LogStage - zero cost structured logging

Maksym Ratoshniuk, 7mind.io

Yet another logger? 🤔

What is a structured logging?

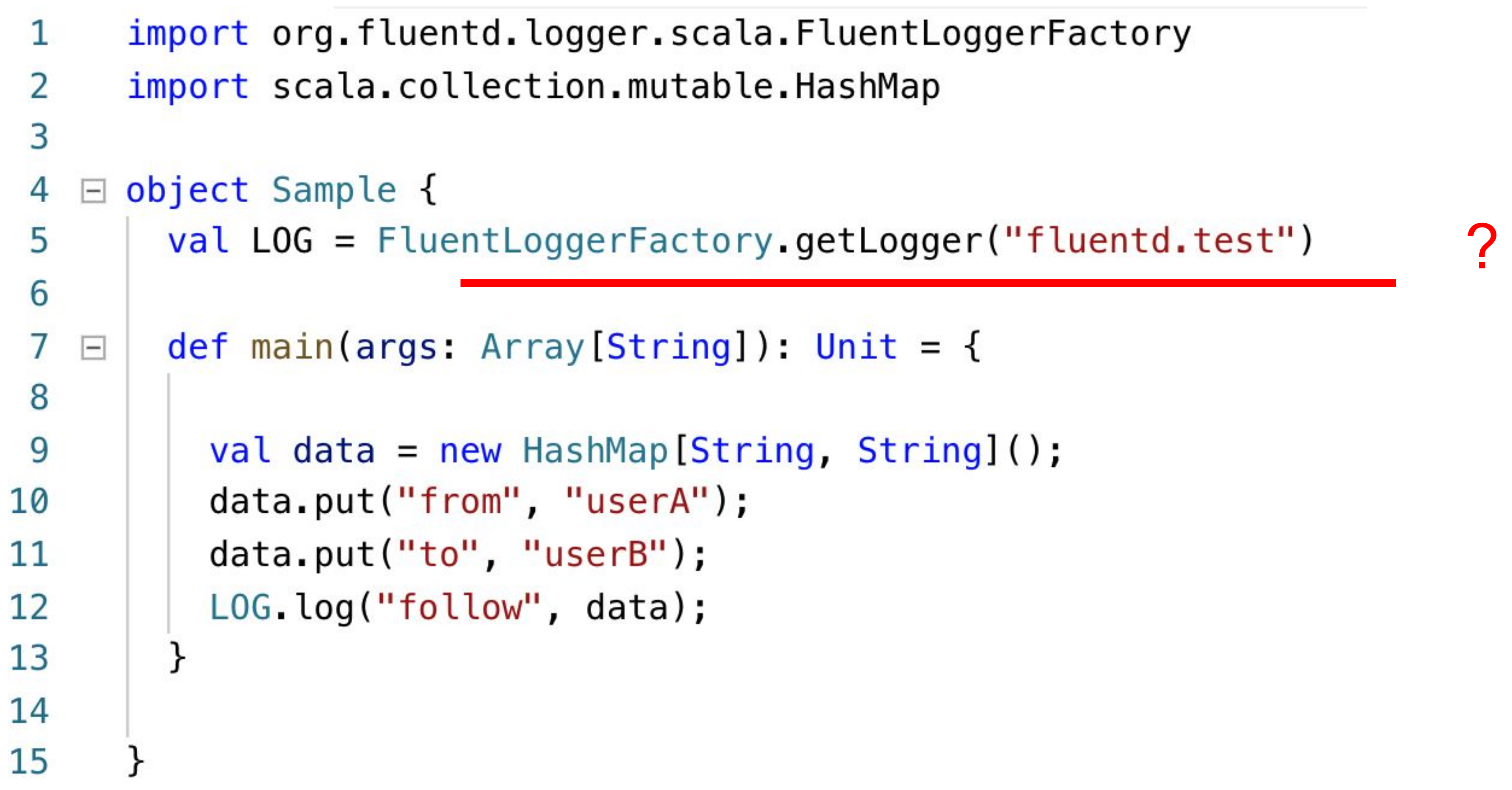
Common use cases:

- Tracing
- Liveness of your system
- Metrics
  - DB Calls
  - Http flow
  - Payment processing
  - Scheduling (jobs)
- Analytics
  - User behaviour
  - A/B Testing

