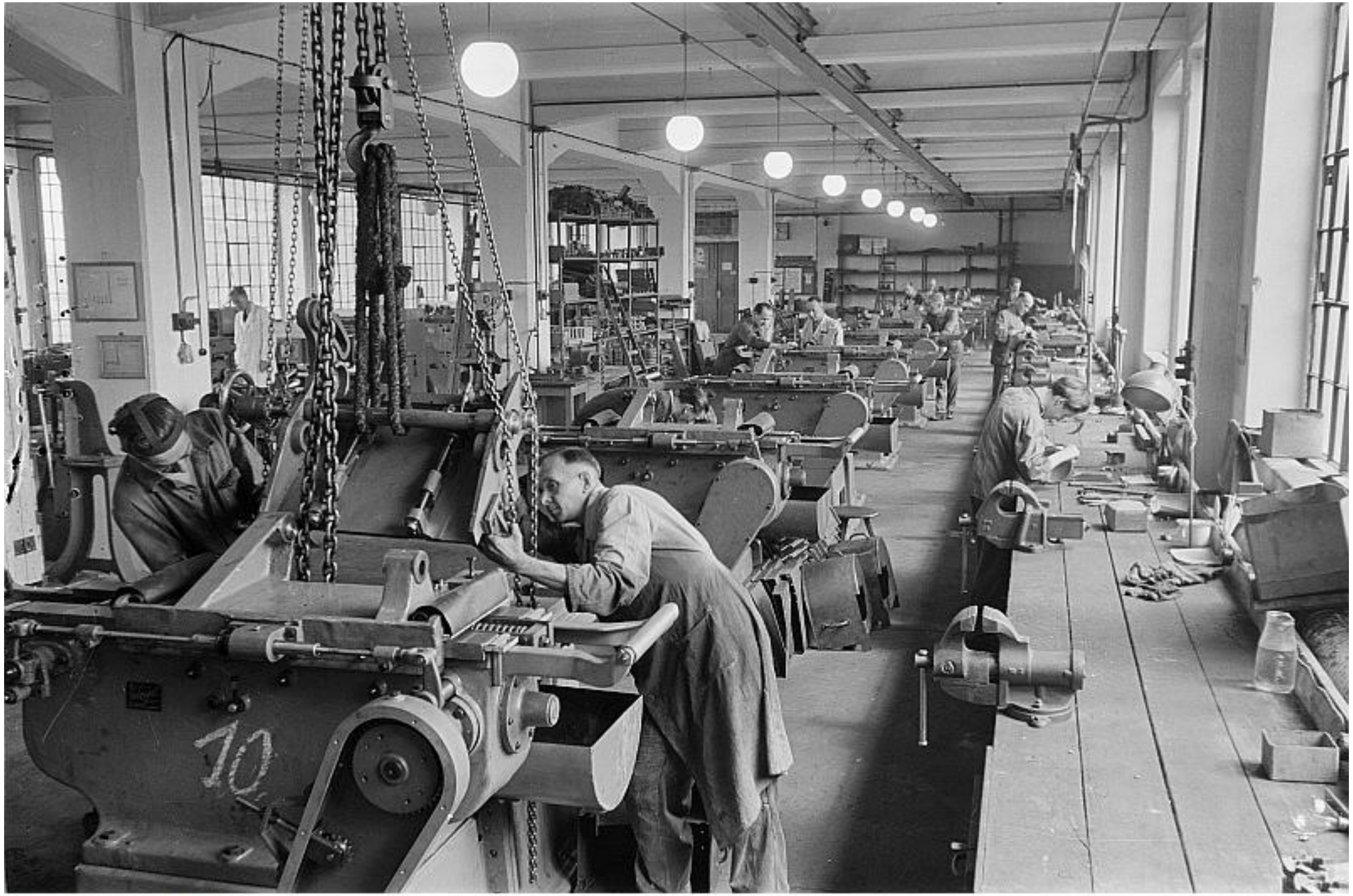
React.sphere.it

Alpakka – a new world of Connectors for Reactive Enterprise Integration

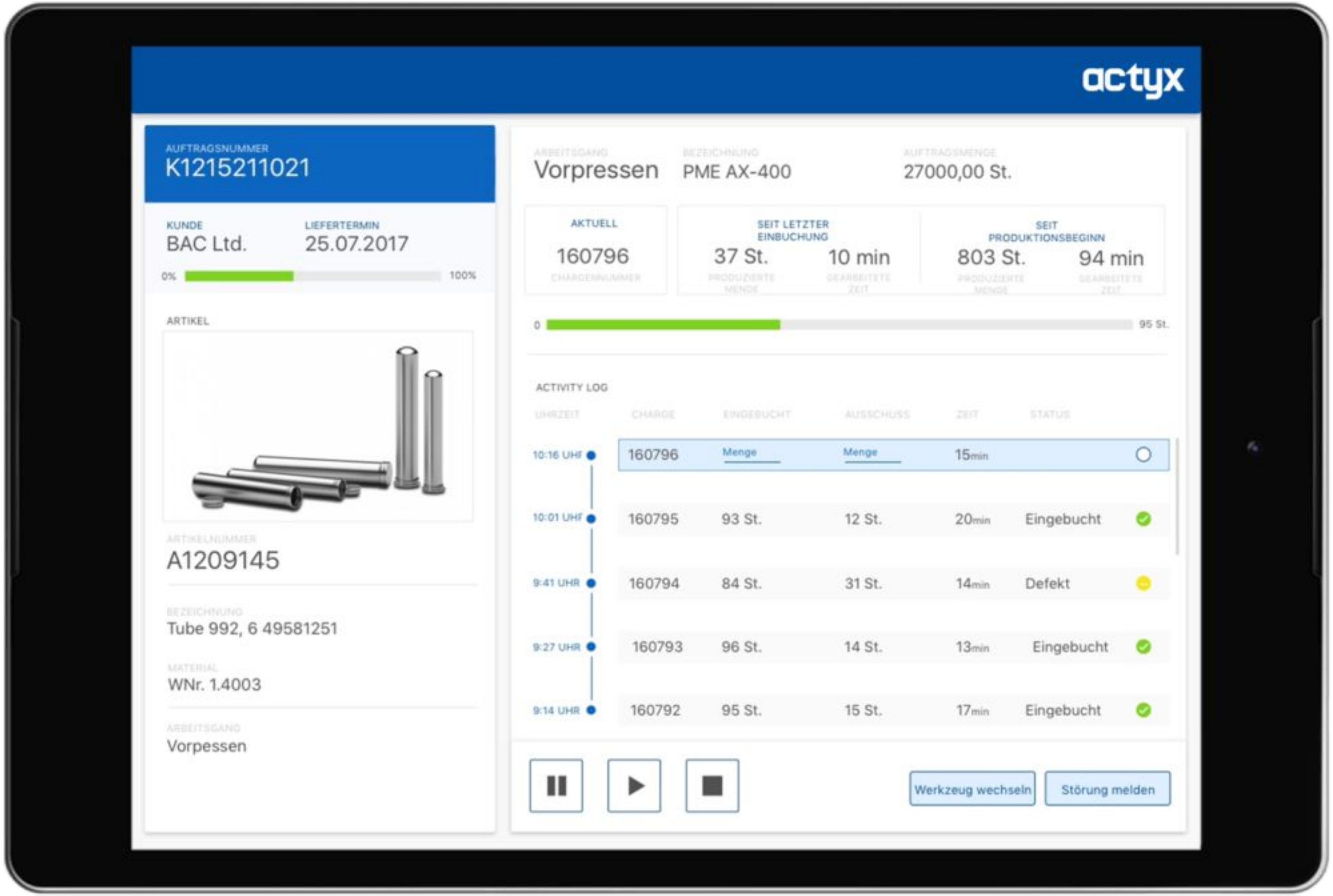
Jan Pustelnik

@gosubpl

actyx



Quelle: Deutsche Fotothek



actyx

AUFTRAGSNUMMER
K1215211021

KUNDE
BAC Ltd.

LIEFERTERMIN
25.07.2017

0% 100%

ARTIKEL

ARTIKELNUMMER
A1209145

BEZEICHNUNG
Tube 992, 6 49581251

MATERIAL
WNr. 1.4003

ARBEITSGANG
Vorpessen

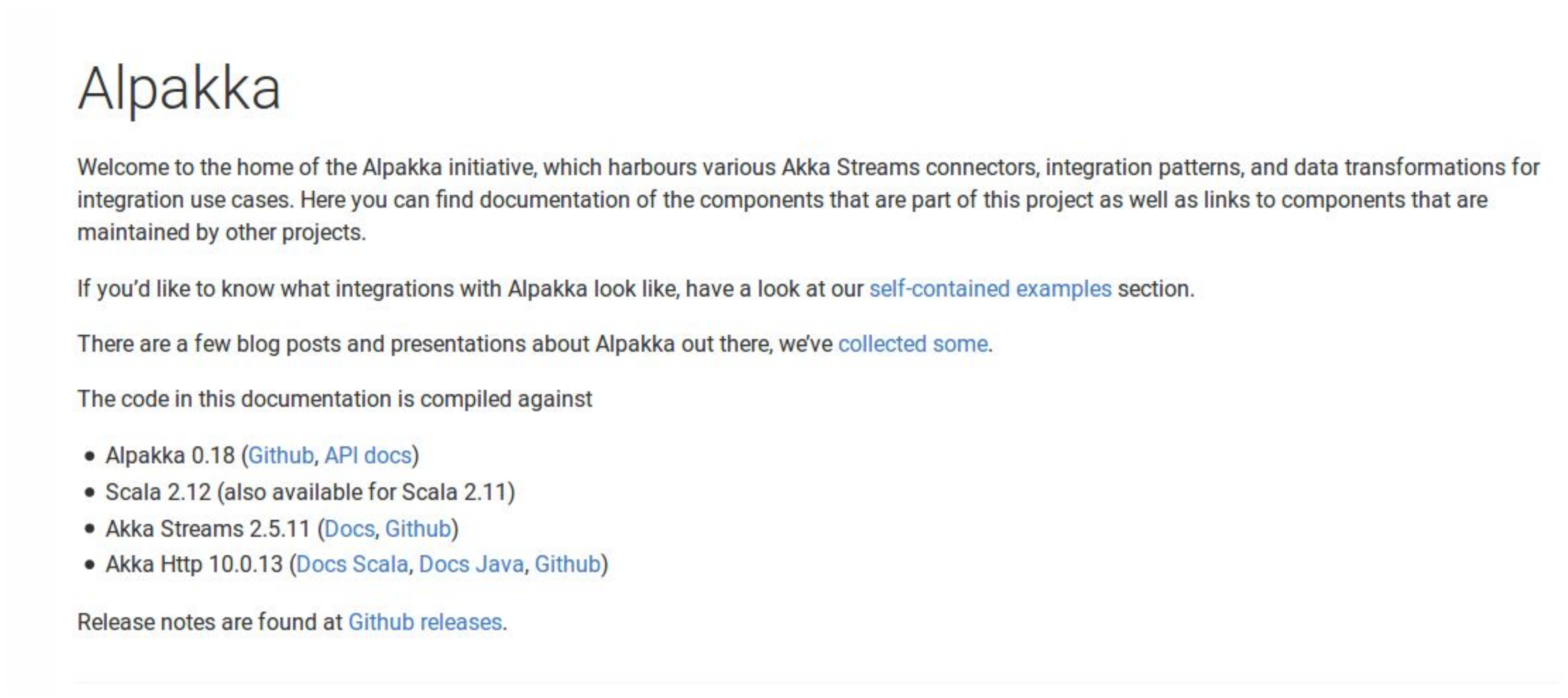
ARBEITSGANG	BEZEICHNUNG	AUFTRAGSMENGE
Vorpressen	PME AX-400	27000,00 St.

AKTUELL	SEIT LETZTER EINBUCHUNG		SEIT PRODUKTIONSBEGINN	
160796	37 St.	10 min	803 St.	94 min
CHARGENNUMMER	PRODUZIERTE MENGE	GEARBEITETE ZEIT	PRODUZIERTE MENGE	GEARBEITETE ZEIT

0 95 St.

ACTIVITY LOG

UHRZEIT	CHARGE	EINGEBUCHT	AUSSCHUSS	ZEIT	STATUS	
10:16 UHR	160796	Menge	Menge	15min		○
10:01 UHR	160795	93 St.	12 St.	20min	Eingebucht	✓
9:41 UHR	160794	84 St.	31 St.	14min	Defekt	●
9:27 UHR	160793	96 St.	14 St.	13min	Eingebucht	✓
9:14 UHR	160792	95 St.	15 St.	17min	Eingebucht	✓



Alpakka

Welcome to the home of the Alpakka initiative, which harbours various Akka Streams connectors, integration patterns, and data transformations for integration use cases. Here you can find documentation of the components that are part of this project as well as links to components that are maintained by other projects.

