Scala Center updates, Q2 2018 Advisory Board Meeting

Scala Center team: Jorge Vicente Cantero, 100% Ólafur Geirsson, 100% Martin Duhem, 100% Guillaume Massé, 100% Julien Richard-Foy, 100% Darja Jovanovic, 80% (intern) Heather Miller, 50%

Initiatives worked on this cycle:

- Bloop
- BSP
- Pipelined compilation
- scalac-profiling
- Zinc
- Pending Scalameta v4.0
- Improved SemanticDB code health
- Improved SemanticDB performance
- Scalameta Native
- Pending Scalafix v0.6
- Bloop
- Dotty
- Scala Native
- load-plugin
- MOOCs
- Collections
- scalajs-bundler
- Accessible Scala
- Other activities: talks, sprees, conferences, SIP

Bloop

@jvican

There has been a lot of activity for the past three months in Bloop:

1. 19 OSS Scala contributors have merged changes in the repository.
2. Release of three milestones with lots of features, improvements and bugfixes. Read the release notes to have an idea of all the developments during the past three months in Bloop. We plan on cutting `1.0.0` next week after confirming the stability of the compilation server.
    i. Release notes for v1.0.0-M9
    ii. Release notes for v1.0.0-M10
    iii. Release notes for v1.0.0-M11

My changes in the repository account for a total of 11,997 lines added and 6,987 lines removed.

My work this past three months has focused on designing a stable configuration file format, improving the sbt integrations and implementing all the features required for the Build Server Protocol.

BSP

@jvican

I have released the first version of BSP, `v1.0.0`. In this release, I've worked with Justin Kaeser and Olafur Páll Geirsson in refining the theoretical details of the specification.

The Build Server Protocol has been announced in this scala-lang blog post (*recommended read*). The blog post contains images showing an implementation of the Build Server Protocol in Bloop (acting as a server) and IntelliJ (acting as a client).

This implementation has helped to refine and prove the feasibility of such a protocol. It features a fast project import and collection of compiler diagnostics from the build tool and displaying them in the editor. The blog post explains other ways this integration can be improved to be better than the stock Gradle, Maven and sbt integrations that IntelliJ currently has.

Aside from this, there has been a lot of work in presenting this work in the talk that Justin and I presented in Scalasphere 2018, and the preparation of a sneak peek in our ScalaDays talks.

Pipelined compilation

@jvican

I have implemented a prototype of pipelined compilation that I plan to merge in Bloop some time during the next weeks.

What is pipelined compilation? Pipelined compilation is a technique proposed by Rory Graves on Scalasphere 2017 to speed up compilation in the context of build graphs. Builds usually compile modules in their topological order and in order to compile a given module, all its project dependencies need to have been compiled before.

What pipelined compilation enables is to start compilation of dependent modules *right after* the typechecking of its project dependencies.

For build graphs that are sequential (modules cannot be compiled in parallel at any point), pipelined compilation achieves a *theoretical* speedup of

30%, taking into account that typer usually takes around 60% of the compilation pipeline.

For build graphs that are parallel, the speedup can be even bigger depending on the shape of the build graph and how many modules can be compiled in parallel.

The practical performance speedup for real-world projects is yet to be assesed in the next weeks. A detailed report analyzing its impact in OSS Scala projects will be presented to the community.

The implementation is not yet merged and is split across the Zinc and Bloop repositories.

1. Zinc changes.
2. Bloop changes.

When the implementation is more solid and there is a better understanding of the performance implications, a PR to merge the zinc implementation upstream will be opened.

scalac-profiling

@jvican

I have published a blog post featuring the use of scalac-profiling to speed up compilation times. The blog post explains how I used scalac-profiling in bloop to speed up compilation by 8x by removing unnecessary macro expansions and deduplicated implicit searches. The plugin generates flame graphs for macro expansions and implicit search and allows more applications explained in the repository.

