This is not the configuration talk you are looking for

Who I Am

Andrea Bessi

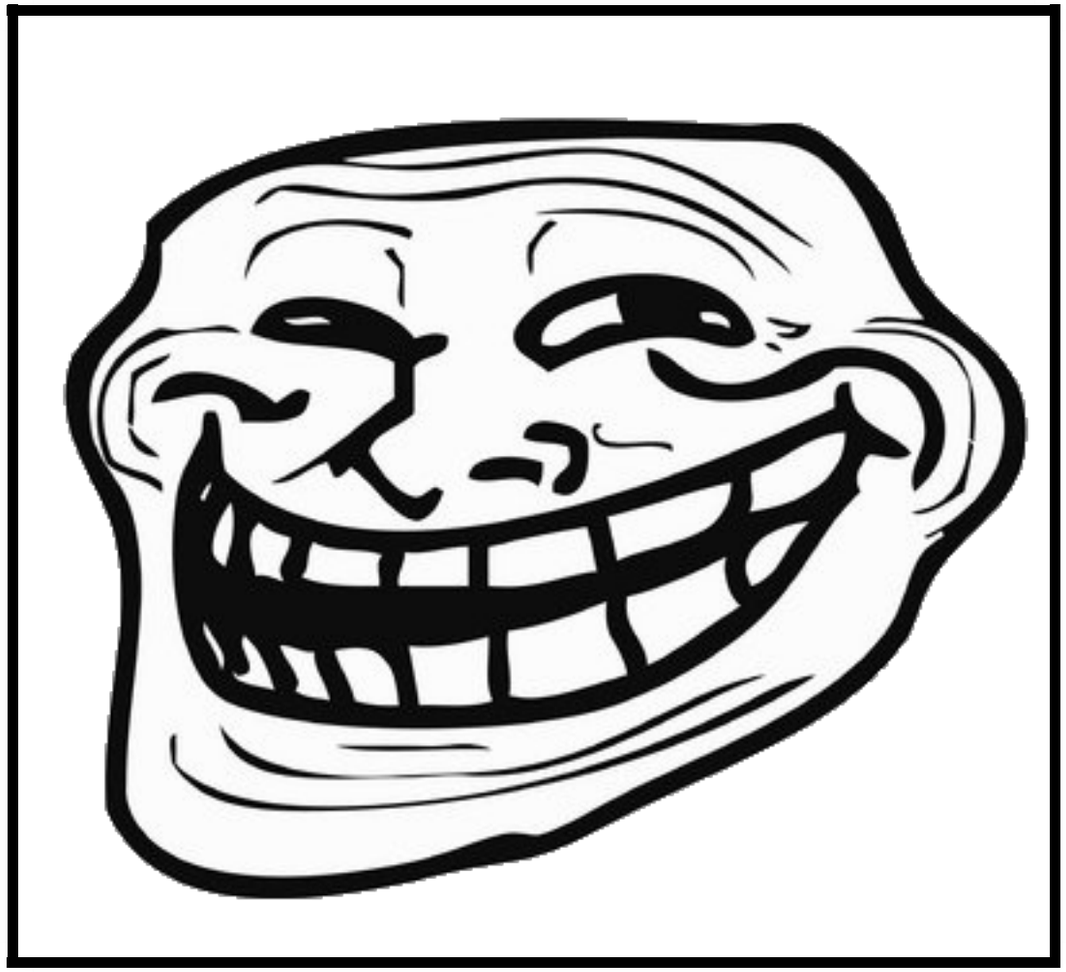
- Software Developer
- Village Idiot
- Someone Who Likes Interactive Talks

Config? What's The Big Deal?

```
import com.typesafe.config.ConfigFactory

val conf = ConfigFactory.load()
val bar = conf.getInt("foo.bar")
```

Thanks

Questions?



Problem #1

Complex Configs

```
saml {
  configs = [
    {
      partner = "partner1"
      keystore {
        path = "path/to/samlKeystore.jks"
        password = "java-keystore-password"
        private-key-password = "certificate-password"
      }
      saml-url {
        callback = "/auth/saml/callback"
        enabled-domains = [
          { domain = "domain01:9443", client-name = "samlAccount"
          { domain = "domain02", client-name = "anotherSamlAccoun
        ]
      }
```

Problem #2

Once upon a time...


play

```
import play.api.Play

val foo: Option[String] = for {
  app <- Play.maybeApplication        // Option[Application]
  config = app.configuration          // Configuration
  f <- config.getString("bar.foo")    // Option[String]
} yield f
```

A first attempt from the top of our mind

```
for {
    app <- Play.maybeApplication
    conf = app.configuration
    logoutUrl <- conf.getString("saml.logout-url")
    keyStorePath <- conf.getString("saml.configs.keystore.path")
    keyStorePassword <- conf.getString("saml.configs.keystore.pas
    privateKeyPassword <- conf.getString("saml.configs.keystore.p
    idpMetadataPath <- conf.getString("saml.configs.idp-metadata-
    samlClientName <- conf.getString("saml.configs.saml-url.enabl
    spEntityId <- conf.getString("saml.configs.sp-metadata-dir")
    callbackUrl <- conf.getString("saml.fallback-url")
    maximumAuthenticationLifetime = conf.underlying.getDuration("
} {
    BaseConfig.setDefaultLogoutUrl(logoutUrl)

    val saml2Client = new Saml2Client()
```