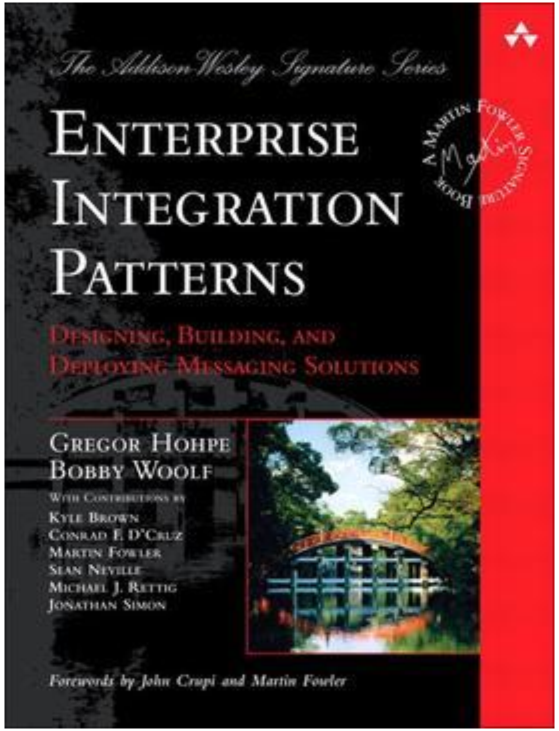
If you'd like to know what integrations with Alpakka look like, have a look at our self-contained examples section.

There are a few blog posts and presentations about Alpakka out there, we've collected some.

The code in this documentation is compiled against

- Alpakka 0.18 (Github, API docs)
- Scala 2.12 (also available for Scala 2.11)
- Akka Streams 2.5.11 (Docs, Github)
- Akka Http 10.0.13 (Docs Scala, Docs Java, Github)

Release notes are found at Github releases.



The Addison-Wesley Signature Series

A Martin Fowler Signature Book

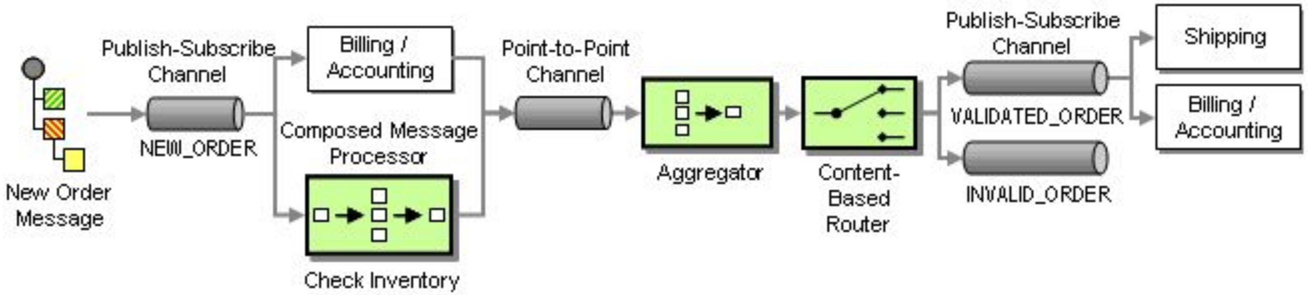
Enterprise Integration Patterns

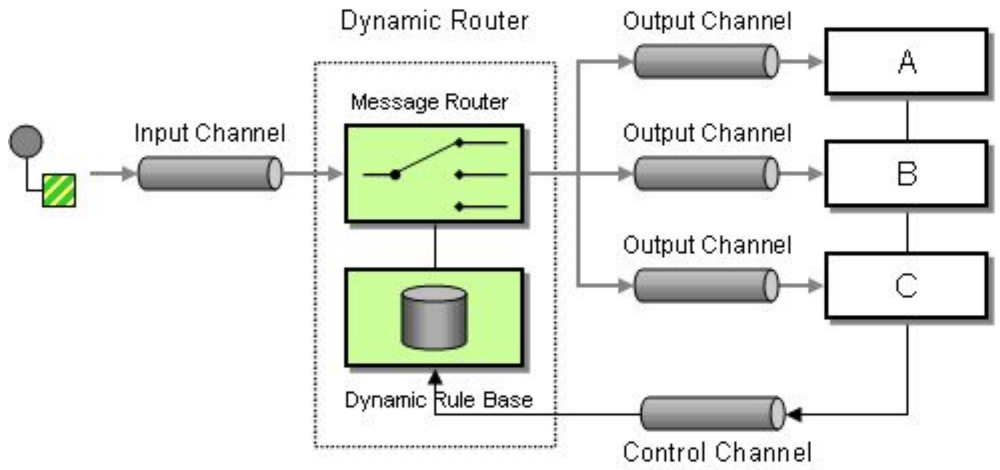
Designing, Building, and Deploying Messaging Solutions

Gregor Hohpe
Bobby Woolf

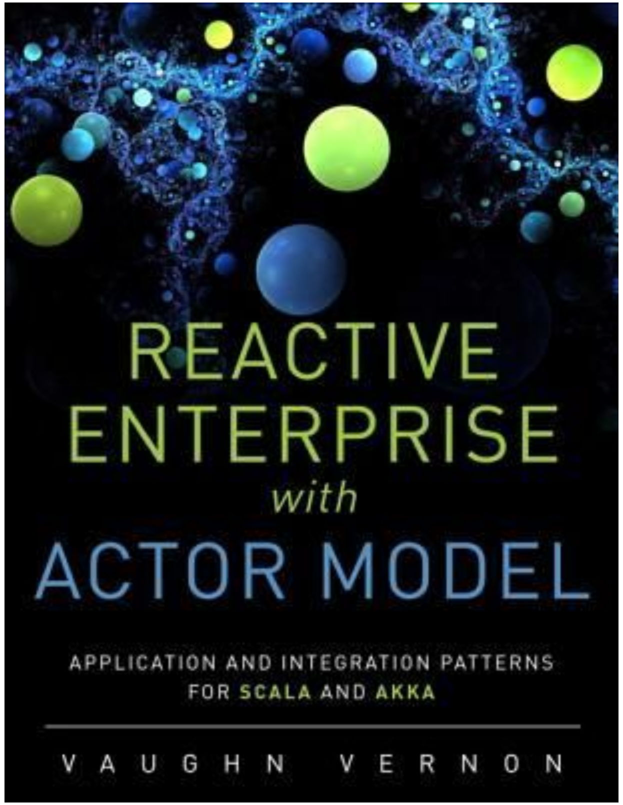
With Contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon

Forewords by John Crupi and Martin Fowler


New Order Message
Publish-Subscribe Channel
NEW_ORDER
Billing / Accounting
Composed Message Processor
Check Inventory
Point-to-Point Channel
Aggregator
Content-Based Router
Publish-Subscribe Channel
VALIDATED_ORDER
INVALID_ORDER
Shipping
Billing / Accounting


Dynamic Router
Output Channel
A
Message Router
Input Channel
Output Channel
B
Output Channel
C
Dynamic Rule Base
Control Channel

akka



REACTIVE ENTERPRISE *with* ACTOR MODEL

APPLICATION AND INTEGRATION PATTERNS FOR SCALA AND AKKA

VAUGHN VERNON

PartitionHub *stage*

fineartamerica

ART HOME TECH Search SELL CREATE DISCOVER

Shop Collections Subjects Community About Help Sign In Join

Back to Gliesian LLC | Shop / Home Decor / Shower Curtains

‹ PREV | NEXT › 2 0 1

Apache Camel Components Poster Shower Curtain

by **Gliesian LLC**

$120.00 ADD TO CART

IMAGE SIZE

DESCRIPTION

Our shower curtains are made from 100% polyester fabric and include 12 holes at the top of the curtain for simple hanging from your own shower curtain rings. The total dimensions of each shower curtain are 71" wide x 74" tall.