Challenges

1. Broken SOLID and lot’s of magic

Fluentd



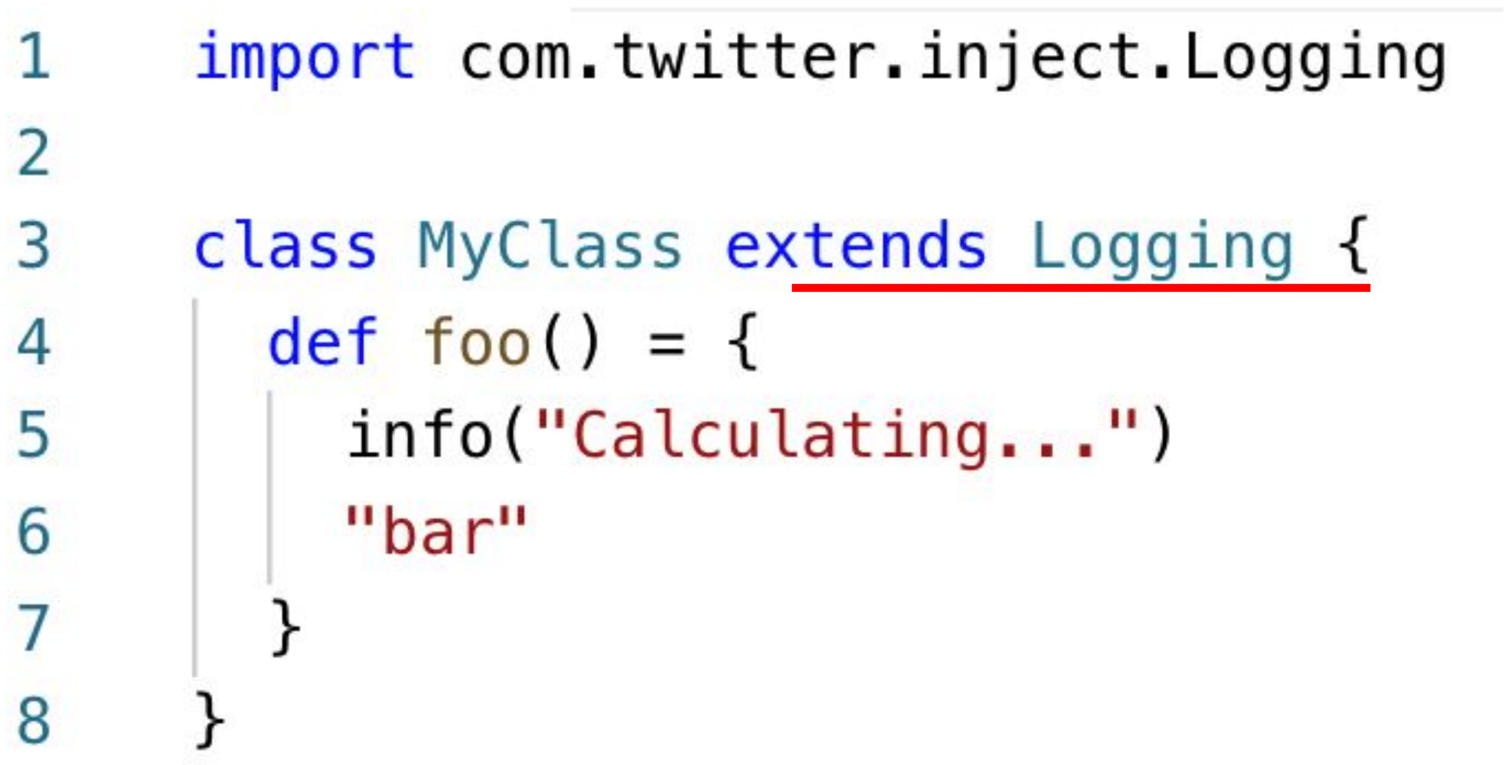
```scala
import org.fluentd.logger.scala.FluentLoggerFactory
import scala.collection.mutable.HashMap

object Sample {
  val LOG = FluentLoggerFactory.getLogger("fluentd.test")

  def main(args: Array[String]): Unit = {

    val data = new HashMap[String, String]();
    data.put("from", "userA");
    data.put("to", "userB");
    LOG.log("follow", data);
  }

}
```

?

Scala-logging

```scala
import com.typesafe.scalalogging._
import org.slf4j.LoggerFactory

class MyClass extends LazyLogging {
  // ...
  val logger = Logger(LoggerFactory.getLogger(this.getClass))
  logger.debug("Here goes my debug message.")
  // ...
}
```