N.B.: we are going to use a simplified example in (almost) all the next slides

```
for {
  app <- Play.maybeApplication
  conf = app.configuration
  logoutUrl <- conf.getString("saml.logout-url")
  keyStorePath <- conf.getString("saml.configs.keystore.path")
  idpMetadataPath <- conf.getString("saml.configs.idp-metadata-ur
} builderMethod(logoutUrl, keyStorePath, idpMetadataPath)
```

'Functions should have a small number of arguments. No argument is best, followed by one, two, and three. More than three is very questionable and should be avoided with prejudice.'

Robert C. Martin

```
for {
  app <- Play.maybeApplication
  conf = app.configuration
  logoutUrl <- conf.getString("saml.logout-url")
  keyStorePath <- conf.getString("saml.configs.keystore.path")
  idpMetadataPath <- conf.getString("saml.configs.idp-metadata-ur
  container = Container(logoutUrl, keyStorePath, idpMetadataPath)
} builderMethod(container)
```

DANGER

No open flames

From a Method with N Parameters

to

A *Constructor* Method with N Parameters

How Do We Solve This?

A Builder, of Course!

```
object SamlBuilder {
  private case class SamlConfigsAccumulator(
    logoutUrl: String = "",
    keyStorePath: String = "",
    idpMetadataPath: String = ""
  )

  private def apply(accumulator: SamlConfigsAccumulator): SamlBui
    new SamlBuilder(accumulator)

  def apply(): SamlBuilder =
    apply(SamlConfigsAccumulator())

}

class SamlBuilder private(acc: SamlConfigsAccumulator) {
```

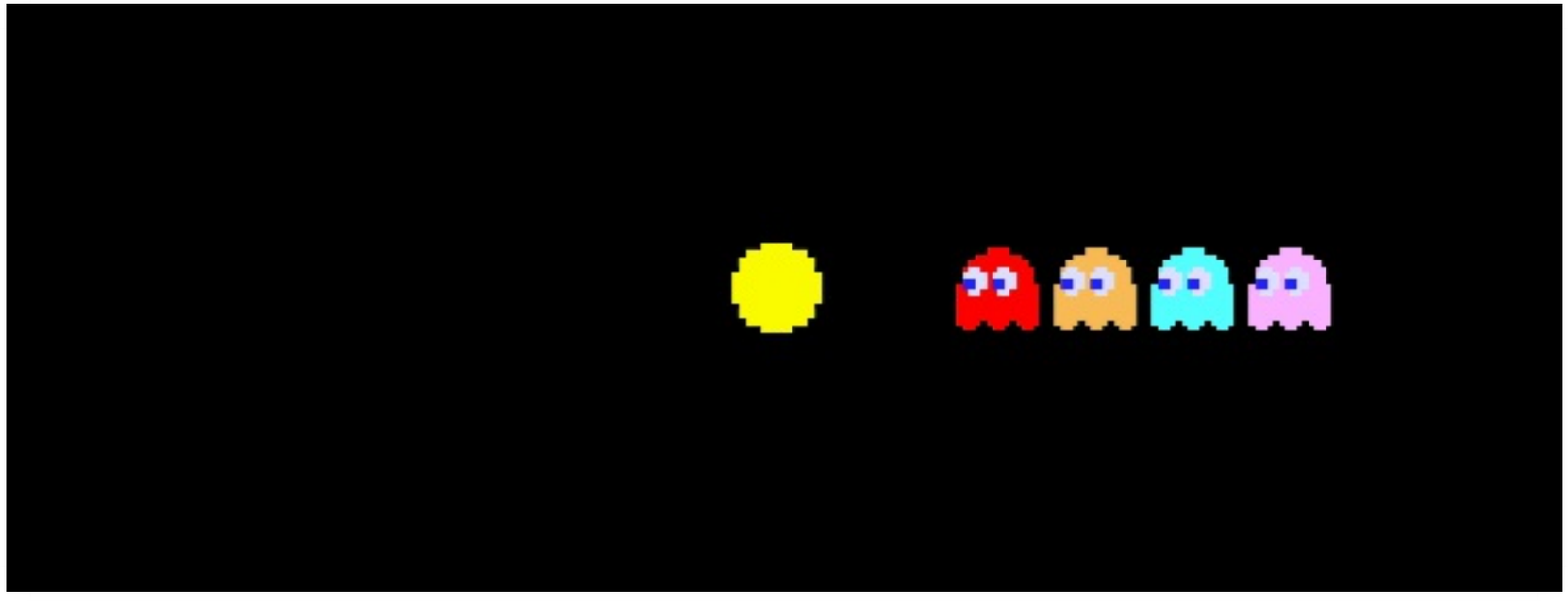
But...!

What prevents us from doing this?

```
SamlBuilder().build()
```

Can we do better?

Phantom Types To The Rescue!