This blog post has been in the makings for a long time due to two reasons: the difficulty of getting reliable performance numbers for every change in the guide and the need for modifications in the plugin to remove compilation overhead and handle big amounts of implicit searches and macro expansions (in the order of tens of thousands).

The new release also integrated with some changes with regards to performance that were merged in Scala 2.12.6 and that the plugin needed to interface with. PR highlights:

1. Add better profiling for macro and implicit interaction.
2. Add reproduced version of case-app ineffiency.
3. improve the internals of the plugin implementation to scale up.

Zinc

@jvican

- Zinc now detects changes in private members of traits.
- General maintenance work and PR reviews.

Pending Scalameta v4.0

In collaboration with Eugene Burmako from Twitter we released a total of 7 Scalameta releases. Release notes:

- v4.0.0-M2
- v4.0.0-M1
- v3.7.3
- v3.7.2
- v3.7.0
- v3.6.0

There are 26 open issues remaining to complete the v4.0 milestone.

The primary focus for v4.0 is to stabilize the SemanticDB specification and ensure the following tools are consistent the spec:

- `metac`: compiler plugin to emit complete SemanticDB files
- `metacp`: command-line tool and library to emit SemanticDB types for public signatures in a classpath
- `metap`: command-line tool and library to pretty-print SemanticDB

To read more about these tools, see the SemanticDB guide and my slides for "SemanticDB for Scala developer tools" for my at ScalaSphere in April.

Improved SemanticDB code health

The primary focus of Scalameta v4.0 is to stabilize SemanticDB APIs and part of that goal involves cleaning up unnecessary abstractions in the codebase. I had observed during the Scala Spree in Krakow that first-time contributors struggled to become productive in the SemanticDB codebase. One milestone towards improving the situation was PR #1506. This PR did a significant refactoring to get rid of one unnecessary layer of abstraction. The PR resulted in the removal of ~5.000 lines of code making the code easier to reason about and contribute to. It was gratifying to validate that code health had had indeed improved by receiving a non-trivial PR from a first-time contributor at the Scala Spree in flatMap(Oslo), after the PR #1506 had been merged.

Improved SemanticDB performance

@olafurpg

One positive side-effect of the recent work in SemanticDB is improved performance. Benchmarks compiling the Slick and Akka projects show that the overhead of enabling the SemanticDB compiler plugin has dropped by over 40% compared to v2.1.7, which is the SemanticDB version used by the latest stable release of Scalafix.

	Slick	**Akka**	**Overhead**
Baseline	113s	100s	0%
v2.1.7	170s	170s	50-70%
v4.0.0-M3	123s	125s	8-25%

The benchmark numbers were obtained by consecutively compiling the projects via sbt and collecting the hot compile times once the numbers stabilized. My hypothesis for why the overhead is lower in Slick compared to Akka is that a larger fraction of the compile-time is spent during typechecking in Slick.

Note that performance has not been a primary focus recent months. There remain plenty opportunities to reduce the overhead of the compiler plugin even further if we choose to invest more in that area.

Scalameta Native

@olafurpg

Scalameta v4.0.0-M1 added support for Scala Native, enabling users to build command-line tools with instant startup times. Previously, command-lines tools using Scalameta such as `scalafmt` (code formatter) suffered by slow startup times of the JVM making it difficult to integrate with editors like Vim and Sublime Text. Early results by building a `scalafmt` binary with Scala Native show that formatting a single file from the command-line can go from 1.5s on the JVM down to 50ms with native, a 30x speedup.

The GIF below shows `metap`, a command-line tool to print SemanticDB files, running for 600ms on the JVM and 15ms with Scala Native

Scala Native enables Scala developers to build a whole new category of developer tools that previously suffered badly from slow startup times on the JVM.

