Common Expression Language in Rust

Alex Snaps, Red Hat

What's CEL ?

Common Expression Language aka CEL

- Fast
- Portable
- Extensible
- Safe

Came out of Google ~2017

Used embedded in "another program" written in

- Go
- Java
- C++
- but also JS, ... **Rust** as community efforts

CEL

Strongly typed

Core types

- String, Int, Double, Boolean
- UInt, Bytes, Timestamp, duration
- Map, List

Protocol Buffer natively supported

- Struct

JSON interop

Spec'ed

With a conformance test suite

A familiar syntax

- CEL's Syntax

```
// Check whether a resource name starts with a group name.
resource.name.startsWith("/groups/" + auth.claims.group)

// Determine whether the request is in the permitted time window.
request.time - resource.age < duration("24h")

// Check whether all resource names in a list match a given
filter.
auth.claims.email_verified
   && resources.all(r, r.startsWith(auth.claims.email))
```

CEL optional syntax

```
values[?2].?field.orValue(0) > 100
```

How we got started with CEL

Use case(s)

Kubernetes

- Validation rules without webhooks

Kuadrant

- Predicates and other data retrieval expressions
- Evaluated on the data plane
- Across multiple runtimes
  - Go and...
  - Rust, possibly within a Wasm runtime

Using CEL in Rust

```
cargo add cel
```

Using CEL in Rust

```
let program = Program::compile("1 == 1").unwrap();
let context = Context::default();

let value = program.execute(&context).unwrap();
assert_eq!(value, true.into());
```

Using CEL in Rust

```rust
let program = Program::compile("foo * 2").unwrap();
let mut context = Context::default();
context.add_variable("foo", 10).unwrap();
let value = program.execute(&context).unwrap();
assert_eq!(value, 20.into());
```

Using CEL in Rust

```rust
let program = Program::compile("add(10, 10)").unwrap();
let mut context = Context::default();
context.add_function("add", |a: i64, b: i64| a + b);
let value = program.execute(&context).unwrap();
assert_eq!(value, 20.into());
```

Using CEL in Rust

```
context.add_function("isEmpty", is_empty);

fn is_empty(This(s): This<Arc<String>>) -> bool {
    s.is_empty()
}
```

```
myString.isEmpty()
```

Using CEL in Rust

```rust
fn fail(ftx: &FunctionContext) -> ResolveResult {
    ftx.error("This function always fails").into()
}
```

```rust
pub struct FunctionContext<'context, 'call: 'context> {
    pub name: &'call str,
    pub this: Option<Value>,
    pub ptx: &'context Context<'context>,
    pub args: &'call [Expression],
    pub arg_idx: usize,
}

pub type ResolveResult = Result<Value, ExecutionError>;
```

Alex, this cel-rust. cel-rust, this is Alex!

- 2024
  - CEL Rust introduced in Kuadrant
  - First bug fixes
    - Added missing macros on `String`, `Timestamp`, ...
    - Fixed some function dispatch code
- 2025
  - Let's fix that parser impedence mismatch
    - Revamp `antlr-rust`
    - Rewrite the entire parser
    - Adapt the interpreter to this new AST

What? Wait... Why?!

Remember, what's CEL ?

Common Expression Language aka CEL

- Fast
- Portable
- Extensible
- Safe

Remember, what's CEL ?

Common Expression Language aka CEL

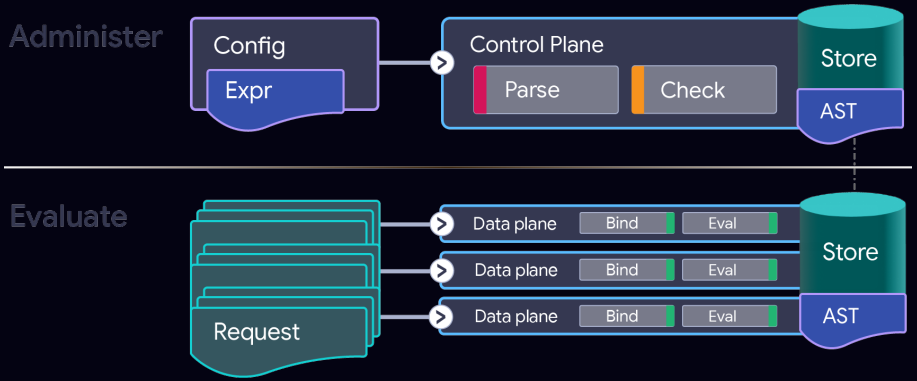
- Fast
- Portable
- ~~Extensible~~
- Safe

Much of which is actually achieved through the AST

Let's start with

FAST!