Finatra

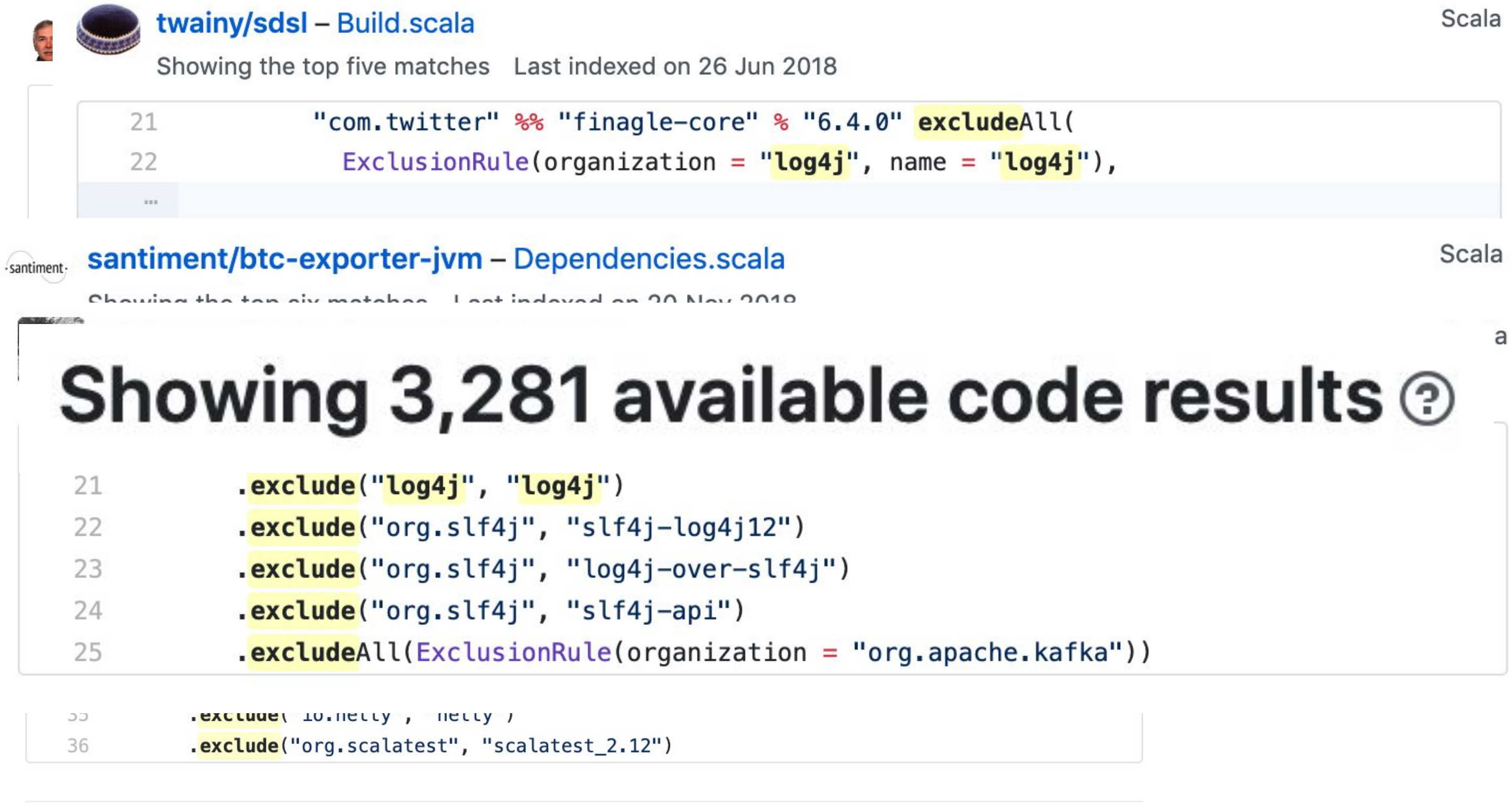


```scala
import com.twitter.inject.Logging

class MyClass extends Logging {
  def foo() = {
    info("Calculating...")
    "bar"
  }
}
```

2. Production unreadiness

- Asynchronous sinks
- File rotation
- Json rendering
- User context (like MDC)
- No performance hotspots

3. Jar Hell



twainy/sdsl – Build.scala

Scala

Showing the top five matches Last indexed on 26 Jun 2018

```
"com.twitter" %% "finagle-core" % "6.4.0" excludeAll(
  ExclusionRule(organization = "log4j", name = "log4j"),
...
```

santiment/btc-exporter-jvm – Dependencies.scala

Scala

Showing the top six matches Last indexed on 20 Nov 2018

a

Showing 3,281 available code results

```
.exclude("log4j", "log4j")
.exclude("org.slf4j", "slf4j-log4j12")
.exclude("org.slf4j", "log4j-over-slf4j")
.exclude("org.slf4j", "slf4j-api")
.excludeAll(ExclusionRule(organization = "org.apache.kafka"))
```

```
.exclude("io.netty", "netty")
.exclude("org.scalatest", "scalatest_2.12")
```

We always write code...

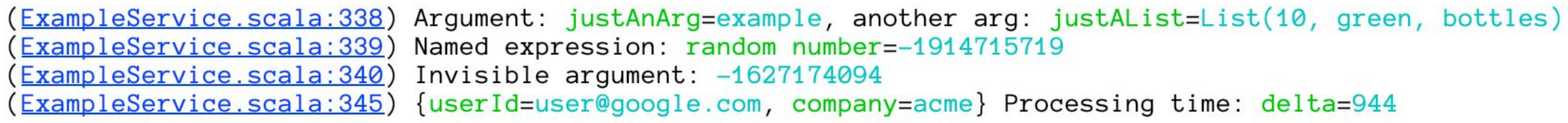
```scala
val user = "JohnDoe"
logger.debug(s"Received a message from $user")
```

… that is always structured

```
Expr(Apply(Select(
  Apply(
    Select(Select(Ident("scala"), scala.StringContext),
      TermName("apply"))
      , List(Literal(Constant("Received a message from "))
          , Literal(Constant("")))
        )
  ),
  TermName("s")
  )
, List(Ident(TermName("user")))
))
```

