Anton Lavrik

From 10s to 1000s of engineers:
Scaling Erlang Developer Experience at WhatsApp

Code BEAM STO V, September 2020
Bio

Erlang user since 2007

Built server applications in Erlang

7 years at WhatsApp

Leading WhatsApp Erlang team:

our mission: improve Erlang to make developers more productive
Erlang at WhatsApp: 11 years of success

“best engineering decision we ever made”

Objectives

Highlight Erlang strengths: what works well

Describe our challenge of scaling Erlang DevX to larger teams and codebases

Discuss Erlang limitations and how to address them:

- static typing
- namespaces
- tools: build system, formatter, IDE integration
Erlang strength: very efficient architecture

Highly available, more reliable than ever

Core design hasn’t changed in 8 years

  Leveraging BEAM: native lightweight processes, message passing, distribution, and share-nothing memory model

Scaled extremely well:

  2B+ users, multi-data centers, containers

  enabled multiple product features
Erlang strength: empowers smaller teams

For example, WhatsApp scaled to 900M users with 50 engineers*

Extremely fast development cycle:

- High-level declarative language
- Fast compilation
- Fast deploys: several minutes via hotload – stateless, stateful, any # of servers

What changed with growth

During startup years: move as fast as possible with a small team

- **Critical:** speed of iteration of 1-2 person teams, reliability of service
- **Less important:** code conventions, tool choice, tests, documentation

Today: rapidly growing teams, codebase, requirements

- **Critical:** throughput of much larger teams and orgs, reliability of service
- **Now important:** code conventions, tool choice, tests, documentation

*Key questions: How to provide fast development cycle with many more engineers? How important this is?*
What we learned at Facebook

Developer productivity for larger teams becomes critical, not merely important.

Things that help optimize development cycle matter a lot.

We have reached this phase with Erlang at WhatsApp.

Example: Hack, statically typed dialect of PHP, shows typechecker errors in the IDE interactively for 1000s of Facebook engineers since 2014*

IDE with typechecker: example from Hack

https://www.wired.com/2014/03/facebook-hack/
Modern requirements for language DevX

- make code easy to navigate and understand
- make incremental changes reliably and efficiently
- refactor code reliably and efficiently
- promote well-structured APIs
- enable fast change-test-review-deploy workflows

A must for larger teams. But useful at any scale.

These were not requirements for Erlang when it was designed in the 90s!
Some Erlang DevX limitations*

no static typing

flat namespace for records and modules

lack of well integrated tooling: IDE integration, formatter, build system, …

*compared to modern languages built with these requirements in mind: Go, TypeScript, Kotlin
Trends in languages

shift to modern languages with integrated tooling, e.g. Erlang competition:

C++, Java → Go, Rust, Kotlin

shift to static typing:

JavaScript → TypeScript
Python → MyPy
Ruby → Sorbet
PHP → Hack

Question: what would a modern Erlang with integrated tooling and static typing look like?
Static typing
Static typing goals

• High productivity for teams of any sizes
• Feasible adoption for Erlang users, teams, codebases

We are working on a prototype, open-sourcing in November
Static typing: high usability

Fast signal from typechecker

Easy to understand error messages

Integrated with language, compiler and IDE
Good error messages: example from Elm

Static typing: reliable signal

Strong guarantees (aka soundness)

Consistency between code and specs

Errors on missing clauses in pattern matching
Missing clauses: example from Elm

Why static typing guarantees matter

reliable signal on incremental changes
reliable signal for refactoring
less tests and defensive is_* guards
Static typing as “better Dialyzer”

highly usable

something you can rely on

doesn’t create friction
API consistency example: option type

Wouldn’t it be great to have one way to do it?

% dynamically typed Erlang

undefined | T
false  | {value, T}
false  | T
null   | T
nil    | T

% Erlang with static typing

-enum option(T) ::
  some {T} | undefined.
Modularity: making opaque truly opaque

static typing answers these questions:

• how to make sure that in a large evolving codebase nobody relies on handle() being a pid()

• how to reliably make something opaque
Declarative APIs for gen_servers

gen_server is a common building block

but! it is too loose:
   can’t easily tell what the API is
   no guarantees things don’t break on API changes, especially for dist calls

plus, we need to make it statically typed

we are prototyping a declarative and statically typed gen_server API
Namespaces
Problem with flat namespaces: example

Problem: Interoperability with Thrift – IDL language with namespaces*

Input .thrift file:

realtime/signaling/client_delivery/if/SignalingDelivery.thrift

Output .hrl and one of the records:

-thrift_realtime_signaling_client_delivery_if_signaling_delivery.hrl
-record(signaling_delivery_signaling_delivery_service_deliver_message_to_client_result, …)

* Thrift is a cross-language RPC framework like Protocol Buffers / GRPC
Reasons for hierarchical namespaces

interop with other languages

structure larger codebases

enable larger open-source library ecosystems
Tools
Build system: rebar3

de-facto community standard

we are using it today

much faster and more robust than ever before!

**but:** it was not designed for larger projects and modern DevX
Build system: new requirements

- large codebases
- monorepos
- multi-language support

requirements for modern DevX:

- support for static typing compilation model
- need to be very fast to expose interactive workflows in IDE

separate solutions may be needed for these
Code formatter: erlfmt

Open-source and ready for wider adoption – give it a try! https://github.com/WhatsApp/erlfmt

Goals:

- automate formatting
- avoid style arguments in code reviews
The time has come! Thank you, Roberto and Erlang LS contributors. Consider joining the community!
Other exciting Erlang DevX projects at WhatsApp

testing Erlang code: ergonomic, scalable, and providing fast signal
running WhatsApp in a single BEAM instance
automated testing
automatic performance regressions detection

stay tuned for future talks!
Conclusion: Erlang DevX can be scaled

Everyone will benefit, not only large teams

Work on static typing is underway – more details later this year

Exciting times for the Erlang community!