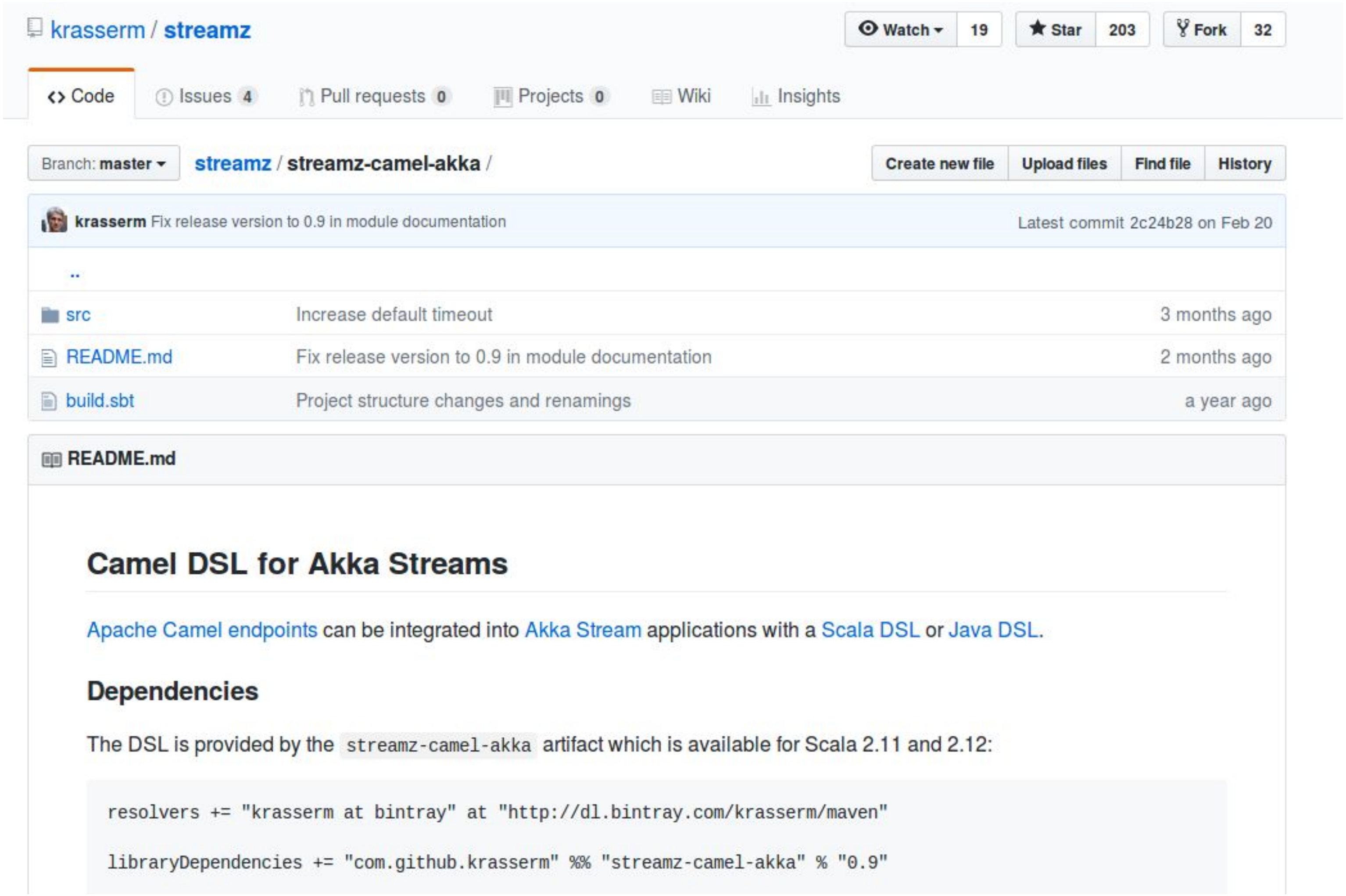
SHIPS WITHIN

2 - 3 business days

```scala
import akka.camel.{ CamelMessage, Consumer }

class MyEndpoint extends Consumer {
  def endpointUri = "jetty:http://localhost:8877/example"

  def receive = {
    case msg: CamelMessage ⇒ { /* ... */ }
    case _                 ⇒ { /* ... */ }
  }
}
```



krasserm Fix release version to 0.9 in module documentation — Latest commit 2c24b28 on Feb 20

..		
src	Increase default timeout	3 months ago
README.md	Fix release version to 0.9 in module documentation	2 months ago
build.sbt	Project structure changes and renamings	a year ago

README.md

Camel DSL for Akka Streams

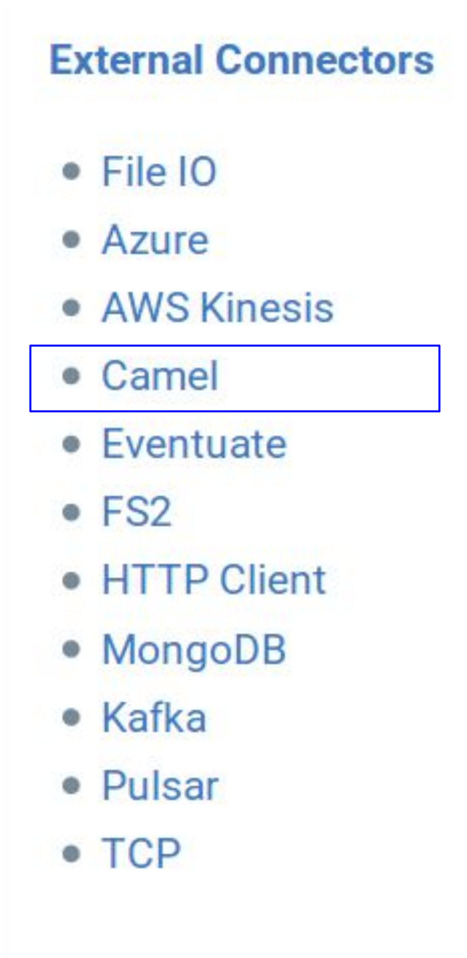
Apache Camel endpoints can be integrated into Akka Stream applications with a Scala DSL or Java DSL.