Pending Scalafix v0.6

@olafurpg

Scalafix v0.6 is currently blocked by the remaining 26 ticket in the Scalameta v4.0 milestone. Recent work in Scalafix has involved many interesting PRs, of which highlights include:

- #737 Big cleanup of internals taking the first steps towards Scalafix v1. The PR touches on a lot of parts, including:

    - New `scalafix.v1` package containing a redesigned semantic API taking advantage of the lessons learned over the past year with SemanticDB. The `scalafix.v1` API is still evolving.
    - Move current APIs to `scalafix.v0` package so that existing rules can continue to work (automatic migration provided).
    - Significantly lower memory pressure when running the command-line interface on larger projects.
- #699 migrate `ExplicitResultTypes` to take full advantage of SemanticDB
- #696, #690, #689, #687, #666, #665, address user reported issues on the sbt plugin
- #741 Add `ScalafixTestkitPlugin` to simplify the g8 template used for custom rules. An automatic rewrite is provided that migrates the sbt builds for custom rule authors.

Read the release notes for v0.6.0-M2 for more details on activity including community contributed improvements.

Martin Duhem

Between mid-march and today, I've been focused mainly on Bloop and on Dotty.

On the Bloop side, I have made a lot of progress and added much requested features such as support for Scala Native and Scala.js. I have presented Bloop in Krakow at Scala Sphere, and co-presented at Scala Days in Berlin with Jorge Vicente Cantero (and will co-present with him again at Scala Days New York).

Regarding Dotty, I have been focusing on improving the getting started experience, adding more features to Dotty IDE, fixing some bugs in the compiler and working on Dottydoc (the equivalent of scaladoc, for Dotty).

I also did some work on Scala Native to make it possible for other tools to run tests written with Scala Native. This is a continuation of my previous work to add a build API to Scala Native.

Finally, I started an experiment to make it easier to start and configure language servers backed by sbt. This is used by `metals` for starting up, and a proposal for use in Dotty is open. I've used it to provide a better getting started experience for newcomers to the Scala language.

Bloop

Improvements

Support linking and running of Scala Native projects

Up to this point, Bloop was able to compile all kinds of projects, including Scala.js and Scala Native projects, but it couldn't run them not perform the last steps of the pipeline (linking, and generating either a native binary or JS code).

I have been working towards fully supporting Scala Native in Bloop, and this work has recently been merged.

Thanks to these changes, users of Bloop can seamlessly compile, link and run their Scala Native projects.

This work: https://github.com/scalacenter/bloop/pull/457

Support linking and running of Scala.js projects

Thanks to the great community that formed itself around Bloop, we are very lucky to see exciting contributions being submitted by our users. Tim Nieradzik (@tindzk) has submitted a patch that added support for Scala.js, by re-using the bits I introduced to support Scala Native.

I have been adding tests and polishing this implementation in order to be able to support Scala.js in Bloop.

Similarly to Scala Native, compiling, linking and running is supported, but running the test is not supported yet. We plan on adding that very soon.

This work: https://github.com/scalacenter/bloop/pull/526

Support running tests of Scala Native projects

Even though Bloop can compile and run Scala Native projects, it is still not able to run the tests. I have been working on a fully working prototype where running the test is possible, but unfortunately its integration in Bloop is waiting for a new release of Scala Native, where the test runner is split out of the sbt plugin.

The necessary modification to Scala Native have been made. More information about this part is available in the "Scala Native" section of this report.

This work: https://github.com/scalacenter/bloop/commit/84395b666eafdeae6f2f664a30a39fc86c6bce7d

Support compiling with Dotty in Bloop

An early prototype of compiling with Bloop using Dotty written by Jorge Vicente Cantero existed, but never made it to the finish line because we were blocked by Dotty being compatible with Zinc 1.0. Thanks to the recent release, I have been able to rebase the previous work and use it for completely supporting Dotty in Bloop.

This work: https://github.com/scalacenter/bloop/pull/460

Extract `classpathOptions` in sbt-bloop

In order to correctly configure the compiler, Bloop needs to know precisely how to configure the classpath for compilation. This info is know in sbt as the `classpathOptions`.

