akka serialization helper

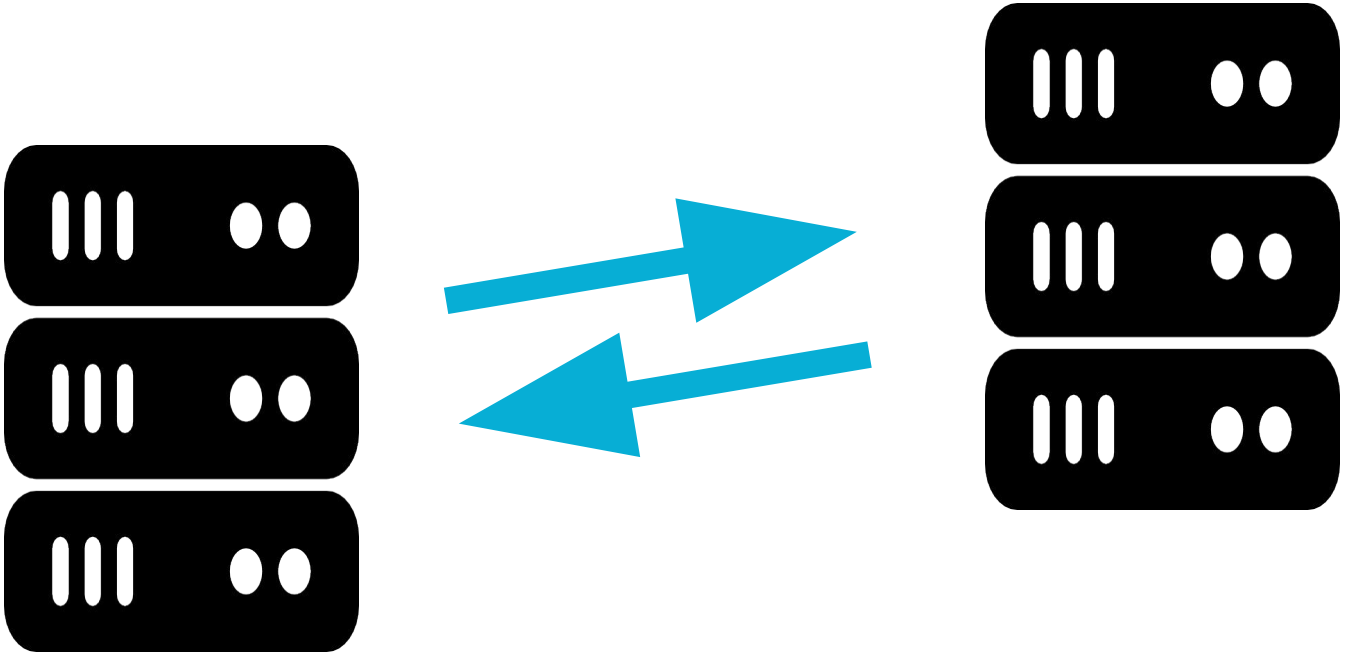
github.com/VirtusLab/akka-serialization-helper

Paweł Lipski, Marcin Złakowski

VirtusLab

Why we created ASH?