"Phantom types are called this way, because they never get instantiated. Really? So what are they good for? Simply to encode type constraints, i.e. prevent some code from being compiled in certain situations."

Heiko Seeberger

```
object SamlBuilder {
  sealed trait Config
  sealed trait EmptyConfig extends Config
  sealed trait LogoutUrl extends Config
  sealed trait KeystorePath extends Config
  sealed trait IdpMetadataPath extends Config

  type FullConfig = EmptyConfig with
                    LogoutUrl with
                    KeystorePath with
                    IdpMetadataPath

  type CompleteSamlBuilder = SamlBuilder[FullConfig]

  private case class SamlConfigsAccumulator(
    logoutUrl: String = ""
```

A Little Digression

Spot The Intruder!

And Here?

```
saml {
  configs = [
    {
      partner = "partner1"
      keystore {
        path = "path/to/samlKeystore.jks"
        password = "java-keystore-password"
        private-key-password = "certificate-password"
      }
      saml-url {
        callback = "/auth/saml/callback"
        enabled-domains = [
          { domain = "domain01:9443", client-name = "samlAccount"
        ]
      }
      idp-metadata-url = "https://my.identiy-provider.com/Federat
```

```
object SamlBuilder {
  sealed trait Config
  sealed trait EmptyConfig extends Config
  sealed trait LogoutUrl extends Config
  sealed trait KeystorePath extends Config
  sealed trait IdpMetadataPath extends Config

  type AlmostFullConfig = EmptyConfig with
                          LogoutUrl with
                          KeystorePath

  type FullConfig = AlmostFullConfig with
                    IdpMetadataPath

  type AlmostCompleteSamlBuilder = SamlBuilder[AlmostFullConfig]
  type CompleteSamlBuilder = SamlBuilder[FullConfig]
```

Consider the similarities

```
val sum = (a: Int) => (b: Int) => a + b
val sum1 = sum(1)
val res = sum1(6)
println(res)
```

```
class SamlBuilder { ... }
val acsb: SamlBuilder[AlmostFullConfig] = SamlBuilder().withKeyst
val csb: SamlBuilder[FullConfig] = acsb.withIdpMetadataPath(???)
csb.build()
```


Help Needed
Partially Applied Phanto ?

Back to our code

How did it look like again?

```
for {
  app <- Play.maybeApplication
  conf = app.configuration
  logoutUrl <- conf.getString("saml.logout-url")
  keyStorePath <- conf.getString("saml.configs.keystore.path")
  idpMetadataPath <- conf.getString("saml.configs.idp-metadata-ur
} {
  SamlBuilder()
    .withLogoutUrl(logoutUrl)
    .withKeystorePath(keyStorePath)
    .withIdpMetadataPath(idpMetadataPath)
    .build()
}
```

But...!

"... the intention of the builder pattern is to find a solution to the telescoping constructor anti-pattern that occurs when the increase of object constructor parameter combination leads to an exponential list of constructors."

Wikipedia

"Traditionally, programmers have used the telescoping constructor pattern, in which you provide a constructor with only the required parameters, another with a single optional parameter, a third with two optional parameters, and so on, culminating in a constructor with all the optional parameters."

Joshua Bloch

But (at the time of this writing) no parameter is optional!