Dependencies

The DSL is provided by the `streamz-camel-akka` artifact which is available for Scala 2.11 and 2.12:

```
resolvers += "krasserm at bintray" at "http://dl.bintray.com/krasserm/maven"

libraryDependencies += "com.github.krasserm" %% "streamz-camel-akka" % "0.9"
```



External Connectors

- File IO
- Azure
- AWS Kinesis
- Camel
- Eventuate
- FS2
- HTTP Client
- MongoDB
- Kafka
- Pulsar
- TCP

Connectors

- AMQP Connector
- Apache Geode connector
- Apache Solr Connector
- AWS DynamoDB Connector
- AWS Kinesis Connector
- AWS Lambda Connector
- AWS S3 Connector
- AWS SNS Connector
- AWS SQS Connector
- Azure Storage Queue Connector
- Cassandra Connector
- Elasticsearch Connector
- File Connectors
- FTP Connector
- Google Cloud Pub/Sub
- Google Firebase Cloud Messaging
- HBase connector
- IronMq Connector
- JMS Connector
- MongoDB Connector
- MQTT Connector
- OrientDB Connector
- Server-sent Events (SSE) Connector
- Slick (JDBC) Connector
- Spring Web
- Unix Domain Socket Connector

Example

JMS Connector

The JMS connector provides Akka Stream sources and sinks to connect to JMS providers.

Reported issues

Tagged issues at Github

Artifacts

sbt **Maven** **Gradle**

```
libraryDependencies += "com.lightbend.akka" %% "akka-stream-alpakka-jms" % "0.18"
libraryDependencies += "javax.jms" % "jms" % "1.1"
```

```
val jmsSource: Source[String, KillSwitch] =
  JmsConsumer.textSource(
    JmsConsumerSettings(connectionFactory).withBufferSize(10).withQueue("test")
  )



val runningSource = jmsSource
  .map(ByteString(_))
  .zip(Source.fromIterator(() => Iterator.from(0)))
  .mapAsyncUnordered(parallelism = 5) { case (byteStr, number) =>
    Source
      .single(byteStr)
      .runWith(FileIO.toPath(Paths.get(s"target/out-$number.txt")))
  }
  .toMat(Sink.ignore)(Keep.left)
  .run()
```

https://developer.lightbend.com/docs/alpakka/current/examples/jms-samples.html

```
val jmsSource: Source[String, KillSwitch] =
  JmsConsumer.textSource(
    JmsConsumerSettings(connectionFactory).withBufferSize(10).withQueue("test")
  )



val runningSource = jmsSource
  .map(ByteString(_))
  .zip(Source.fromIterator(() => Iterator.from(0)))
  .mapAsyncUnordered(parallelism = 5) { case (byteStr, number) =>
    Source
      .single(byteStr)
      .runWith(FileIO.toPath(Paths.get(s"target/out-$number.txt")))
  }
  .toMat(Sink.ignore)(Keep.left)
  .run()
```

https://developer.lightbend.com/docs/alpakka/current/examples/jms-samples.html

```
val jmsSource: Source[String, KillSwitch] =
  JmsConsumer.textSource(
    JmsConsumerSettings(connectionFactory).withBufferSize(10).withQueue("test")
  )



val runningSource = jmsSource
  .map(ByteString(_))
  .zip(Source.fromIterator(() => Iterator.from(0)))
  .mapAsyncUnordered(parallelism = 5) { case (byteStr, number) =>
    Source
      .single(byteStr)
      .runWith(FileIO.toPath(Paths.get(s"target/out-$number.txt")))
  }
  .toMat(Sink.ignore)(Keep.left)
  .run()
```

https://developer.lightbend.com/docs/alpakka/current/examples/jms-samples.html

```
val jmsSource: Source[String, KillSwitch] =
  JmsConsumer.textSource(
    JmsConsumerSettings(connectionFactory).withBufferSize(10).withQueue("test")
  )



val runningSource = jmsSource
  .map(ByteString(_))
  .zip(Source.fromIterator(() => Iterator.from(0)))
  .mapAsyncUnordered(parallelism = 5) { case (byteStr, number) =>
    Source
      .single(byteStr)
      .runWith(FileIO.toPath(Paths.get(s"target/out-$number.txt")))
  }
  .toMat(Sink.ignore)(Keep.left)
  .run()
```

https://developer.lightbend.com/docs/alpakka/current/examples/jms-samples.html

```
val jmsSource: Source[String, KillSwitch] =
  JmsConsumer.textSource(
    JmsConsumerSettings(connectionFactory).withBufferSize(10).withQueue("test")
  )



val runningSource = jmsSource
  .map(ByteString(_))
  .zip(Source.fromIterator(() => Iterator.from(0)))
  .mapAsyncUnordered(parallelism = 5) { case (byteStr, number) =>
    Source
      .single(byteStr)
      .runWith(FileIO.toPath(Paths.get(s"target/out-$number.txt")))
  }
  .toMat(Sink.ignore)(Keep.left)
  .run()
```

https://developer.lightbend.com/docs/alpakka/current/examples/jms-samples.html

```
val jmsSource: Source[String, KillSwitch] =
  JmsConsumer.textSource(
    JmsConsumerSettings(connectionFactory).withBufferSize(10).withQueue("test")
  )



val runningSource = jmsSource
  .map(ByteString(_))
  .zip(Source.fromIterator(() => Iterator.from(0)))
  .mapAsyncUnordered(parallelism = 5) { case (byteStr, number) =>
    Source
      .single(byteStr)
      .runWith(FileIO.toPath(Paths.get(s"target/out-$number.txt")))
  }
  .toMat(Sink.ignore)(Keep.left)
  .run()
```

https://developer.lightbend.com/docs/alpakka/current/examples/jms-samples.html

```
val jmsSource: Source[String, KillSwitch] =
  JmsConsumer.textSource(
    JmsConsumerSettings(connectionFactory).withBufferSize(10).withQueue("test")
  )
```

```
val runningSource = jmsSource
  .map(ByteString(_))
  .zip(Source.fromIterator(() => Iterator.from(0)))
  .mapAsyncUnordered(parallelism = 5) { case (byteStr, number) =>
    Source
      .single(byteStr)
      .runWith(FileIO.toPath(Paths.get(s"target/out-$number.txt")))
  }
  .toMat(Sink.ignore)(Keep.left)
  .run()
```