Serialization happens all the time

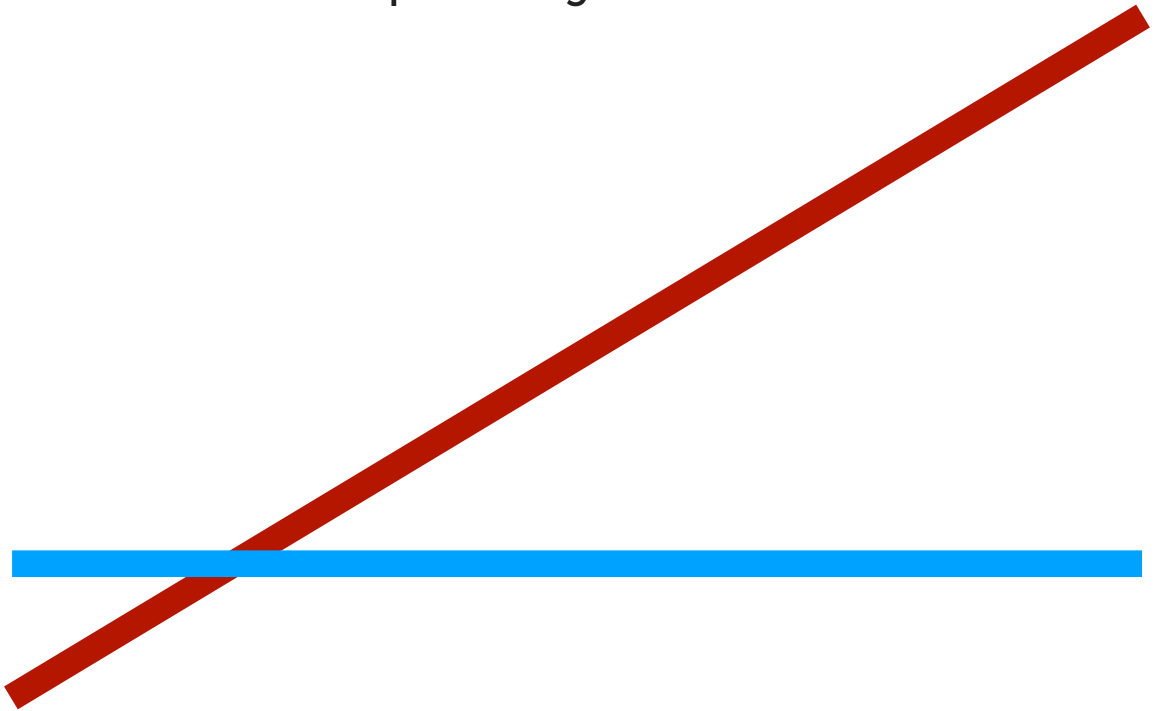


Serializables

- Messages
- Events (akka-persistence) ➔ journal
- States (akka-persistence) ➔ snapshots

Testing is never exhaustive...

...and has linear complexity in dev work



Static code analysis?

- Good for non-domain/technical issues
- Hard/impossible to test domain specific scenarios
- Constant dev work complexity with respect to codebase size...
- ...unlike testing, which has linear dev work complexity wrt. codebase size

What can go wrong in serialization?

1. Missing serialization binding

Why it exists?

```
package org
trait MySer


akka.actor {
  serializers {
    jackson-json = "akka.serialization.jackson.JacksonJsonSerializer"
  }

  serialization-bindings {
    "org.MySer" = jackson-json
  }
}
```

Easy to forget

```
trait MySer

case class MyMessage() //extends MySer
```

```
JavaSerializationException: Attempted to serialize message
using Java serialization while
`akka.actor.allow-java-serialization` was disabled.
```

Add ASH sbt plugin...

plugins.sbt:

```
addSbtPlugin("org.virtuslab.ash" % "sbt-akka-serialization-helper" % "0.4.4")

// (see the latest version in Github)
```

build.sbt:

```
lazy val app = (project in file("app"))
    .enablePlugins(AkkaSerializationHelperPlugin)
```

Introducing magic annotation

```
import org.virtuslab.ash.annotation._

@SerializabilityTrait
trait MySer
```

Detects usages in compile time

```
case class MyMessage() //extends MySer
// ...
actorRef ! MyMessage()
```

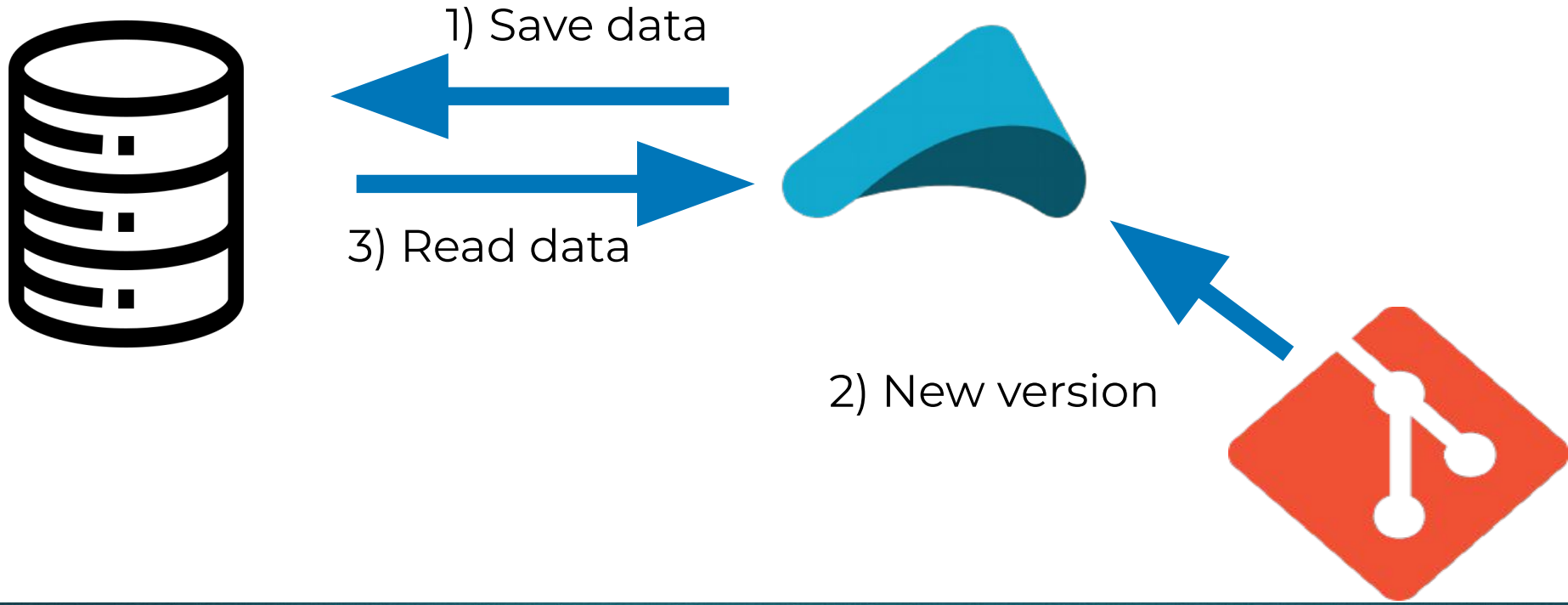
```
error: `org.MyMessage` is used as Akka message but does
not extend a trait annotated with
org.virtuslab.ash.annotation.SerializabilityTrait.

Passing an object of a class NOT extending base trait
as a message may cause Akka to reject the message during
runtime.
```

2. Inconsistent persistent data

What is persisted?