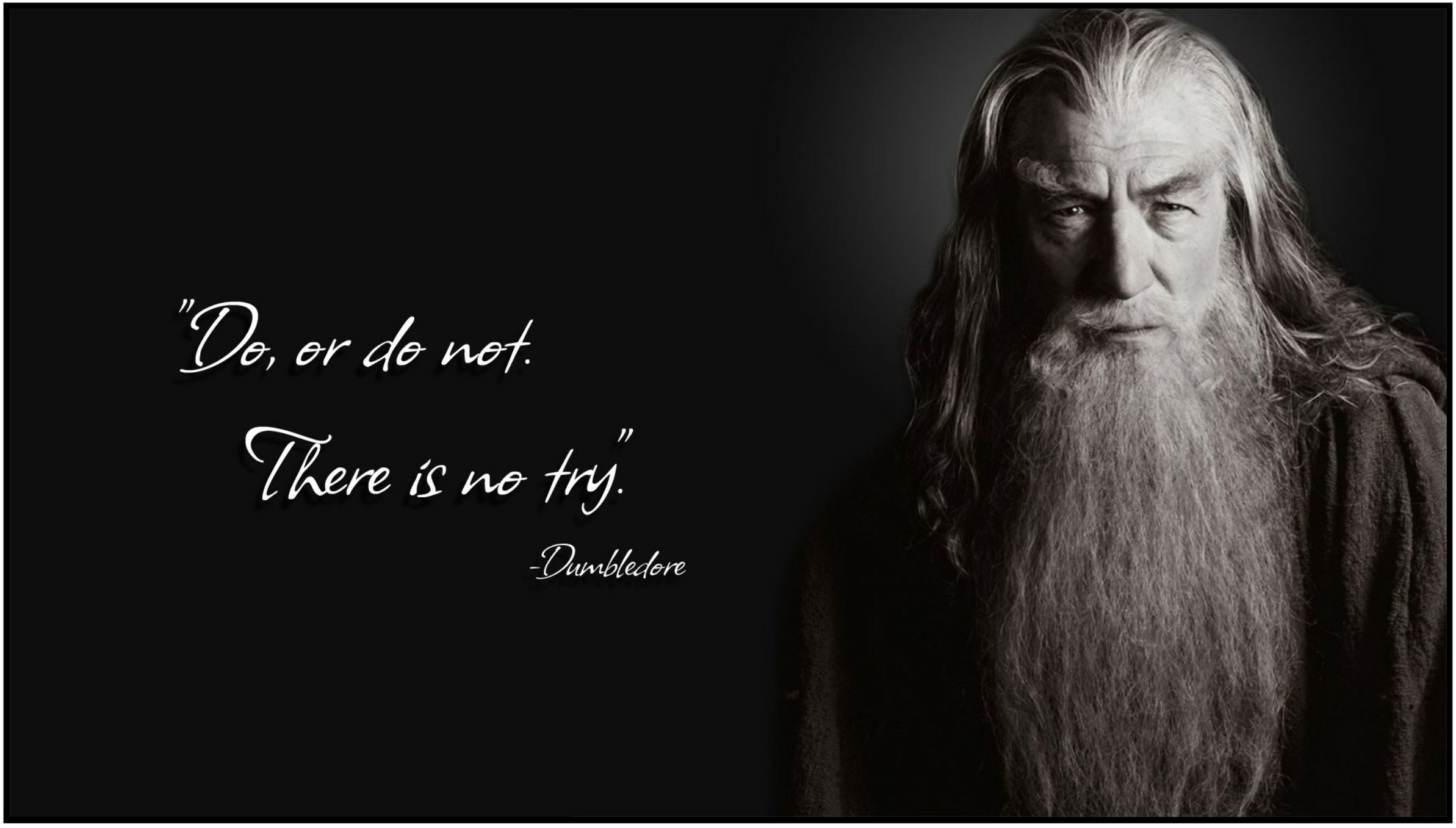
Moreover

It's really verbose, and this doesn't pay off anywhere else!

Moreover

Let's have another look at our code...

```
for {
  app <- Play.maybeApplication
  conf = app.configuration
  logoutUrl <- conf.getString("saml.logout-url")
  keyStorePath <- conf.getString("saml.configs.keystore.path")
  idpMetadataPath <- conf.getString("saml.configs.idp-metadata-ur
} {
  SamlBuilder()
    .withLogoutUrl(logoutUrl)
    .withKeystorePath(keyStorePath)
    .withIdpMetadataPath(idpMetadataPath)
    .build()
}
```


"Do, or do not.
There is no try."
-Dumbledore

```
Option(...).foreach { x =>
  SamlBuilder()
    ...
    .build()
}
```

What happens if the Option is 'None'?

- Our application (SAML SSO in this case) won't be configured correctly
- Misbehaviour will lurk *silently* in our application

We are dealing with 'Option'

But do we really need them, here?

Options

- are great for sequencing computation
  - but we aren't sequencing anything here
- are great when we have a default value
  - but there's no default value at our disposal here

```
val t = Some("SoyaWannaBurger")
t.filter(_.length > 0)
 .map(_.toUpperCase)
 .getOrElse("I'm not hungry")
```

Fail Early

or

How I Learned to Stop Worrying and Love the ~~Bomb~~ Exception

```
val optBuilder: Option[SamlBuilder[FullConfig]] = for {
  app <- Play.maybeApplication
  conf = app.configuration
  logoutUrl <- conf.getString("saml.logout-url")
  keyStorePath <- conf.getString("saml.configs.keystore.path")
  idpMetadataPath <- conf.getString("saml.configs.idp-metadata-ur
} yield SamlBuilder()
          .withLogoutUrl(logoutUrl)
          .withKeystorePath(keyStorePath)
          .withIdpMetadataPath(idpMetadataPath)
val builder = optBuilder.get
builder.build()
```

But...!

The Logging Conundrum

How can we let the deployer know which configuration isn't found?

A 'Viable' Solution

```
val optBuilder: Option[SamlBuilder[FullConfig]] = for {
  app <- Play.maybeApplication
  conf = app.configuration
  logoutUrl <- conf.getString("saml.logout-url")
  _ = Logger.debug(s"Logout URL is $logoutUrl")
  keyStorePath <- conf.getString("saml.configs.keystore.path")
  _ = Logger.debug(s"Keystore Path is $keyStorePath")
  idpMetadataPath <- conf.getString("saml.configs.idp-metadata-ur
  _ = Logger.debug(s"IDP metadata path is $idpMetadataPath")
} yield SamlBuilder()
  .withLogoutUrl(logoutUrl)
  .withKeystorePath(keyStorePath)
  .withIdpMetadataPath(idpMetadataPath)
val builder = optBuilder.get
builder.build()
```