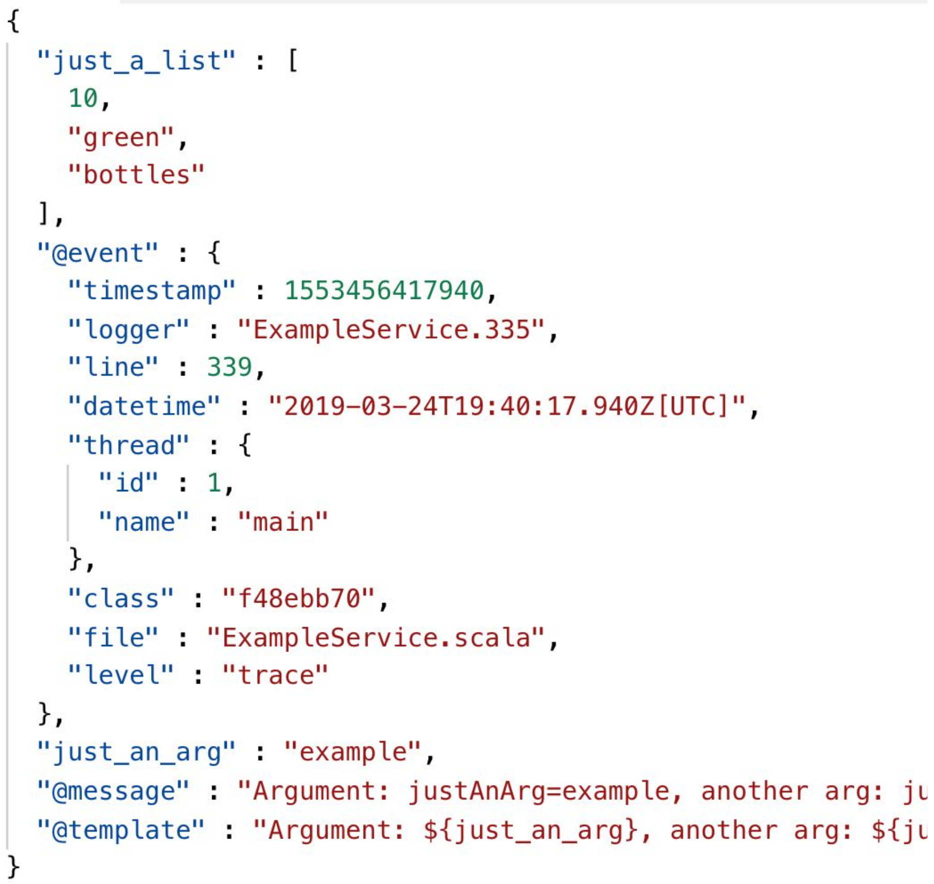
Logging always should be easy to read...

```scala
class ExampleService(log: IzLogger) {
  val justAnArg = "example"
  val justAList = List[Any](10, "green", "bottles")

  log.trace(s"Argument: $justAnArg, another arg: $justAList")
  log.info(s"Named expression: ${Random.nextInt() -> "random number"}")
  log.warn(s"Invisible: ${Random.nextInt() -> "random number" -> null}")

  val ctxLog = log("userId" -> "user@google.com", "company" -> "acme")
  val delta = Random.nextInt(1000)

  ctxLog.info(s"Processing time: $delta")
}
```



```
(ExampleService.scala:338) Argument: justAnArg=example, another arg: justAList=List(10, green, bottles)
(ExampleService.scala:339) Named expression: random number=-1914715719
(ExampleService.scala:340) Invisible argument: -1627174094
(ExampleService.scala:345) {userId=user@google.com, company=acme} Processing time: delta=944
```

… and easier to deal with



```
{
  "just_a_list" : [
    10,
    "green",
    "bottles"
  ],
  "@event" : {
    "timestamp" : 1553456417940,
    "logger" : "ExampleService.335",
    "line" : 339,
    "datetime" : "2019-03-24T19:40:17.940Z[UTC]",
    "thread" : {
      "id" : 1,
      "name" : "main"
    },
    "class" : "f48ebb70",
    "file" : "ExampleService.scala",
    "level" : "trace"
  },
  "just_an_arg" : "example",
  "@message" : "Argument: justAnArg=example, another arg: ju
  "@template" : "Argument: ${just_an_arg}, another arg: ${ju
}
```

```scala
class ExampleService(logger: IzLogger)
{
  val justAnArg = "example"
  val justAList = List[Any](
     10, "green", "bottles"
  )
  logger.trace(
   s"Argument:   $justAnArg, another
arg: $justAList"
  )
}
```

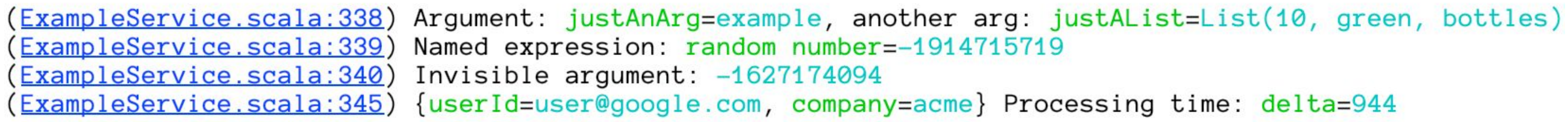
LOGSTAGE

First-class logging framework for Scala

Features

Macro-based structuring and context extraction

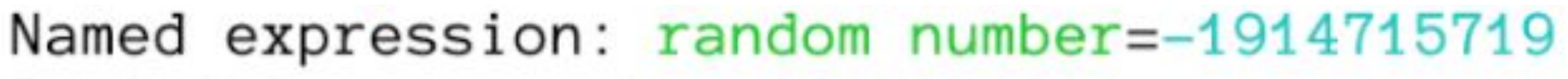
- Argument names, types, ordering
- Static information (file, line, class, function)
- Static part of our message - message template (interpolation context)



```
(ExampleService.scala:338) Argument: justAnArg=example, another arg: justAList=List(10, green, bottles)
(ExampleService.scala:339) Named expression: random number=-1914715719
(ExampleService.scala:340) Invisible argument: -1627174094
(ExampleService.scala:345) {userId=user@google.com, company=acme} Processing time: delta=944
```