- Events (to the **journal**)
- Actor states (to the **snapshots**)

Typical tragic story



Schema definitions in Scala

```
class ActorEvent(foo: AnotherClass) //is this persisted?

class ClassYouSeeForTheFirstTime() //maybe this is?
```

If only there was an easy way to check this...

So we made one!

```
❯ sbt ashDumpPersistenceSchema
[...]
❯ ls target/
<project>-dump-persistence-schema-<version>.yaml
[...]
```

Schema of all things persisted!

```
- name: org.Event
  typeSymbol: trait
- name: org.Event.SomeInternals
  typeSymbol: class
  fields:
  - name: a
    typeName: java.lang.String
  - name: b
    typeName: scala.Int
  - name: c
    typeName: scala.Double
  parents:
  - org.Data
```

Easy to `diff`

```
❯ diff old.yaml new.yaml
8,9c8,9
<     - name: b
<       typeName: scala.Int
---
>     - name: c
>       typeName: scala.Double
```

3. Jackson Akka Serializer

Dangerous code for Jackson

```
case class Message(animal: Animal) extends MySer

sealed trait Animal
case class Lion(name: String) extends Animal
case class Tiger(name: String) extends Animal
```

```
Cannot construct instance of `Animal`
(no Creators, like default constructor, exist):
abstract types either need to be mapped to concrete
types, have custom deserializer, or contain additional
type information
```

Correct code: a lot of Jackson annotations required :/

```
case class Message(animal: Animal) extends MySer

@JsonTypeInfo(use = JsonTypeInfo.Id.NAME, property = "type")
@JsonSubTypes(
  Array(
    new JsonSubTypes.Type(value = classOf[Lion], name = "lion"),
    new JsonSubTypes.Type(value = classOf[Tiger], name = "tiger")))
sealed trait Animal
case class Lion(name: String) extends Animal
case class Tiger(name: String) extends Animal
```

Dangerous code

```
case object Tick
```

No Exception during serialization

...but deserialization ends up very bad

Instead of restoring the singleton (`Tick$.MODULE$`),

Jackson will create **another instance** of `Tick$` class!

```
actorRef ! Tick

// Inside the actor:
def receive = {
    case Tick => // this won't get matched
}// message will be unhandled
```

Corrected code

```
case class Tick()
```

Dangerous code

```
sealed trait Heartbeat

object Heartbeat {
  case object Tick extends Heartbeat
  case object Tock extends Heartbeat
}
```

Universal solution - custom serializer/deserializer

```
@JsonSerialize(using = classOf[HeartbeatJsonSerializer])
@JsonDeserialize(using = classOf[HeartbeatJsonDeserializer])
sealed trait Heartbeat

object Heartbeat {
  case object Tick extends Heartbeat
  case object Tock extends Heartbeat
}
```

This is just glorified manual serialization/deserialization

```
class HeartbeatJsonSerializer extends StdSerializer[Heartbeat](classOf[Heartbeat]) {
  import Heartbeat._

  override def serialize(value: Heartbeat, gen: JsonGenerator, provider: SerializerProvider): Unit = {
    val strValue = value match {
      case Tick => "Tick"
      case Tock  => "Tock"
    }
    gen.writeString(strValue)
  }
}

class HeartbeatJsonDeserializer extends StdDeserializer[Heartbeat](classOf[Heartbeat]) {
  import Heartbeat._

  override def deserialize(p: JsonParser, ctxt: DeserializationContext): Heartbeat = {
    p.getText match {
      case "Tick" => Tick
      case "Tock" => Tock
    }
  }
}
```

Change the serializer!

Introducing Circe

- Derives Codecs in compile time
- Full support for Scala types, including objects
- Requires mentioning serializable types (in code)

Serializer implementation with codec registration

```
class MySerializer extends CirceAkkaSerializer[MySer] {
  // ...
  override lazy val codecs =
    Seq(
      Register[CommandOne],
      Register[CommandTwo]
    )
   // whoops what if I forget to register a codec?
}
```

Another magic annotation and compiler plugin!

```
import org.virtuslab.ash.annotation._

@Serializer(classOf[MySer])
class MySerializer extends ...
```

No codec for `CommandThree` is registered in class annotated with @org.virtuslab.ash.annotation.Serializer.

This will lead to a missing codec for Akka serialization in the runtime.

Make sure to visit **sphere.it**