Meet Bloop

Martin Duhem, Scala Center

GitHub: @Duhemm

Twitter: @mnduhem

Who am I?

- Martin Duhem
- Scala Center engineer since March 2017
- Likes tooling and compilers
- Projects
  - Macro stuff
  - sbt
  - Zinc
  - Scala Native
  - Dotty
  - Bloop

Agenda

1. What Bloop is (not)
2. Why Bloop is exciting for Scala developers
3. Why Bloop is exciting for tooling authors
4. What to look forward to in Bloop

What is Bloop?

What Bloop is not

First, let's first make it clear that...

- Bloop is **not** a build tool!
- Bloop **does not** resolve your dependencies.
- Bloop **does not** generate sources.
- Bloop **does not** bundle your application.

Rationale for Bloop

- *"Do one thing well instead of trying to do everything"* -- Jon Pretty
- Why did we invest in Bloop?
- Migrating to new tools is difficult
    - Difficult to learn
    - Difficult to get used to
    - Migration requires to invest a lot of time

Rationale for Bloop

With this in mind, it’s important to note that we’ve created bloop to primarily support all the existing build tools that companies and Scala open-source developers use today.

- We are better off integrating with existing tools.
- That doesn't mean that we can't bring improvements to Scala developers.

What is Bloop?

1. Bloop is a compile/test server

What is a compile/test server

- A compile/test server is a server that takes care of:
  - compiling your projects
  - running your tests
- Several clients can communicate with the same server.
- Compiler instances are shared between several projects and clients.
- Compilation state is shared between clients.
- Commands may run concurrently.

As a compile/test server, Bloop...

gives you faster compilation!

- The JVM works hard to optimize the code it's running.
- The more you compile, the faster it gets.
- The JITted code benefits all the clients using the same server!

```
$ time sh -c 'bloop clean mini-better-files; bloop compile mini-better-files'
        8.58s
$ time sh -c 'bloop clean mini-better-files; bloop compile mini-better-files'
        2.75s
$ time sh -c 'bloop clean mini-better-files; bloop compile mini-better-files'
        2.20s
$ time sh -c 'bloop clean mini-better-files; bloop compile mini-better-files'
        1.83s
```

As a compile/test server, Bloop...

gives you faster compilation!

Benchmarking compilation, warm sbt vs warm Bloop

Project	sbt	Bloop
guardian/frontend	88.62s	63.42s (72%)
sbt/sbt	38.05s	28.47s (75%)
apache/spark	237.24s	171.48s (72%)

As a compile/test server, Bloop...

lets you run tasks remotely

- The nature of Bloop lets you run your tasks remotely
- Your large compilation jobs can run on powerful clusters
- Your CI can benefit from compilation speed improvements

What is Bloop?

2. Bloop is a command line tool for compiling, testing and running your project.

As a command line tool, Bloop...

focuses on speed and simplicity:

- Bloop exposes a small, well-documented set commands

```
$ bloop compile my-project
$ bloop clean my-project
$ bloop test my-project
$ bloop console my-project
```

```
$ bloop compile --help
Command: compile
Usage: bloop compile <project>
  --project | -p  <value>
        The project to compile (will be inferred from remaining cli args).
  --incremental
        Compile the project incrementally. By default, true.
  --reporter  <value>
        Pick reporter to show compilation messages. By default, bloop's used.
[...]
```

- Commands use sane defaults to be as natural as possible.

As a command line tool, Bloop...

focuses on speed and simplicity:

- Bloop offers a snappy experience, right from your shell:

```
$ bloop compile neural-network
$ git checkout -- that/file/here
$ bloop test neural-network
$ bloop run neural-network -- -train images/**/*.jpg -out $HOME/data.dat
```

- You stay inside your usual shell.
- Benefit from it, rather than writing new `Task[T]`s!

As a command line tool, Bloop...

embraces your existing build:

- Your full-featured build tool generates Bloop's configuration
- Bloop takes care of the rest.
- Currently, sbt and Maven are supported
- More to come!
- You don't need to reconfigure anything.

As a command line tool, Bloop...

embraces your existing build:

- Don't remember how to run your test with Maven?
- Or how to run your app with sbt?
- Bloop exposes the same interface regardless of your build tool.

Why is Bloop exciting for Scala developers?

Bloop is exciting because...