Aliasing for references

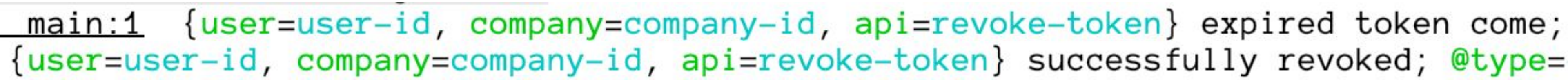
```scala
logger.info(s"Named expression: ${Random.nextInt() -> "random number"}")
```



```
Named expression: random number=-1914715719
```

Dynamic context & method granularity

```scala
def revokeToken(user: User) : Either[Throwable, String] = {
    val revokeTokenLogger = logger.apply(
      "user" -> user.id ,
      "company" -> user.companyId,
      "api" -> "revoke-token"
    )
    revokeTokenLogger.info("user has an expired token. start revoking")
    revokeService.revoke(user).map {
      token =>
        revokeTokenLogger.trace("successfully revoked")
        token
    }.leftMap {
      thr =>
        revokeTokenLogger.trace(s"fail to revoke. ${thr.getMessage -> "reason"}")
        thr
    }
}
```

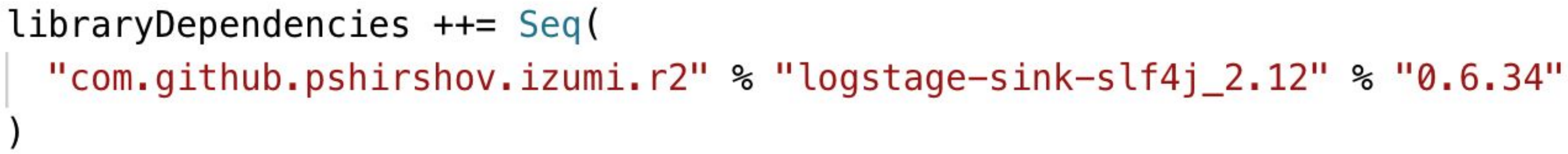


```
 main:1  {user=user-id, company=company-id, api=revoke-token} expired token come;
{user=user-id, company=company-id, api=revoke-token} successfully revoked; @type=
```

Reference configuration

SLF4J backend

A drop-in replacement for Logback, route your legacy logs also

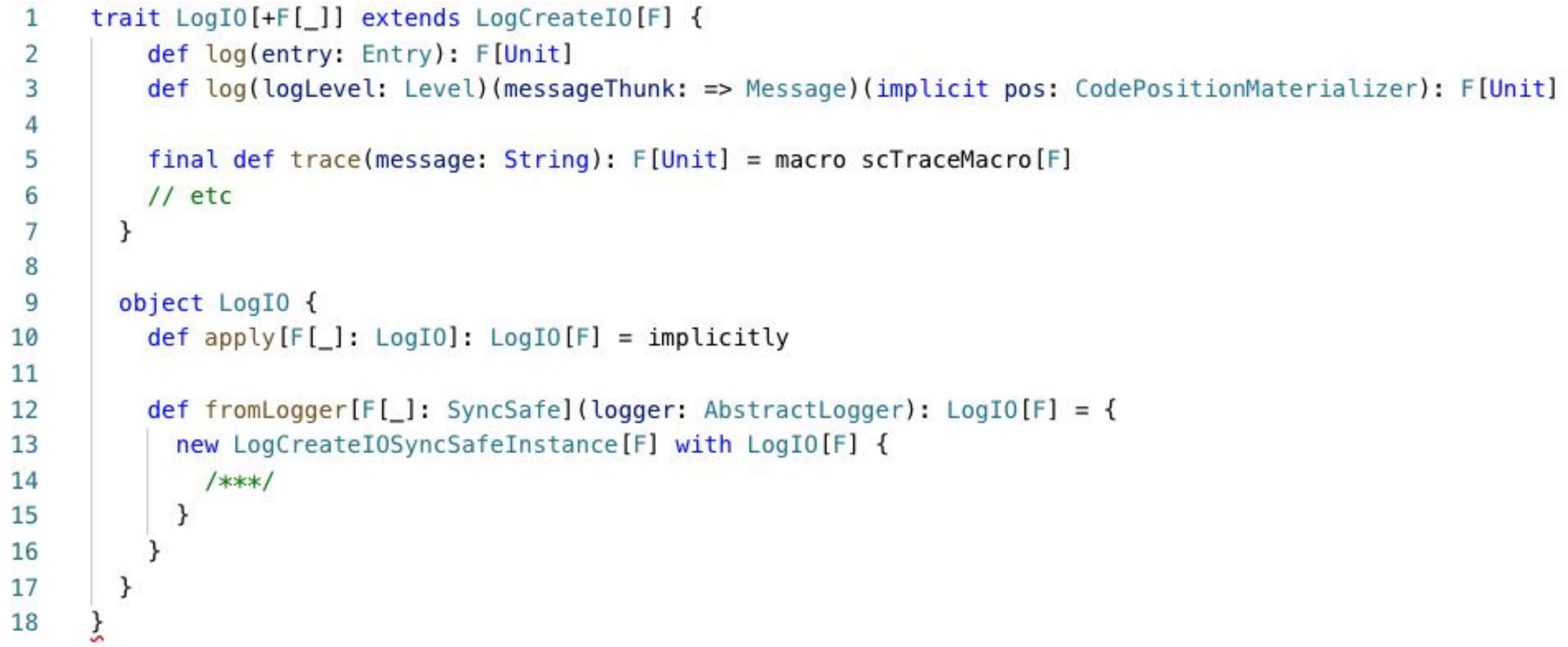


```
libraryDependencies ++= Seq(
  "com.github.pshirshov.izumi.r2" % "logstage-sink-slf4j_2.12" % "0.6.34"
)
```