Another 'Viable' Solution

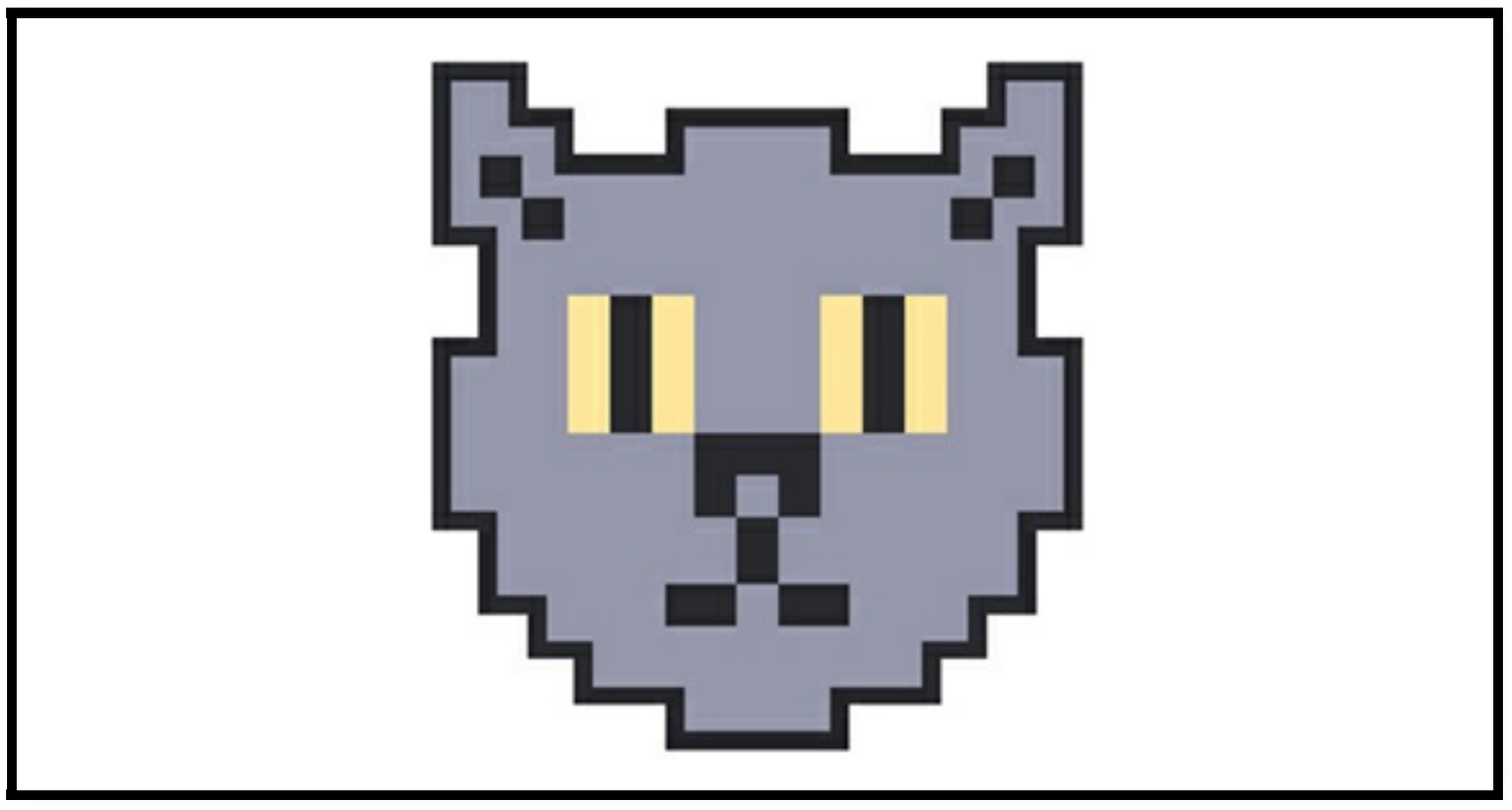
```
val conf = Play.maybeApplication.map(_.configuration).get
val logoutUrl = conf.getString("saml.logout-url").get
val keyStorePath = conf.getString("saml.configs.keystore.path").g
val idpMetadataPath = conf.getString("saml.configs.idp-metadata-u
val builder = SamlBuilder()
  .withLogoutUrl(logoutUrl)
  .withKeystorePath(keyStorePath)
  .withIdpMetadataPath(idpMetadataPath)
builder.build()
```

But...!

We are still sequencing

Wouldn't it be great if there was a way to apply computation independently?