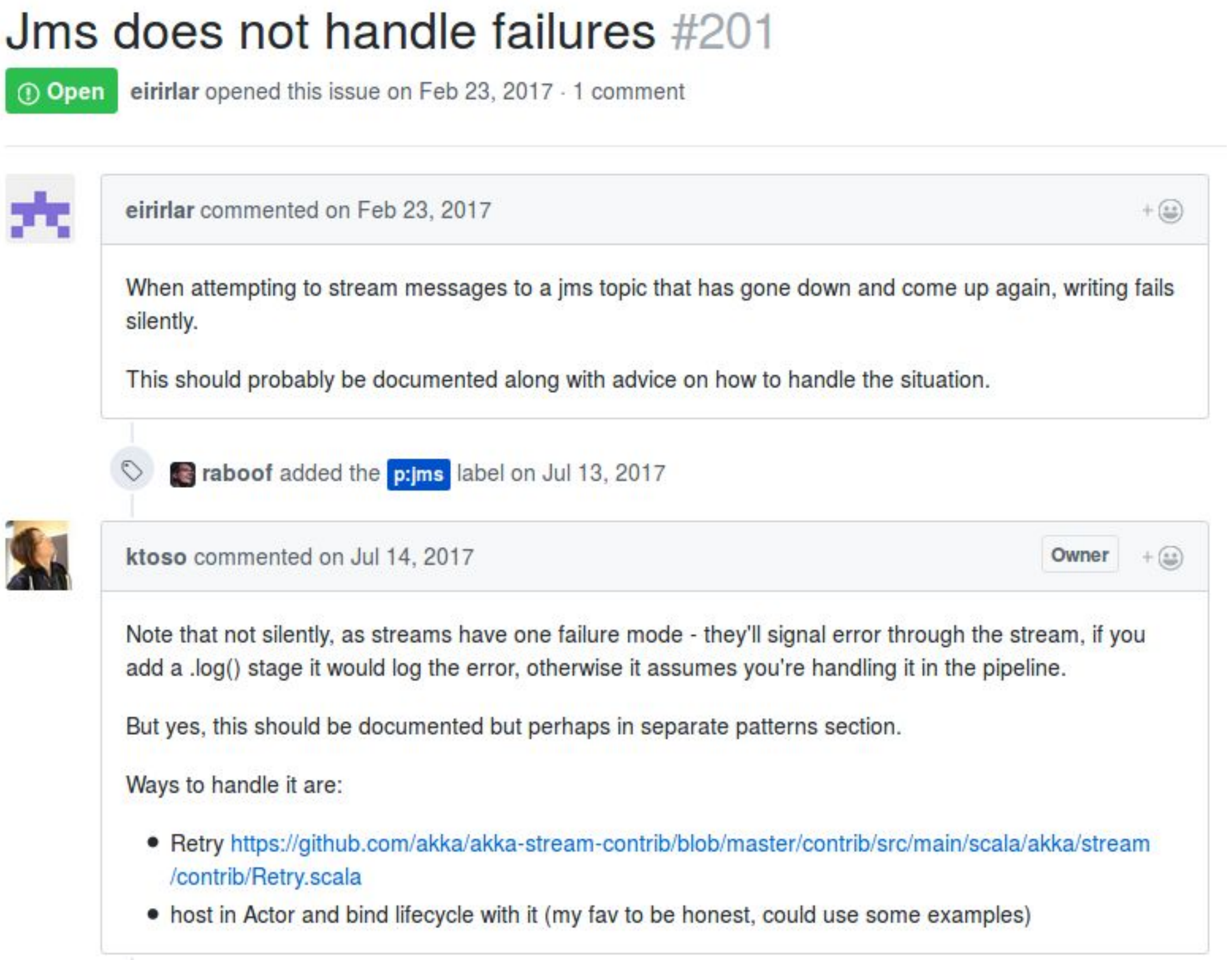
https://developer.lightbend.com/docs/alpakka/current/examples/jms-samples.html

```
val jmsSource: Source[String, KillSwitch] =
  JmsConsumer.textSource(
    JmsConsumerSettings(connectionFactory).withBufferSize(10).withQueue("test")
  )



val runningSource = jmsSource
  .map(ByteString(_))
  .zip(Source.fromIterator(() => Iterator.from(0)))
  .mapAsyncUnordered(parallelism = 5) { case (byteStr, number) =>
    Source
      .single(byteStr)
      .runWith(FileIO.toPath(Paths.get(s"target/out-$number.txt")))
  }
  .toMat(Sink.ignore)(Keep.left)
  .run()
```

https://developer.lightbend.com/docs/alpakka/current/examples/jms-samples.html



Jms does not handle failures #201

Open eirirlar opened this issue on Feb 23, 2017 · 1 comment

eirirlar commented on Feb 23, 2017

When attempting to stream messages to a jms topic that has gone down and come up again, writing fails silently.

This should probably be documented along with advice on how to handle the situation.

raboof added the p:jms label on Jul 13, 2017

ktoso commented on Jul 14, 2017 Owner

Note that not silently, as streams have one failure mode - they'll signal error through the stream, if you add a .log() stage it would log the error, otherwise it assumes you're handling it in the pipeline.

But yes, this should be documented but perhaps in separate patterns section.

Ways to handle it are:

- Retry https://github.com/akka/akka-stream-contrib/blob/master/contrib/src/main/scala/akka/stream/contrib/Retry.scala
- host in Actor and bind lifecycle with it (my fav to be honest, could use some examples)

Backpressure

mongoDB®

MongoDB Connector

The MongoDB connector allows you to read and save documents. You can query as a stream of documents from MongoSource or update documents in a collection with MongoSink.

This connector is based off the mongo-scala-driver and does not have a java interface. It supports driver macros and codec allowing to read or write scala case class objects.

Reported issues

Tagged issues at Github

Artifacts

sbt | **Maven** | **Gradle**

```
libraryDependencies += "com.lightbend.akka" %% "akka-stream-alpakka-mongodb" % "0.18"
```

```
private val client =
MongoClient(s"mongodb://localhost:27017")
private val db = client.getDatabase("alpakka-mongo")
private val numbersColl = db.getCollection("numbers")

val source: Source[Document, NotUsed] =
  MongoSource(numbersColl.find())
```

```
ObservableToPublisher[T](
    observable: mongoDB.Observable[T]
)
```

```
object MongoSource {

  def apply[T](query: Observable[T]):
    Source[T, NotUsed] =

Source.fromPublisher(ObservableToPublisher(query))

}
```

Cassandra Connector

The Cassandra connector allows you to read and write to Cassandra. You can query a stream of rows from CassandraSource or use prepared statements to insert or update with CassandraFlow or CassandraSink.