Out of box provisioning

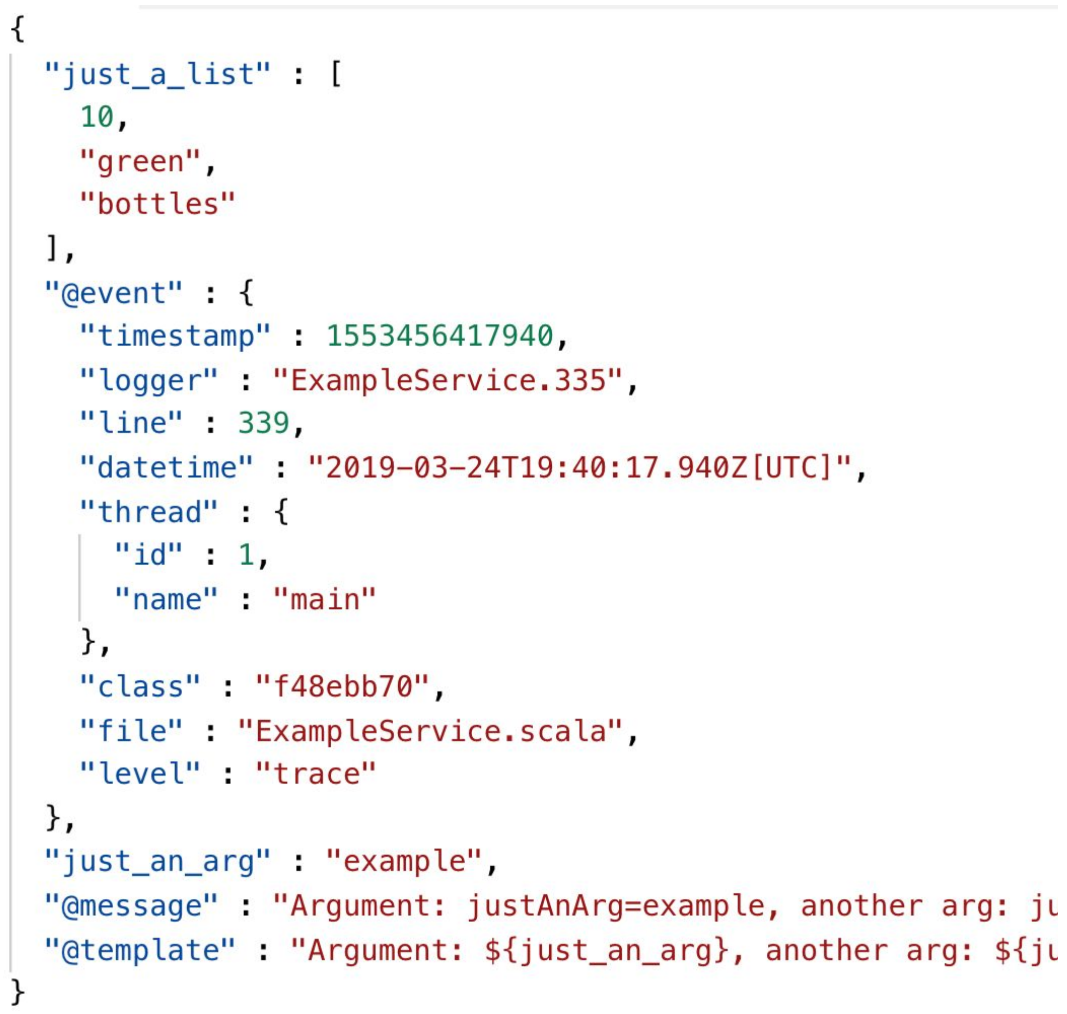
- Console sink
- File sink
- Asynchronous sink (single worker thread at the moment)
- String and Json rendering

Effectful adapters for ZIO, Cats, Monix



```scala
trait LogIO[+F[_]] extends LogCreateIO[F] {
    def log(entry: Entry): F[Unit]
    def log(logLevel: Level)(messageThunk: => Message)(implicit pos: CodePositionMaterializer): F[Unit]

    final def trace(message: String): F[Unit] = macro scTraceMacro[F]
    // etc
  }

  object LogIO {
    def apply[F[_]: LogIO]: LogIO[F] = implicitly

    def fromLogger[F[_]: SyncSafe](logger: AbstractLogger): LogIO[F] = {
      new LogCreateIOSyncSafeInstance[F] with LogIO[F] {
        /***/
      }
    }
  }
}
```