Introduce a Little Functional Programming

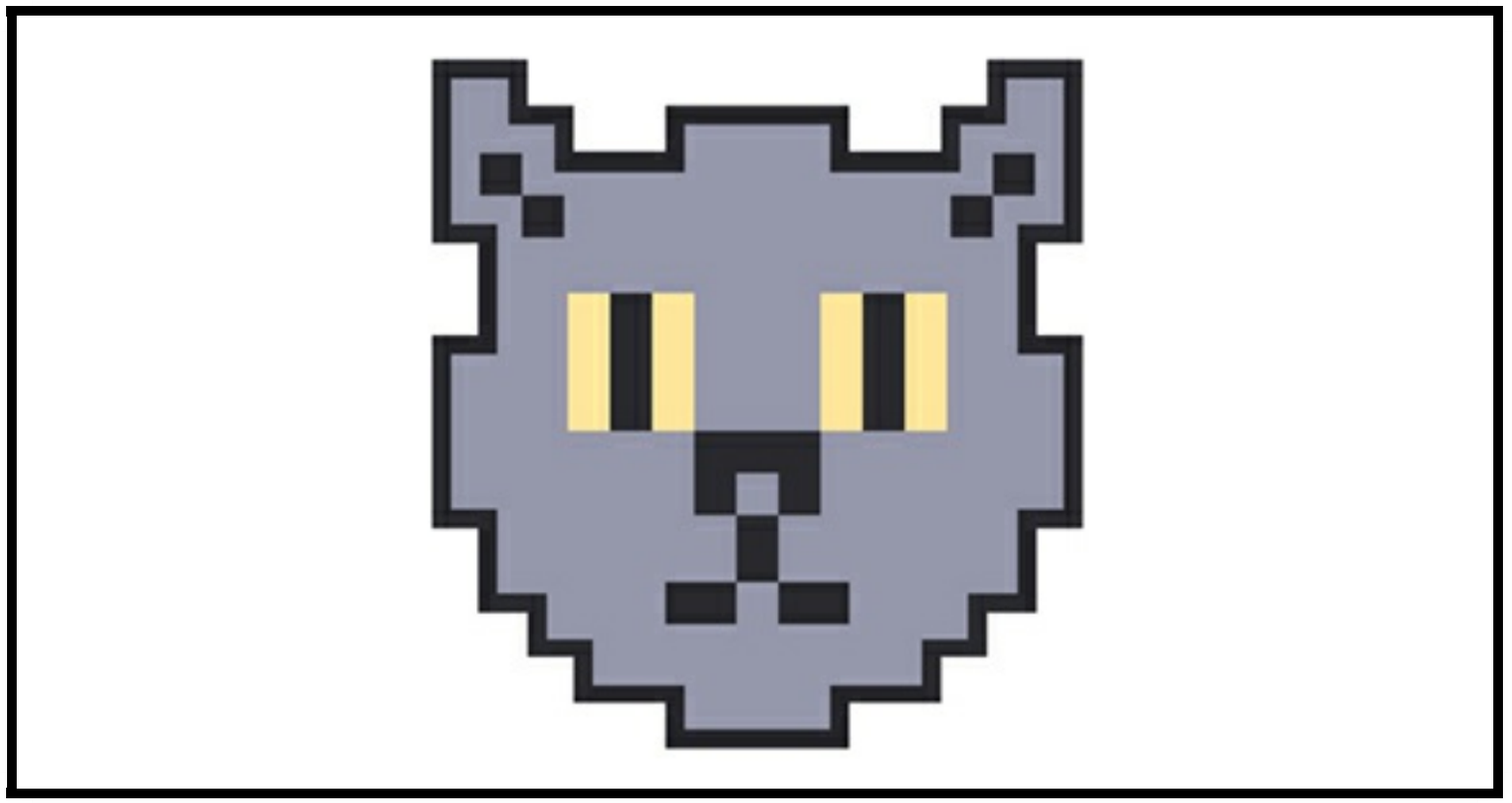


"Semigroupal is a type class that allows us to combine contexts. If we have two objects of type F[A] and F[B], a Semigroupal[F] allows us to combine them to form an F[(A, B)]"

Scala with Cats

```
val conf: Option[Configuration] = Play.maybeApplication.map(_.con
val logoutUrl: Option[String] = conf.flatMap(_.getString("saml.lo
val keyStorePath: Option[String] = conf.flatMap(_.getString("saml
val idpMetadataPath: Option[String] = conf.flatMap(_.getString("s
val optBuilder: Option[SamlBuilder[FullConfig]] =
  (logoutUrl, keyStorePath, idpMetadataPath)
    .mapN((ksPath, ksPsw, pkPsw, idpPath, spId, name, cbUrl, loUr
      .withLogoutUrl(loUrl)
      .withKeystorePath(ksPath)
      .withIdpMetadataPath(idpPath)
val builder: SamlBuilder[FullConfig] = optBuilder.get
builder.build()
```

What Do The Cats Say?



"Cats provides a data type called Validated that has an instance of Semigroupal but no instance of Monad. The implementation of product is therefore free to accumulate errors"

Scala with Cats

```
import cats.Semigroupal
import cats.data.{NonEmptyVector, Validated}

val conf: Option[Configuration] = Play.maybeApplication.map(_.con
val logoutUrl: Option[String] = conf.flatMap(_.getString("saml.lo
val keyStorePath: Option[String] = conf.flatMap(_.getString("saml
val idpMetadataPath: Option[String] = conf.flatMap(_.getString("s
```

```
val valBuilder: Validated[NonEmptyVector[String], SamlBuilder[Ful
  Validated.fromOption[NonEmptyVector[String], String](logoutUrl,
  Validated.fromOption[NonEmptyVector[String], String](keyStorePa
  Validated.fromOption[NonEmptyVector[String], String](idpMetadat
).mapN((loUrl, ksPath, idpPath) => SamlBuilder()
  .withLogoutUrl(loUrl)
  .withKeystorePath(ksPath)
  .withIdpMetadataPath(idpPath)
```

In the meantime...


play

This opens up new possibilities!