Having the `classpathOptions` lead to other performance work, which is the next item on this list.

This work: https://github.com/scalacenter/bloop/pull/110

Move `classpathOptions` to `CompileConfiguration`

Zinc needs `ClasspathOptions` to know how to configure the classpath for compilation (those options answer questions such as "should the Scala library be left on the classpath?" or "Should the compiler be included", etc.) These options were passed directly to the class that wraps a compiler instance, rather than passed directly when the compiler is actually invoked.

This means that it was not possible to share the same compiler instance between different projects that had different `ClasspathOptions`, because the wrapper around the compiler had to be different. All in all, this caused either performance issues or correctness issues.

I refactored Zinc so that the `ClasspathOptions` are passed when the compiler is invoked.

This work: https://github.com/scalacenter/zinc/pull/4

- Many bugfixes in all parts of the project
- Improvements and fixes to our benchmarking infrastructure

Bugfixes

Kebabize name of arguments in autocompletion

Bloop sometimes generated autocompletion for wrong argument names, because of an error in how the completion suggestions were generated. I fixes this issue so that the correct names are proposed.

This work: https://github.com/scalacenter/bloop/pull/487

Fix depth of project loading

Bloop used to try loading all files whose name matched `*.json` in a directory hierarchy, regardless of their depth in the hierarchy. This caused performance issues and, under certain conditions, Bloop could try loading files that were not project definitions. No more.

This work: https://github.com/scalacenter/bloop/pull/366

Dotty

Improvements

Sharing more infrastructure between Dotty and Dottydoc

Dotty features a powerful miniphase infrastructure which allows it to squash several phases and perform less tree traversals. Dottydoc is built around the same idea, but unfortunately, very little code to perform the phase squashing is shared between the two.

Moreover, Dottydoc phases do not resemble, in the way they are written, the phases of Dotty. This is confusing for the developers and leads to bugs.

I worked on a large refactoring of Dottydoc, so that the Mega/Miniphase infrastructure of Dotty is reused. This allowed me to re-write the phases in a style that is closer to the rest of the compiler and easier to understand.

Overall, the code after refactoring is shorter (by about 10%) and easier to understand than the previous implementation.

This work:
https://github.com/lampepfl/dotty/compare/master...Duhemm:topic/dottydoc-simplify

Keep documentation on package objects

Scaladoc and Dottydoc allow users to put documentation for a whole package by adding it to the package object. Unfortunately, this was broken in Dottydoc and the documentation put on package objects would never show up in the generated documentation.

I fixed the ordering of the phases of Dottydoc so that this configuration is no longer lost.

This work: https://github.com/lampepfl/dotty/pull/4573

Enable `-Ycheck:all` in Dottydoc

`-Ycheck` is the flag that we use in Dotty to perform more verification during compilation, to make sure that the state in which the code is is correct after every phase of the compiler.

Unfortunately, the way Dottydoc was working meant that several invariants of the compiler would be violated, which prevented us from enable `-Ycheck` when running Dottydoc.

The problem was that Dottydoc would generate JVM incompatible names for the symbols introduced with `@usecase` in the documentation. Changing the way these names are encoded so that they remain unique and compatible with the JVM fixed our problems and allowed us to perform more verifications in the tests of Dottydoc.

This work: https://github.com/lampepfl/dotty/pull/4575

Sticky attachments

In Dotty, documentation is attached to the trees as "attachments". These attachments can then be retrieved and used in other parts of the compiler.

Unfortunately, those attachments were lost when the tree went under transformation. For instance, the attachments that were given to `object`s were lost during desugaring.

To circumvent this problem and recover the elusive attachments, I introduced the concept of sticky attachments in the compiler, which are attachments that survive tree transformation.