Automatic structure identifiers



```
{
  "just_a_list" : [
    10,
    "green",
    "bottles"
  ],
  "@event" : {
    "timestamp" : 1553456417940,
    "logger" : "ExampleService.335",
    "line" : 339,
    "datetime" : "2019-03-24T19:40:17.940Z[UTC]",
    "thread" : {
      "id" : 1,
      "name" : "main"
    },
    "class" : "f48ebb70",
    "file" : "ExampleService.scala",
    "level" : "trace"
  },
  "just_an_arg" : "example",
  "@message" : "Argument: justAnArg=example, another arg: ju
  "@template" : "Argument: ${just_an_arg}, another arg: ${ju
}
```

If you have a structured DB as a storage, you can querying your logs with the same structure

Template as an identifier

Let’s dive into coding

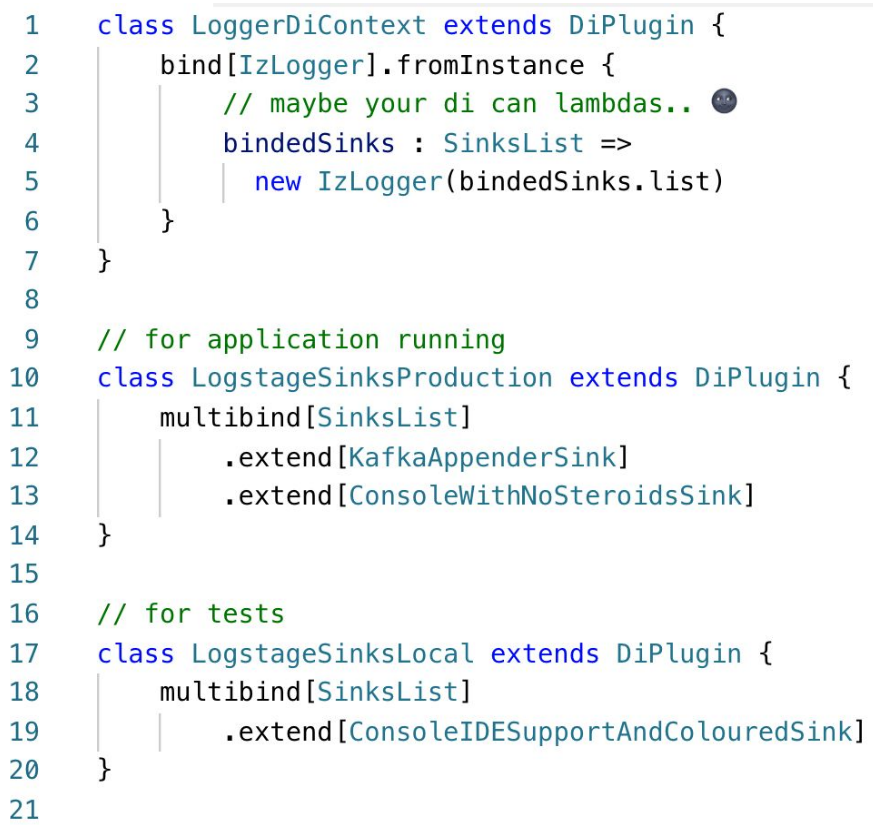
Almost no dependencies

Clean, neat, no singletons (except slf4j interop), which may impact on working of systems with isolated classloaders

Modular

```scala
object Test extends App {
    // own policies
    val fileRenderingPolicy : RenderingPolicy = ???
    val consoleRenderingPolicy : RenderingPolicy = ???

    // own sinks
    val fileSink = new FileSink(fileSink, fileRotation)
    val consoleSink = new ConsoleSink(consoleRenderingPolicy)

    // own rotation settings
    val fileRotation : FileRotation = ???
    // own router

    val yourRouter : LogRouter = ???

    // enjoy your settings
    val logger = IzLogger(sinks = List(fileSink, consoleSink), router = yourRouter)
}
```

DI ready



```scala
class LoggerDiContext extends DiPlugin {
    bind[IzLogger].fromInstance {
        // maybe your di can lambdas.. 🌚
        bindedSinks : SinksList =>
          new IzLogger(bindedSinks.list)
    }
}

// for application running
class LogstageSinksProduction extends DiPlugin {
    multibind[SinksList]
        .extend[KafkaAppenderSink]
        .extend[ConsoleWithNoSteroidsSink]
}

// for tests
class LogstageSinksLocal extends DiPlugin {
    multibind[SinksList]
        .extend[ConsoleIDESupportAndColouredSink]
}
```

Two separate settings set.