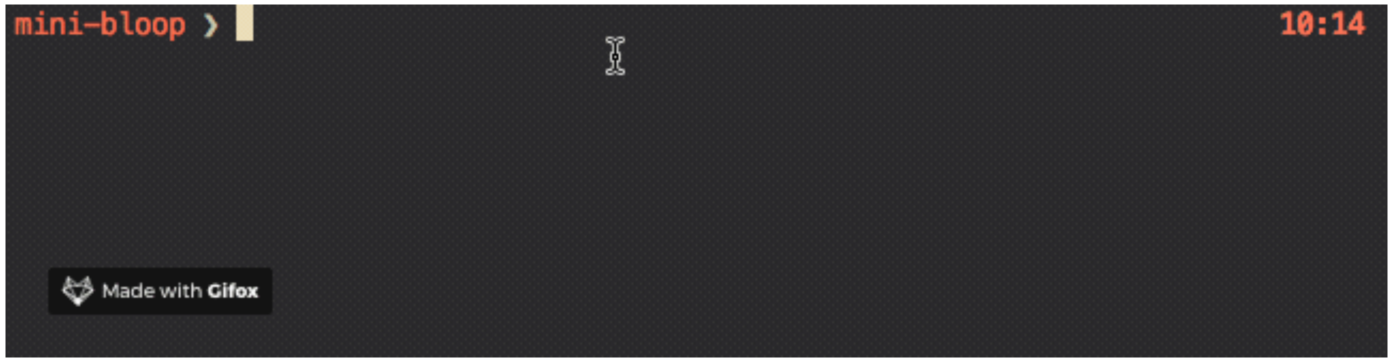
it is simple to use

- No complicated build DSL, no scopes, no commands vs tasks vs settings
- Unified interface across several build tools
- How do I just compile my tests in sbt?
- How do I run a specific `main` with pre-defined arguments?

Bloop is exciting because...

It is simple to use

- Bloop exposes only a few commands
- All of them can easily be discovered through `tab` completion



```
mini-bloop ❯                                                    10:14
```

Bloop is exciting because...

it is fast

- No startup time
- Closing your shell is not a big deal!
- Your hot compilers and project state will still be there.

Bloop is exciting because...

it is transparent

- Configuration files are easy to read:

```
$ cat .bloop/mini-bloop.json
{
  "version" : "1.0.0",
  "project" : {
    "name" : "mini-bloop",
    "directory" : "/projects/mini-bloop",
    "sources" : [ "/projects/mini-bloop/src/main/scala" ],
    "dependencies" : [  ],
    "classpath" : [
      "/Users/(...)/org/scala-lang/scala-library/2.12.3/scala-library-2.12.3.jar"
    ],
    "out" : "/projects/mini-bloop/.bloop/mini-bloop",
    "classesDir" : "/projects/mini-bloop/.bloop/mini-bloop/scala-2.12/classes",
    "scala" : {
      "organization" : "org.scala-lang",
      "name" : "scala-compiler",
      "version" : "2.12.3",
      (...)
```

Bloop is exciting because...

it is transparent

- Just read the configuration files to figure what values Bloop uses
- How do I find out what options are used to compile the tests?
- Why doesn't it find the classes for `that-other-project`?

Why is Bloop exciting for tooling authors?

"Bloop was definitely the enabler that made that possible"

-- *Jon Pretty*

Bloop is exciting because...

it opens up new possibilities

- Bloop is easy to integrate with other tools
- Via shell:

```
$ bloop test my-project && publish my-project # hypothetical
```

```
$ make # Call Bloop from Make
$ make publish # Why not?
```

- Via `BSP` (See Jorge's and Justin's talk)

Bloop is exciting because...

other tools can benefit from it

- What if your IDE extracted your projects' structures from Bloop?
- What if sbt delegated compilation to Bloop?
- What if your IDE ran your tests using Bloop?
- All those options are being explored via BSP

Bloop is exciting because...

it removes duplication of efforts

- By leveraging Bloop, other build tools can focus on what matters.
- Re-implementing test discovery, compiler interfacing, etc. doesn't require innovation.
- Let's build new frontends, new models, and delegate the gruntwork to Bloop.

Bloop is exciting because...

it can be used for experiments

- Bloop's codebase and model are small and simple.
- That makes Bloop a great platform for trying new things out.
- People are already experimenting with Bloop.