This work: https://github.com/lampepfl/dotty/pull/4292

Typed imports

Imports are not typed in Dotty, because it is not obvious what type should be given to these nodes. Indeed, in imports, the same name can mean both a type and a term, for instance.

Because imports are untyped, this means that they need to be special cased in many places, such as the IDE for instance: jump-to-definition doesn't work on import nodes, because the node doesn't have a type.

The second problem is that, Instead of being verified during the `typer` phase, imports are checked in `posttyper`. Unfortunately, this means that wrong imports won't be reported for programs that do not pass the `typer`, producing confusing error messages where the root of the error is not mentioned.

In order to solve that, I worked on a proof of concept where imports are typed. In order to give types to import nodes, I re-use the kind of denotation that is assigned to overloaded symbols. This design works and solves our problem, but we're worried about the additional complexity that incurs from this change, and are still thinking about other way around the problem.

This work:
https://github.com/dotty-staging/dotty/commit/cc62c2849a9a11cf689780f4645db2bf591bb787

Jump to definition in imports

Jump-to-definition doesn't work on import nodes, because they need special casing (see previous section).

I worked on Dotty IDE so that jump-to-definition can be supported on those nodes. This doesn't use the work I described in "Typed imports", but instead relies on special casing handling of those nodes. We're waiting to see what is the best design before integrating this into the compiler.

This work: https://github.com/lampepfl/dotty/pull/4199

Add tests for Dotty IDE

Together with Nicolas Stucki, we worked on adding tests for Dotty IDE, which only relied on manual testing beforehand. These tests allow us to quickly and simply write interactive tests, where the language server can be queried given a position or range of positions.

This work: https://github.com/lampepfl/dotty/pull/3766

Bugfixes

Intersection dominator of array types

An error in the compilation of the dominator type for intersection types that involve `Array`s caused wrong generic java signatures to be generated by Dotty, for code such as:

```
def foo[U](u: Array[Int] & Array[U]): Unit = ()
```

This work: https://github.com/lampepfl/dotty/pull/4249

Generic Java signature for FunctionXXL

In Dotty, functions with more than 22 type parameters are converted into `FunctionXXL`. Unfortunately, Dotty generated wrong Java generic signatures for these functions, and instead generated the signature for `Function`, and not `FunctionXXL`. I fixed this issue.

This work: https://github.com/lampepfl/dotty/pull/4287

Fix crashes in the Dotty language server

A recent refactoring in the Dotty language server caused it to crash sometimes at startup. The reason for crashing was that it sometimes tried to continue reading a zip file after closing it.

This work: https://github.com/lampepfl/dotty/pull/4208

Ignore `.sbt` files in Dotty IDE

Dotty IDE was trying to typecheck and provide completions for `.sbt` files, which was doomed to fail because it doesn't have the necessary information to understand those files.

The reason for this bug existing in the first place, is that the Dotty Language Server was being enabled for all files that are considered by VSCode as "Scala" files, which included sbt files.

I changes our VSCode plugin so that these files are no longer inspected by the Dotty language server.

This work: https://github.com/lampepfl/dotty/pull/4556

Fix positions of lifted expressions

When lifting expressions (such as in `val foo = bar :: Nil`), Dotty was assigning wrong positions to the new trees. This error was found when we noticed that jump-to-definition was not working on the left operand of right associative operators.

I fixed the positions of the lifted expressions so that they are correct, which allowed Dotty IDE to behave correctly.

This work: https://github.com/lampepfl/dotty/pull/4516

Scala Native

Moving the test runner out of the sbt plugin

Test written with Scala Native need a special runner to be executed, because they cannot be started using reflection, as it is done on the JVM. Instead, I developped a few months ago a solution (very inspired from the solution used by Scala.js) that starts a

server which will communicate with the build tool, and orchestrate the test runs. This allows us to run tests transparently in sbt, via a special test runner.