Isn’t cool? 😍

DIStage out of box

```
libraryDependencies ++= Seq(
    "com.github.pshirshov.izumi.r2" % "logstage-di_2.12" % "0.6.34"
)
```

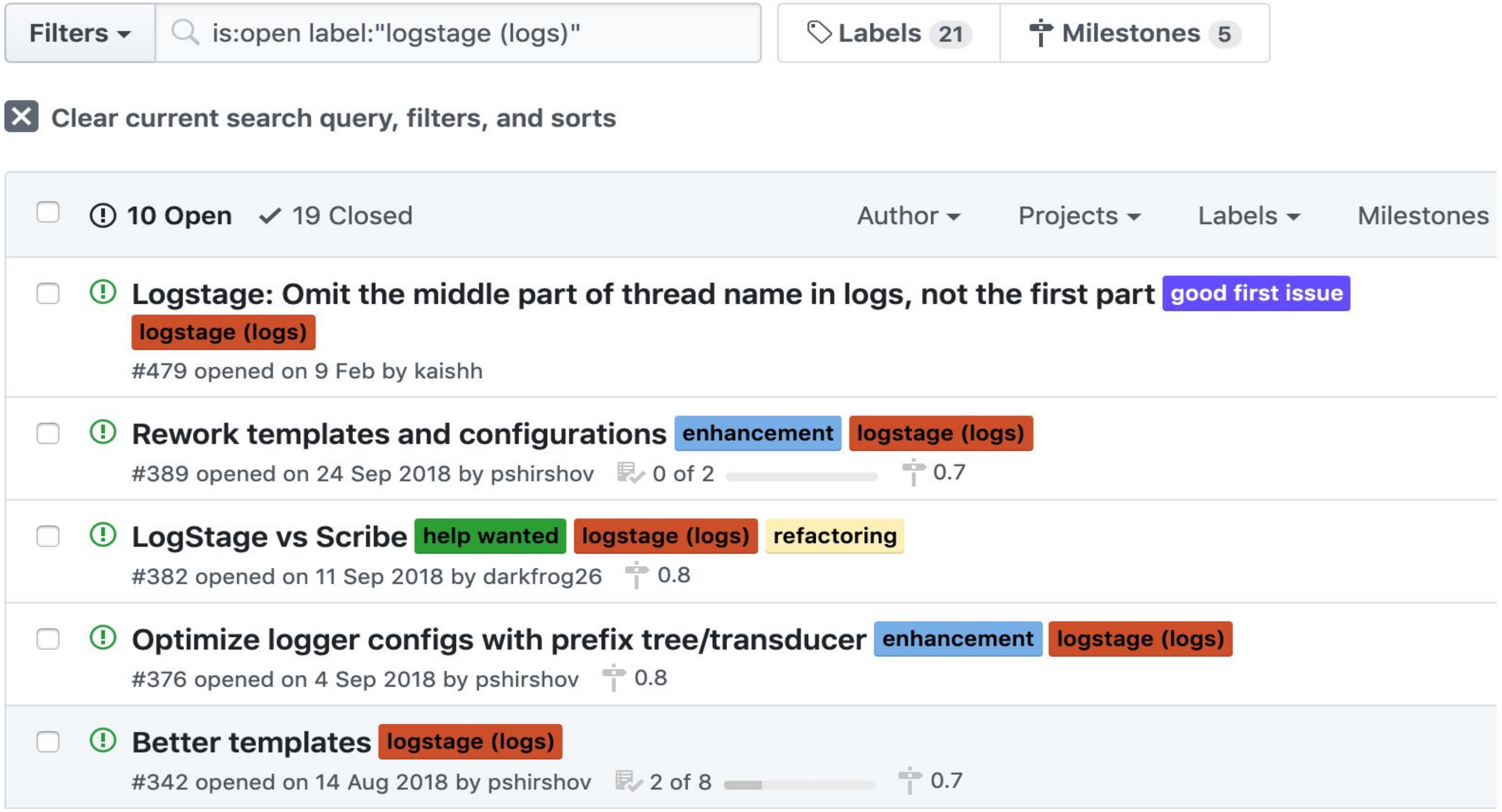
Comparison with popular frameworks

	Logstage	Scribe	Airframe	Logback + SLF4j	Scala Logging
Structured	✓	×	×	×	×
SOLID rules	✓	×	×	×	×
No singletons	✓	×	×	×	×
Modularity	✓	×	×	×	×
Asynchronous	✓	✓	✓	✓	✓
DI-readiness	✓	×	×	×	×
Colourful	✓	×	✓	✓	×
Dynamic Context	✓	×	×	✓	✓
File rotation	✓	✓	✓	✓	✓

Plans to work with

- Rethinking of rendering policy
- Better configuration
- Integrations with Logback, Azure, ElasticSearch, Kafka
- Profiling and optimization performance

Welcome to contribution 🤓



Filters ▾ is:open label:"logstage (logs)" Labels 21 Milestones 5

Clear current search query, filters, and sorts

10 Open ✓ 19 Closed Author ▾ Projects ▾ Labels ▾ Milestones

Logstage: Omit the middle part of thread name in logs, not the first part good first issue logstage (logs)
#479 opened on 9 Feb by kaishh

Rework templates and configurations enhancement logstage (logs)
#389 opened on 24 Sep 2018 by pshirshov 0 of 2 0.7

LogStage vs Scribe help wanted logstage (logs) refactoring
#382 opened on 11 Sep 2018 by darkfrog26 0.8

Optimize logger configs with prefix tree/transducer enhancement logstage (logs)
#376 opened on 4 Sep 2018 by pshirshov 0.8

Better templates logstage (logs)
#342 opened on 14 Aug 2018 by pshirshov 2 of 8 0.7

Thank you for listening!

https://github.com/7mind

https://ratoshniuk.github.io/

https://github.com/ratoshniuk/scalaua-2019