Unlogged batches are also supported.

Reported issues

Tagged issues at Github

Artifacts

sbt **Maven** **Gradle**

```
libraryDependencies += "com.lightbend.akka" %% "akka-stream-alpakka-cassandra" % "0.18"
```

```
implicit val session = Cluster.builder
  .addContactPoint("127.0.0.1")
  .withPort(9042)
  .build
  .connect()

val stmt = new SimpleStatement(
   s"SELECT * FROM $keyspaceName.test"
   )
   .setFetchSize(20)

val rows = CassandraSource(stmt).runWith(Sink.seq)
```

```
implicit val session = Cluster.builder
  .addContactPoint("127.0.0.1")
  .withPort(9042)
  .build
  .connect()

val stmt = new SimpleStatement(
    s"SELECT * FROM $keyspaceName.test"
    )
    .setFetchSize(20)

val rows = CassandraSource(stmt).runWith(Sink.seq)
```

```
implicit val session = Cluster.builder
  .addContactPoint("127.0.0.1")
  .withPort(9042)
  .build
  .connect()

val stmt = new SimpleStatement(
    s"SELECT * FROM $keyspaceName.test"
    )
    .setFetchSize(20)

val rows = CassandraSource(stmt).runWith(Sink.seq)
```

```
      new OutHandler {
         override def onPull(): Unit = {
           implicit val ec = materializer.executionContext

           maybeRs match {
             case Some(rs) if rs.getAvailableWithoutFetching > 0
=> push(out, rs.one())
             case Some(rs) if rs.isExhausted => completeStage()
             case Some(rs) =>
               // fetch next page
               val futRs = rs.fetchMoreResults().asScala()
               futRs.onComplete(futFetchedCallback.invoke)
             case None => () // doing nothing, waiting for futRs
in preStart() to be completed
           }
         }
      }
      new OutHandler {
         override def onPull(): Unit = {
           implicit val ec = materializer.executionContext

           maybeRs match {
             case Some(rs) if rs.getAvailableWithoutFetching > 0
=> push(out, rs.one())
             case Some(rs) if rs.isExhausted => completeStage()
             case Some(rs) =>
               // fetch next page
               val futRs = rs.fetchMoreResults().asScala()
               futRs.onComplete(futFetchedCallback.invoke)
             case None => () // doing nothing, waiting for futRs
in preStart() to be completed
           }
         }
      }
      new OutHandler {
         override def onPull(): Unit = {
           implicit val ec = materializer.executionContext

           maybeRs match {
             case Some(rs) if rs.getAvailableWithoutFetching > 0
=> push(out, rs.one())
             case Some(rs) if rs.isExhausted => completeStage()
             case Some(rs) =>
               // fetch next page
               val futRs = rs.fetchMoreResults().asScala()
               futRs.onComplete(futFetchedCallback.invoke)
             case None => () // doing nothing, waiting for futRs
in preStart() to be completed
           }
         }
       }
      new OutHandler {
         override def onPull(): Unit = {
           implicit val ec = materializer.executionContext

           maybeRs match {
             case Some(rs) if rs.getAvailableWithoutFetching > 0
=> push(out, rs.one())
             case Some(rs) if rs.isExhausted => completeStage()
             case Some(rs) =>
               // fetch next page
               val futRs = rs.fetchMoreResults().asScala()
               futRs.onComplete(futFetchedCallback.invoke)
             case None => () // doing nothing, waiting for futRs
in preStart() to be completed
           }
         }
      }
      new OutHandler {
         override def onPull(): Unit = {
           implicit val ec = materializer.executionContext

           maybeRs match {
             case Some(rs) if rs.getAvailableWithoutFetching > 0
=> push(out, rs.one())
             case Some(rs) if rs.isExhausted => completeStage()
             case Some(rs) =>
               // fetch next page
               val futRs = rs.fetchMoreResults().asScala()
               futRs.onComplete(futFetchedCallback.invoke)
             case None => () // doing nothing, waiting for futRs
in preStart() to be completed
           }
         }
       }
```

File Connectors

The File connectors provide additional connectors for filesystems complementing the sources and sinks for files already included in core Akka Streams (which can be found in akka.stream.scaladsl.FileIO)).

Reported issues

Tagged issues at Github

Artifacts

sbt Maven Gradle

```
libraryDependencies += "com.lightbend.akka" %% "akka-stream-alpakka-file" % "0.18"
```

Tailing a file into a stream

The `FileTailSource` starts at a given offset in a file and emits chunks of bytes until reaching the end of the file, it will then poll the file for changes and emit new changes as they are written to the file (unless there is backpressure).

```
val fs = FileSystems.getDefault
val lines: Source[String, NotUsed] =
   scaladsl.FileTailSource.lines(
  path = fs.getPath(path),
  maxLineSize = 8192,
  pollingInterval = 250.millis
)

lines.runForeach(line => System.out.println(line))
```

```scala
val fs = FileSystems.getDefault
val lines: Source[String, NotUsed] =
   scaladsl.FileTailSource.lines(
  path = fs.getPath(path),
  maxLineSize = 8192,
  pollingInterval = 250.millis
)

lines.runForeach(line => System.out.println(line))
```

```
val fs = FileSystems.getDefault
val lines: Source[String, NotUsed] =
   scaladsl.FileTailSource.lines(
  path = fs.getPath(path),
  maxLineSize = 8192,
  pollingInterval = 250.millis
)

lines.runForeach(line => System.out.println(line))
```

https://github.com/akka/alpakka/blob/master/file/src/main/java/akka/stream/alpakka/file/javadsl/FileTailSource.java#L78-L124

Pseudocode:

```
    onPull:
        Schedule a callback, trying to read a max
(maxLineSize) chunk from the input file

    Callback:
        Push out the contents of the buffer read in a callback
```

https://github.com/akka/alpakka/blob/master/file/src/main/java/akka/stream/alpakka/file/javadsl/FileTailSource.java#L78-L124