In order to be able to make other tools (such as Mill or Bloop) able to use it, I worked to split this out of the sbt plugin that Scala Native offers. This work has not been released yet, but will certainly be part of the next release of Scala Native.

This work: https://github.com/scala-native/scala-native/pull/1234

load-plugin

`load-plugin` is a set of functions that can be injected in sbt in order to make it easier to dynamically load sbt plugins (`load-plugin` itself is not an sbt plugin.)

`load-plugin` adds two commands to sbt:

- `load-plugin` which downloads and loads a plugin inside an already loaded build:
- ```
  > load-plugin com.eed3si9n:sbt-assembly:0.14.6 sbtassembly.AssemblyPlugin
  # This plugin is downloaded and loaded inside the build
  ```

- `if-absent` which will perform some action if a plugin absent from a build:
- ```
  > if-absent dotty.tools.sbtplugin.DottyPlugin \
    "set every scalaVersion := \"0.9.0\"" \
    "load-plugin ch.epfl.lamp:sbt-dotty:0.9.0 dotty.tools.sbtplugin.DottyPlugin\""
  # If the Dotty plugin is not already loaded, set the Scala version and load the plugin.
  ```

Together, these commands can be used to inject programmatically any kind of sbt plugin, notably language servers.

This also has the advantage of being able to handle both configure and un-configured (as in, default) builds in sbt.

I've been using this to develop a proof of concept with Dotty IDE where one can simply open VSCode inside a completely empty directory, and have the Dotty LSP start up when a Scala file is created. This server is configured on the fly, but is able to perform fully and give accurate help to the user, completely seamlessly.

This re-creates the experience that users get when using VSCode with a Java language server, and is the best experience in my opinion.

This work: https://github.com/scalacenter/load-plugin Integration in Dotty: https://github.com/lampepfl/dotty/pull/4304Integration in Metals: https://github.com/scalameta/metals/pull/287

MOOCs

- Mentioned our openedx instance on scala-lang.org. #1079
- Filtered out errors caused by CEDE's infrastructure instability #380
- Reviewed/tweaked contents of the akka-streams lessons
- Engaged people to discuss on our openedx instance
- Polished the slides and assignments of week1
- Submitted MOOC to CEDE
- Wrote a chapter about FRP
- Wrote technical documentation about our MOOCs infrastructure

Collections

Published a blog article, Scala 2.13's Collections

- Moved extensibility framework to the core collections #6674
- Added user documentation #1078
- Reviewed lampepfl/dotty#4317
- Made reported positions deterministic in Scala.js #3346
- Updated the FAQ
- Renamed the migration rule #18 and explained future plans
- Clarified the new structure of the collections' projects (scala-collection-compat, scala, collection-strawman) #558#562#564
- Created https://github.com/julienrf/scala-collection-contrib
- Added CLA check to scala-collection-compat
- Hunted bugs, regressions, issues #6541#6542#6543#6544#6545
- Upgraded scala-parser-combinators #148
- Backported PRs from collection-strawman to scala
- Initiated the Scala.js library migration to 2.13.0-M4 #3329
- Fixed partest enrich-gentraversable #6488

- Reverted renaming of a public method in scala-reflect #6489
- Collaborate with Dotty team on soundness issues in the strawman #521
- Fixed bug in ListBuffer #6469
- Added BuildFrom to scala-collection-compat #3
- Added support for Scala 2.11 to scala-collection-compat #5
- Added `lazyAppendAll` alias to `append` #6
- Added an `iterator()` alias to `iterator` #7
- Struggled again to setup scoverage for the scala library
- Updated PR adding support for 2.13 to scala-parser-combinators #134
- Added rewrite rule from `append` to `lazyAppendAll` #527
- Removed rewrite rule that replaced `Stream` with `LazyList` #528
- Migrated scalacheck to 2.13.0-M4 #391
- Fixed some Scala integration tests #6482
- Finished ArrayDeque, Queue #490
- Backported all fixes applied to scala/scala #493
- Fixed a few bugs #494, #495, #496, #497, #498, #499, #500

scalajs-bundler

- Reviewed and merged PRs (most notably, #247, #241, #234)
- Released 0.11.0 and 0.12.0

Accessible Scala

Scala is proudly a welcoming environment for all. The Scala Center is demonstrating this by supporting the development of Accessible Scala, a tool for blind and partially-sighted developers (see SCP-016).