A day in the life of a CEL expression



Informed by the golang implementation

Just reuse the official `CEL.g4` ANTLR4 grammar

... tho ANTLR has no rust target

antlr-rust project existed

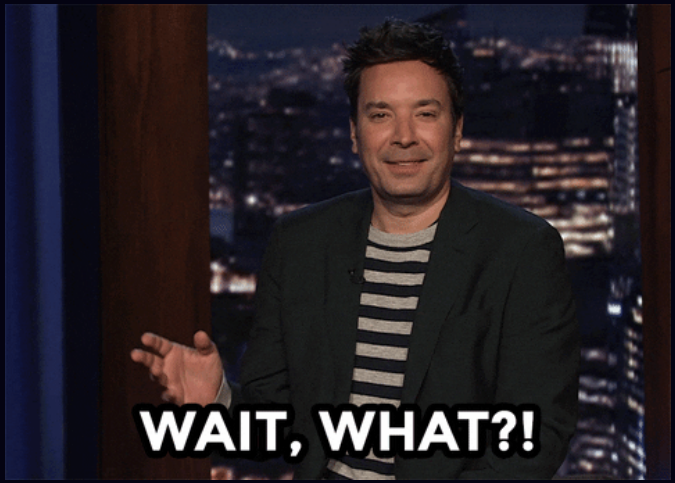
... but abandonned and had to fork to fix and improve to work at least support CEL

Community ftw! We now have an `antlr4rust` working Rust target for ANTLR4

With a conformant parser, we are good now, yes?

- Holidays 2025
- Opened up the interpreter to be type agnostic
- Just a quick refactoring...
- ... of a +2,805 −555 PR!
- Cow<dyn Val> all the things

Do you even know what you are doing?

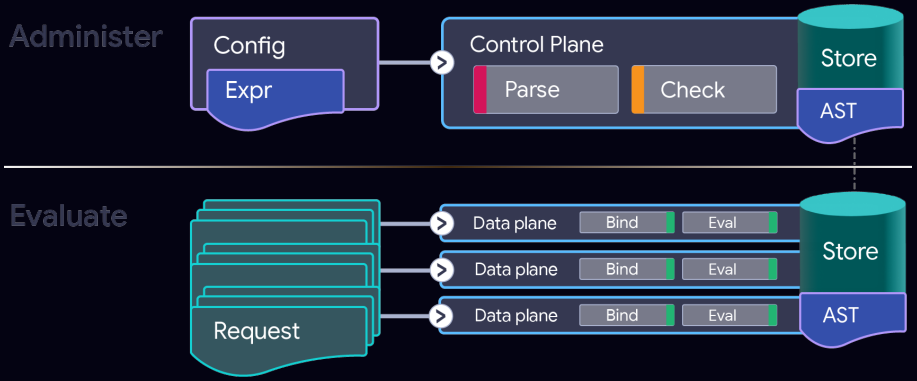


Overloads

```
fn main() {
  let program = Program::compile("[1, 2].size() == 2.size()").
unwrap();
  let mut context = Context::default();
  context.add_function("size", my_size);
  let value = program.execute(&context);
  assert_eq!(value, Ok(true.into()));
}

pub fn my_size(_ftx: &FunctionContext, This(this): This<i64>)
    -> Result<i64, ExecutionError>
{
  Ok(this)
}
```

Confidence in being able to evaluate always



The `check` phase

```
let program = Program::compile("[1, 2].size() == 2.size()").unwrap
();
let value = program.execute(&Context::default());
```

```
let parser: Parser = Parser::default();
let result: Result<IdedExpr, ParseErrors> = parser
    .parse("[1, 2].size() == 2.size()");
let expr = result.unwrap();
let context = Context::default();
let value: ResolveResult = Value::resolve(&expr, &context);
```

```
let result: Result<CheckedAst, Error> = check(expr, &context);
let value = Value::resolve(&result.unwrap().expr, &context);
```

The road ahead

Roadmap

Current

Wrap up the overloads

Next

Type checking

And this too btw

Performance work

... and this too

Protobuf & WKT support

~~Legal disclaimer~~ mentions

▓▓ None of this would exist without these people

▓▓ github.com/cel-rust

- @clarkmcc // Clark McCauley
- @orf // Tom Forbes
- @adam-cattermole // Adam Cattermole
- @howardjohn // John Howard
- ... and the many contributors

▓▓ github.com/antlr4rust

- @rrevenantt // Konstantin
- @kaby76 // Ken Domino
- @cyqw

▓▓ Special thanks for this presentation to

- @mfontanini from https://github.com/mfontanini/presenterm

Thanks!

Links

https://github.com/cel-rust/cel-rust

https://kuadrant.io

Questions?