In particular...

"PureConfig is a Scala library for loading configuration files. It reads Typesafe Config configurations written in HOCON, Java .properties, or JSON to native Scala classes in a boilerplate-free way. Sealed traits, case classes, collections, optional values, and many other types are all supported out-of-the-box."

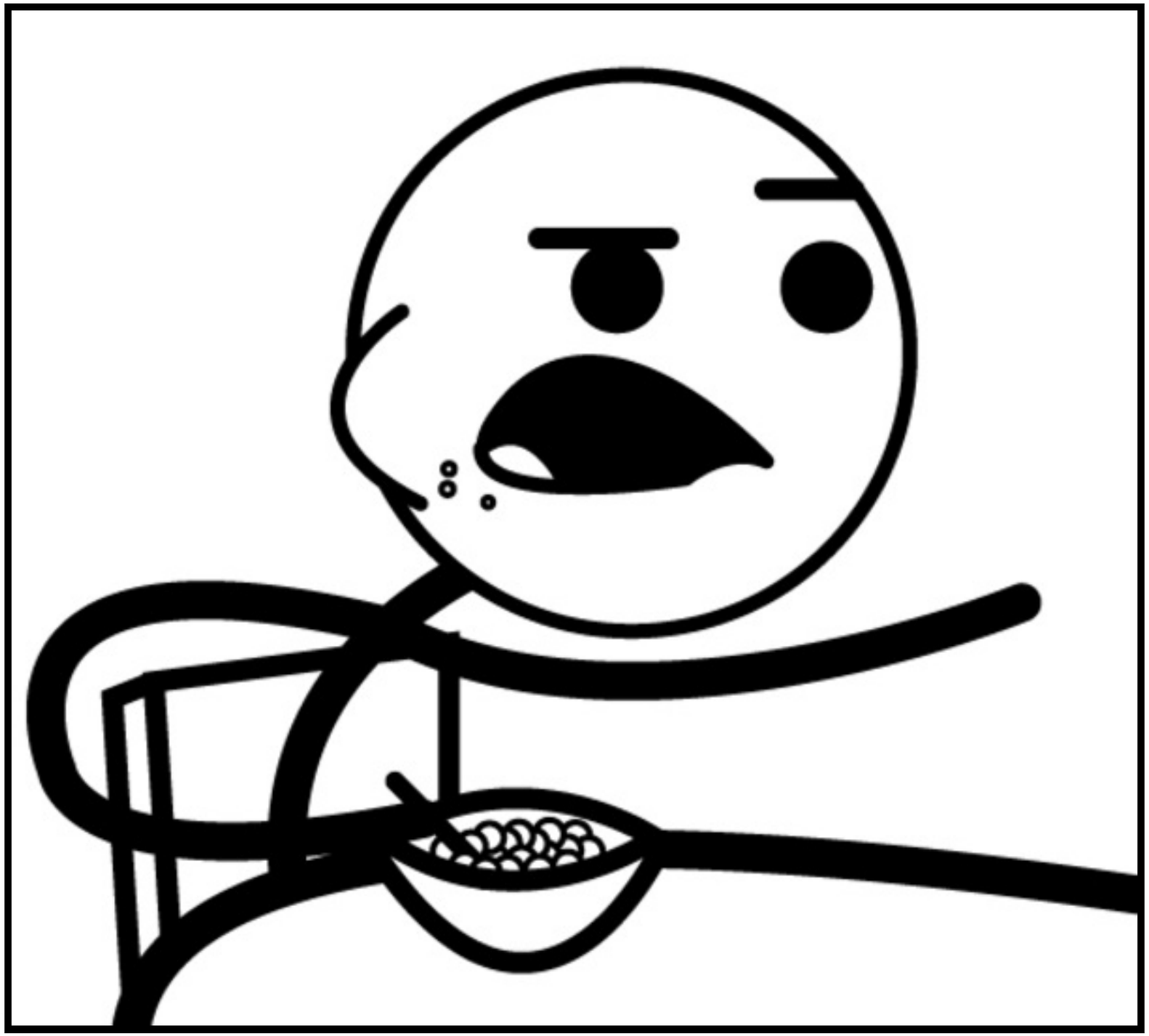
PureConfig Home Page

```
saml {
  configs = [
    {
      partner = "partner1"
      keystore {
        path = "path/to/samlKeystore.jks"
        password = "java-keystore-password"
        private-key-password = "certificate-password"
      }
      saml-url {
        callback = "/auth/saml/callback"
        enabled-domains = [
          { domain = "domain01:9443", client-name = "samlAccount"
          { domain = "domain02", client-name = "anotherSamlAccoun
        ]
      }
```

```
case class Keystore(path: String, password: String, privateKeyPas

case class EnabledDomain(domain: String, clientName: String)

case class SamlUrl(callback: String, enabledDomains: Set[EnabledD

case class SamlConfig(
    partner: String,
    keystore: Keystore,
    samlUrl: SamlUrl,
    idpMetadataUrl: String,
    spMetadataDir: String,
    maxAuthLifetime: FiniteDuration)

case class Saml(configs: Set[SamlConfig], fallbackUrl: String, lo
```

```
def provideSaml: Saml = {
  val samlConfig: Either[ConfigReaderFailures, Saml] =
    pureconfig.loadConfig[Saml]("saml")
  samlConfig.left.foreach(f => Logger.warn(
      s"""Couldn't create a full SAML configuration due to
      the following missing keys:
      ${f.toList.map(_.description).mkString("; ")}
      Proceeding with default values."""))
  samlConfig.toOption.get
}
```