In the common use case when someone is working on a customer plugin in one build and wanting to iterate on core tooling <u>the turns arounds are seconds instead of 10+ minutes."</u>

(A happy Bloop user on simulating source dependencies with multiple separate repos)

Experimenting with Bloop

Bloopstrap

Quick-compile mode

Experimenting with Bloop

Bloopstrap

- Generate Bloop config from simple build definition.
- Targeted to projects that don't need a complex sbt / maven / ... build.

```
bloopstrap = {
  module       = "bloopstrap"
  organization = "com.github.duhemm"
  scala        = "org.scala-lang:scala-compiler:2.12.4"

  dependencies += ${libs.bloop}
  dependencies += ${libs.metaconfig}
}

libs {
  bloopV      = "1.0.0-M8"
  metaconfigV = "0.6.0"

  bloop      = "ch.epfl.scala:bloop-frontend_2.12:"${libs.bloopV}
  metaconfig = "com.geirsson:metaconfig-typesafe-config_2.12:"${libs.metaconfigV}
}
```

Experimenting with Bloop

Bloopstrap

- Builds are written in HOCON or JSON
- They are machine independent (should be checked in version control)
- Bloopstrap resolves your dependencies and generates the configurations
- Bloop can then take over!

Experimenting with Bloop

Bloopstrap

- Is that enough?
- Most likely not.
- But… should we always push more code into our builds?
- Is it normal to have thousands of LOC of tool-dependent code in my build?

Experimenting with Bloop

Bloopstrap

- Bloopstrap is definitely not a complete production-ready solution.
- Just an experiment on how to define the simplest declarative builds.
- Missing: How to publish without sbt?
- Check it out! Duhemm/bloopstrap

Experimenting with Bloop

Quick-compile mode

- Just testing that a project builds should be faster than full compilation.
- It's unnecessary to emit code for method bodies.
- Let's get rid of them to save time.
- `-Ystop-after:refchecks` is nice, but has limitations:
  - In multi-project builds, downstream projects need classfiles.
  - Zinc needs to extract (some more) info after `refchecks`.

Experimenting with Bloop

Quick-compile mode

- Let's first see how much improvement we can hope to see...
- (Cold) Compiling the scala library:
  - 51.6 seconds
- With `-Ystop-after:refchecks` (still cold):
  - 26.2 seconds
- This is almost 50%!

Experimenting with Bloop

Quick-compile mode

- What should we do?
- Beyond `refchecks`, we know that the trees are (mostly) correct.
- Let's write a compiler plugin that runs after `refchecks` and:
  - Replaces method bodies with `???`
  - Removes private symbols
- Let's remove specialization, too.

Experimenting with Bloop

Quick-compile mode

- We need to hook this plugin inside Bloop
- Prototype:
  - Duplicate every project into `normal` and `check` mode
  - The `check` project has our plugin as scalac option
  - When doing `compile` only, use the `check` project
  - When doing `run`, `test` or `console`, use the other project

Experimenting with Bloop

Quick-compile mode, results

- Compiling the (scala-library, scala-reflect, scala-compiler):
- Bloop normal mode: 66,933 ms
- Bloop quick mode: 52,751 ms
- About 21% better

Experimenting with Bloop

Quick-compile mode

- This is just a quick prototype
- Certainly has several problems. For instance: macros.
- Nothing is done with the fields at the moment.
- Shows how easy it is to hack stuff into Bloop!

What to look forward to in Bloop

What to look forward to in Bloop

Scala Native support

- Scala Native introduced a build API in 0.3.7
- Simple, stable API to use the Scala Native toolchain
- Developed it along with Bloop to see how integration would go
- Will be available soon in Bloop!

What to look forward to in Bloop

Scala.js support

- Scala.js will be supported in Bloop.
- This is planned for Bloop 1.0.0.
- Will probably happen soon after Scala Native support gets in.

What to look forward to in Bloop

Dotty support

- We already have a prototype
- It was blocked by Dotty being stuck on the old incremental compiler
- This is planned for Bloop 1.0.0.

What to look forward to in Bloop

Your idea?

- As I said, Bloop has a small, well-documented codebase
- Complete developer documentation on Bloop's website
- Contributions are very welcome
- Improvements to Bloop will benefit many tools!

Thanks!

Questions?

MacOS:

```
$ brew install scalacenter/bloop/bloop
```

Linux / Windows:

```
$ curl -L https://tinyurl.com/bloop100M9 | python
```

https://github.com/scalacenter/bloop

https://scalacenter.github.io/bloop