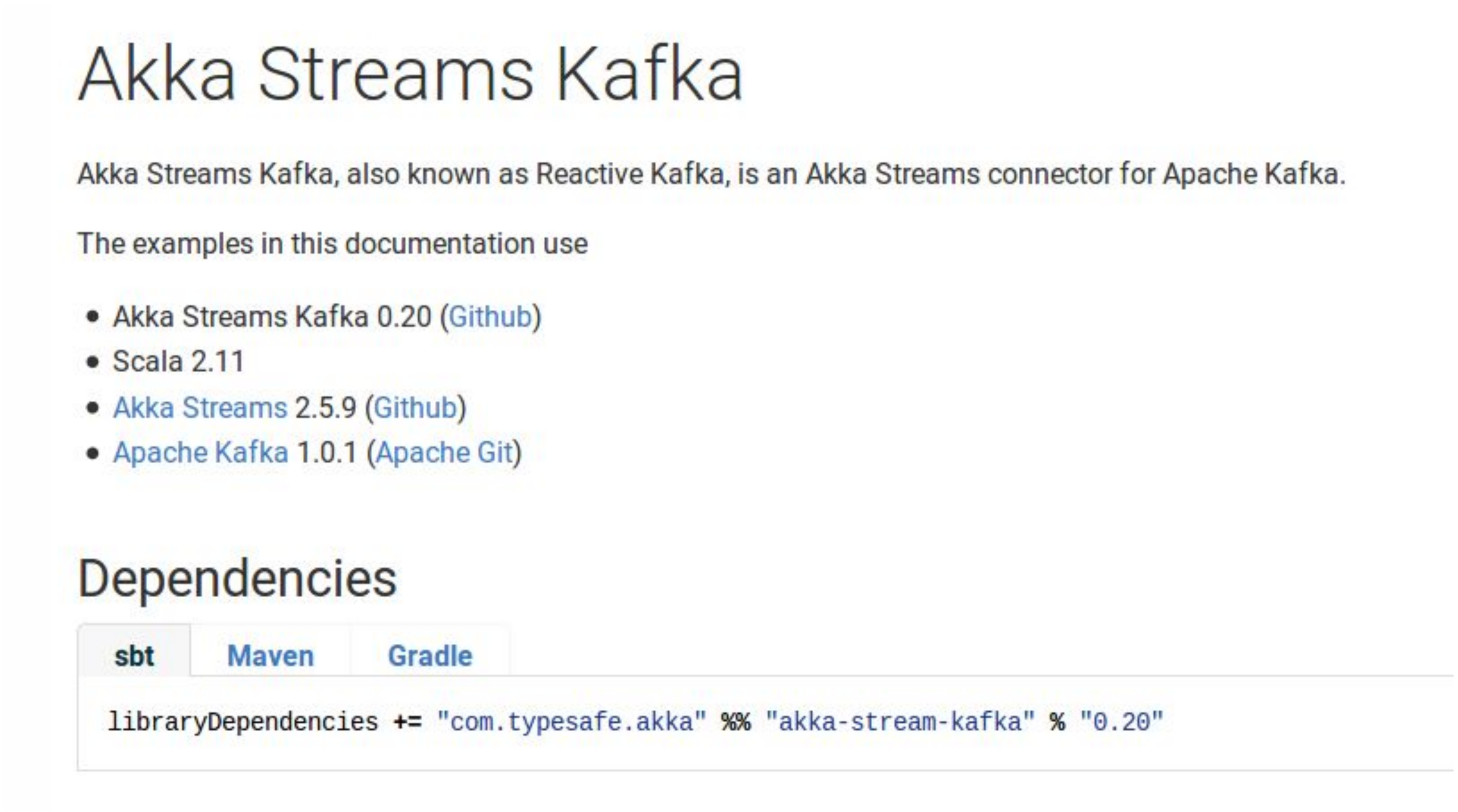
```
Pseudocode:
    onPull:
        Schedule a callback, trying to read a max
(maxLineSize) chunk from the input file

    Callback:
        Push out the contents of the buffer read in a callback
```

https://github.com/akka/alpakka/blob/master/file/src/main/java/akka/stream/alpakka/file/javadsl/FileTailSource.java#L78-L124

```
Pseudocode:
    onPull:
        Schedule a callback, trying to read a max
(maxLineSize) chunk from the input file

    Callback:
        Push out the contents of the buffer read in a callback
```



Akka Streams Kafka

Akka Streams Kafka, also known as Reactive Kafka, is an Akka Streams connector for Apache Kafka.

The examples in this documentation use

- Akka Streams Kafka 0.20 (Github)
- Scala 2.11
- Akka Streams 2.5.9 (Github)
- Apache Kafka 1.0.1 (Apache Git)

Dependencies

```
libraryDependencies += "com.typesafe.akka" %% "akka-stream-kafka" % "0.20"
```

```
private def pump(): Unit = {
   if (isAvailable(out)) {
     if (buffer.hasNext) {
       val msg = buffer.next()
       push(out, createMessage(msg))
       pump()
     }
     else if (!requested) {
       requested = true
       consumer.tell(requestMessages, self.ref)
     }
   }
}
```

https://github.com/akka/reactive-kafka/blob/master/core/src/main/scala/akka/kafka/internal/SubSourceLogic.scala#L230-L242

```scala
private def pump(): Unit = {
  if (isAvailable(out)) {
    if (buffer.hasNext) {
      val msg = buffer.next()
      push(out, createMessage(msg))
      pump()
    }
    else if (!requested) {
      requested = true
      consumer.tell(requestMessages, self.ref)
    }
  }
}
```

https://github.com/akka/reactive-kafka/blob/master/core/src/main/scala/akka/kafka/internal/SubSourceLogic.scala#L230-L242

```scala
private def pump(): Unit = {
   if (isAvailable(out)) {
     if (buffer.hasNext) {
       val msg = buffer.next()
       push(out, createMessage(msg))
       pump()
     }
     else if (!requested) {
       requested = true
       consumer.tell(requestMessages, self.ref)
     }
   }
}
```

https://github.com/akka/reactive-kafka/blob/master/core/src/main/scala/akka/kafka/internal/SubSourceLogic.scala#L230-L242

Contributors Welcome!

https://github.com/akka/alpakka/blob/master/contributor-advice.md

Public factory methods

Depending on the technology you integrate with Akka Streams and Alpakka you'll create Sources, Flows and Sinks. Regardless on how they are implemented make sure that you create the relevant Sources, Sinks and Flows APIs so they are simple and easy to use.

Public factory methods

Depending on the technology you integrate with Akka Streams and Alpakka you'll create Sources, Flows and Sinks. Regardless on how they are implemented make sure that you create the relevant Sources, Sinks and Flows APIs so they are simple and easy to use.

Flows

When designing Flows, consider adding an extra field to the in- and out-messages which is passed through. A common use case we see, is committing a Kafka offset after passing data to another system.

Graph stage checklist

- Keep mutable state within the GraphStageLogic only
- Open connections in preStart
- Release resources in postStop
- Fail early on configuration errors
- Make sure the code is thread-safe; if in doubt, please ask!
- No Blocking At Any Time -- in other words, avoid blocking whenever possible and replace it with asynchronous programming (async callbacks, stage actors)

Integration testing

Can be done by running your software in Docker or referencing it externally.
Bit difficult though. Best ask for help :)

React.sphere.it

Join the conversation #sphereIT #ReactSphere