Crystal is a Surprise

Adrian Kosmaczewski

2022-07-15

I blogged about Dart a few weeks ago, and I said it was refreshingly boring. I am probably late to the party (well, evidently I am, as it's been around since 2014), but I discovered Crystal recently, and it is not only boring but also surprising in many delicious ways.

(And yes, I like boring technology. A lot.)

If I say that Crystal is "compiled Ruby" I'm oversimplifying things. Well, ahem, am I? At last year's Crystal Conference, celebrating the release of Crystal 1.0, the keynote speaker was none other than Yukihiro 'Matz' Matsumoto himself. (If you don't know who he is, he invented Ruby.)

Beyond the syntactic similarities, Crystal is quite an improvement over Ruby. It includes many modern features, such as: null safety, a package dependency system, a build tool, type inference, garbage collection, a robust type system, a built-in preprocessing macro system, generics, Go-like fibers and channels for multiprocessing, and a lightning-fast compiler (based on the LLVM toolchain) that outputs lightning-fast code. All very modern.

Another nice touch is that the compiler can generate statically linked binaries not needing anything else to run; this feature makes Crystal a great candidate for writing containerized ("Dockerized") applications.

And finally, the thing that surprised me the most: it comes from my childhood neighborhood. The company behind it has its headquarters merely 10 blocks away from the location where I grew up as a kid, in Vicente López, north of the city of Buenos Aires. Call me nostalgic, but it struck a chord.

Fortune

As usual, I created yet another example of the Fortune application, available for your exploration in GitLab. I used the Kemal framework, and to be honest, I quite literally copied the source code of the Ruby version, changed a few things, and voilà, done.

From a syntactic point of view, Crystal is uncannily similar to Ruby. Let's compare both "Fortune" apps written with them, starting with Crystal.

```
require "kemal"

version = "1.2-crystal"
hostname = `hostname`

get "/" do |env|
  message = `fortune`
  number = rand(1000)
  accept = env.request.headers["Accept"]
  if accept == "application/json"
    obj = { :version => version,
            :hostname => hostname,
            :message => message,
            :number => number }
    obj.to_json
  elsif accept == "text/plain"
    "Fortune %s cookie of the day #%d:\n\n%s" % [version, number, message]
  else
    render "src/views/fortune.ecr"
  end
end

Kemal.config.port = 8080
Kemal.run
```

Look at the Ruby version for comparison:

```
require "sinatra"
require "sinatra/reloader" if development?

set :port, 8080

get '/' do
  @version = "1.2-ruby"
  @hostname = `hostname`
  @message = `fortune`
  @number = rand(1000)
  if request.accept? 'text/html'
    erb :fortune
  elsif request.accept? 'application/json'
    obj = { :version => @version,
            :hostname => @hostname,
            :message => @message,
            :number => @number }
    JSON.generate(obj)
  elsif request.accept? 'text/plain'
    "Fortune %s cookie of the day #%d:\n\n%s" % [@version, @number, @message]
```

```
  end
end
```

Seriously… what!?

But the biggest surprise, aside from the syntax, was the result of the web framework comparison I presented in this blog post. Based on energy ranking, memory required at runtime, number of lines of code, build time, and size of the final container image… **the big winner in the ranking is Crystal!** I'm planning an update to this ranking including other criteria, such as ecosystem size, average latencies, and other important points. But this clearly was an unexpected result, and it caught me completely off-guard.

Crystal compiles quickly into small and fast binaries. What's not to love?

Conway

I also created the Crystal version of the Conway app, and again, I quite literally copied the code of the Ruby version, and fixed the few following things until the app worked:

- Use `require` instead of `require_relative`;
- Implement the `hash(hasher)` method in the `Coord` class, to be able to use it as a `Hash` key;
- Added type information in some method signatures and when instantiating generic collections;
- Replace `attr_reader` with `getter`;
- Implement `to_s(io : IO)` for performance;
- Change the handling of `SIGINT` signals in the main app;
- Replace `for in range` loops with `(0...bound).each`;
- Override `==()` instead of `eq?()` for equality.

And I promise, that was it. It must have taken me no more than 40 minutes until it worked. Many of the changes enumerated above have to do with the fact that Crystal precluded some dynamic aspects of the Ruby runtime for performance reasons. A very wise choice.

The only caveat I could detect in the developer experience was that the VSCode extension for Crystal does not (yet) allow for debugging of Crystal apps. This is quite a bit of a pain point, I must admit. But not a big showstopper.

All in all, I've been pleasantly surprised by Crystal: it's a very good idea with a brilliant implementation. It is definitely a potent tool with a great future, and it has a vibrant ecosystem.