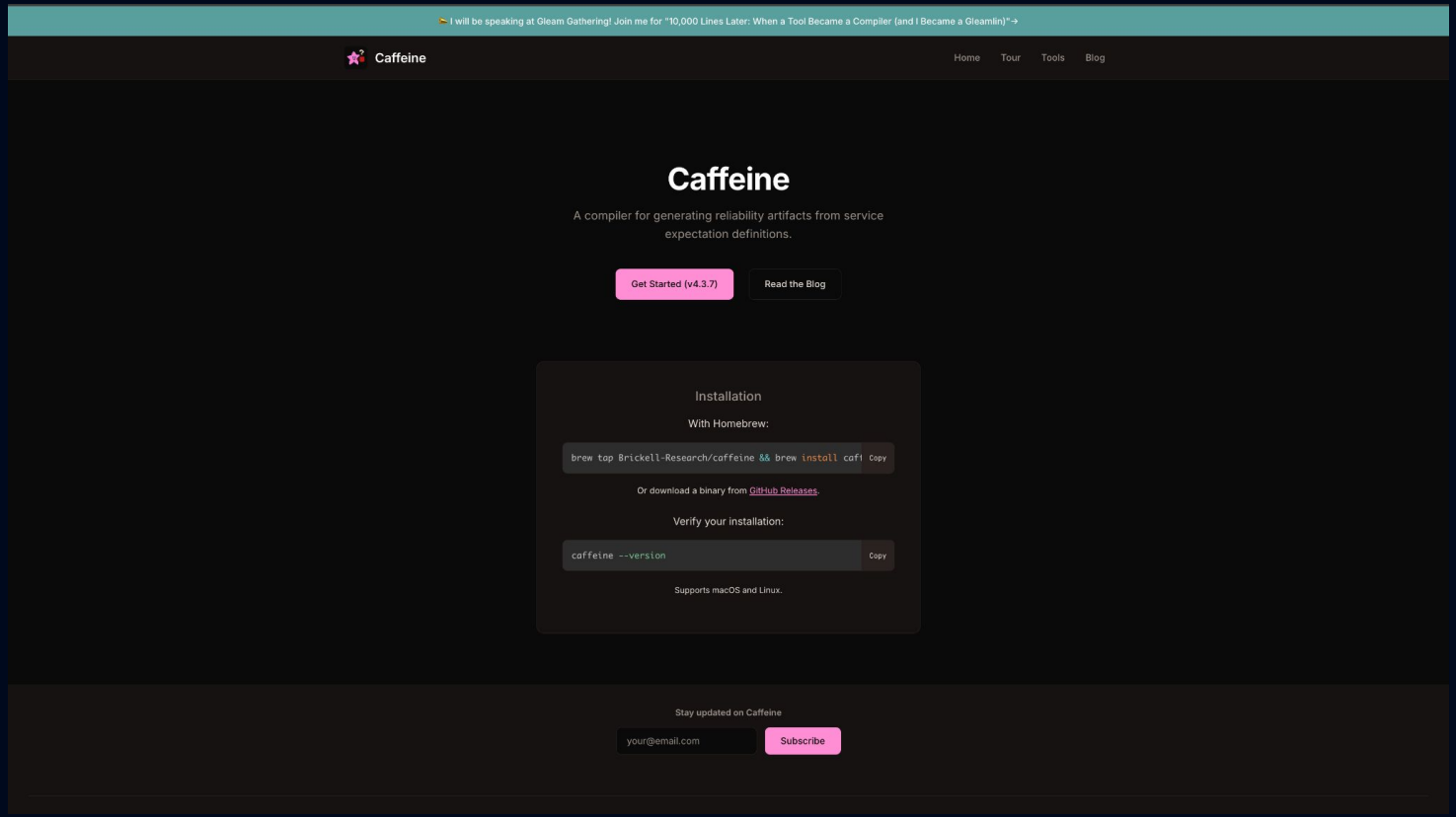


Distributing Caffeine - Full Pipeline

5



We Have a Programming Language!

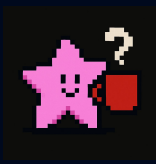


BRICKELL RESEARCH

Gleam is…

1. [Approachable] Small enough to learn

Gleam is…



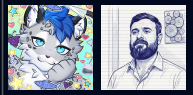
1. [Approachable] Small enough to learn
2. [Micropatterns] Functional enough to model a compiler

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



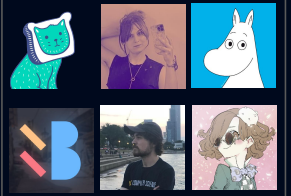
glint

filepath

argv

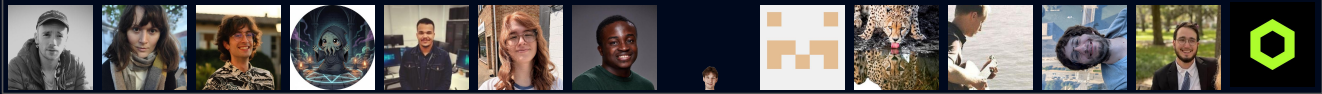
envoy

ansi



stdlib

simplifile



json

gleeunit

We're Brickell Research and we ❤️ Gleam

🏠 caffeine-lang.run
🧑‍💻 github.com/Brickell-Research/caffeine_lang
✉️ rob@brickellresearch.org