This quarter, we completed the project and delivered various artifacts: An online demo (video) you can try in your browser, a vscode extension, and a library to describe scala code verbally. Besides, we solved the issue of describing complex expressions with a technique we call the Cursor. It consists of navigating the AST and describe each nod we visit.

You can find more details on the blog post we released.

---------- insert Jorge report----------

Other activities: sprees, talks, conferences

Sprees, talks, conferences

Scala Sphere, Krakow, 25-17 April

Scala spree

In collaboration with VirtusLabs who hosted and sponsored the venue, the Scala Center team organised a Scala Spree on April 15th. There were about 25 participants that had an opportunity to work on 12 libraries and tools and successfully merged many PRs by the end of the event.

Spree photo

Talks

Ólafur Páll Geirsson presented "SemanticDB for Scala developer tools" Slides

Martin Duhem presented "Meet Bloop" Slides

Jorge Vicente Cantero presented "Build Server Protocol and New IDEAS" together with Justin Kaeser from Jetbrains Slides

flatMap Oslo, 3-4 May 2018

Scala spree

In collaboration with Arktekk who hosted and sponsored the venue, Scala Center team organised Scala spree on May 2nd. There were 9 participants that had an opportunity to work on Dotty, Scalameta and Scalafix libraries which resulted in 6 open PRs by the end of the event.

Spree photo

Talks

Ólafur Páll Geirsson Presented "Six steps from zero to IDE" Slides. Talk

Scala Days Berlin, 14-18 May 2018

Scala spree

In collaboration with Zalando who hosted and sponsored the venue, Scala Center team organized a Scala spree on 14th of May. This was a two to three times bigger spree than any before and the library authors decided to propose a different structure to be able to scale the learning experience to match the number of participants. There were about 60 participants that had an opportunity to work on the Scala compiler, Dotty compiler, sbt, Scalafix and Bloop. By the end of the event, 8 pull requests were merged into sbt, 10 people completed an intensive Scalafix workshop, 2 long-standing issues were closed in Bloop and more. *Scala Days takeaway blog mentioning the spree* *Spree photo*

Talks

Ólafur Páll Geirsson Presented "Six steps from zero to IDE" Slides.

Julien Richard-Foy and Stefan Zeiger presented “Migrating to Scala 2.13” Abstract

Martin Duhem and Jorge Vicente Cantero presented "Meet Bloop" Slides

Scala Contributors Summit

On the 18th of May, we had the first Scala Contributors Summit in the Zalando offices. Gathering about 50 people involved in the development, documentation and community building around Scala, it was the opportunity to discuss overarching plans for contributions. After an initial brainstorming of topics the attendees thought worth discussing, they split in various working groups with different focuses, ranging from tooling to migration plans for Scala 3, macros, build tools to education and education. Most working groups concluded with concrete tasks that could be undertaken in the near future.*Contributors Summit photo*

SIP Meeting May 2018

We scheduled a SIP meeting the last hour of the Contributors Summit because most of the Committee Members were participating that day and could meet in person. The SIP meeting was about the role of the Committee in the upcoming changes towards Scala 3. To read/watch more about the SIP May 2018 please go to: minutes or watch the meeting on Scala Center's YouTube channel

In short: Quorum was not met, but the meeting still took place in a form of an open discussion. The meeting was devoted to discussing how SIP Committee should handle the approval of Scala 3 changes into the specification and how it would organize during the next year, given that Scala 3 will be feature freeze by then.