"Wait a sec: weren't we avoiding case classes because we'd end up with 'a constructor method with N parameters'? "



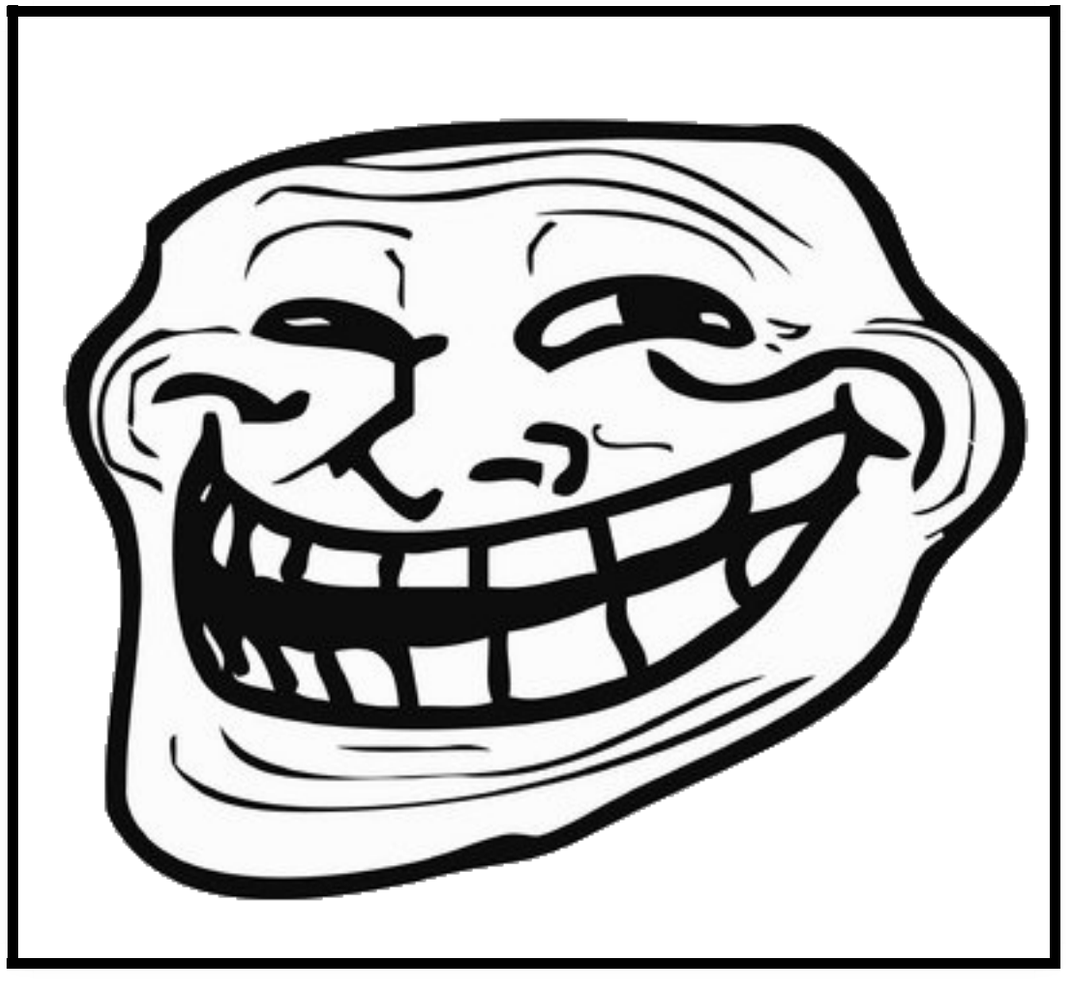
"Yes. Here we're rolling back to them because they allow us to avoid *a lot* of boilerplate "

Conclusions

Did we really talk about configurations?

- First attempt
  - 'Imperative' programming
- Builder Pattern
  - OOP, Design Patterns, Scala's type system (Phantom Types)
- Semigroupal, Validated
  - FP
- PureConfig
  - Shapeless, Macros

That's what happens with Scala



Thanks

Questions?

- Robert C. Martin's Clean Code Tip of the Week: Avoid Too Many Arguments
- Phantom Types by Heiko Seeberger
- Builder Pattern
- Telescoping Constructor Anti-Pattern
- Scala with Cats
- PureConfig
- Refined

Thanks

You can find these slides on GitHub

Go Grab Them!

Thanks