Micro Service Orchestration

A Functional Approach

Andrew Kuhnhausen - FinagleCon 2015

@kuhnhausen | github.com/trane | blog.errstr.com

Agenda

back story

primitives

Finch

abstractions

/me
work

- Engineer at  Lookout
- Started Functional Programming study groups
- Diversity Committee (let's talk)
- Moving teams to work on Functional infrastructure
- We are hiring - come ride the Curry-Howard correspondence with us

/me
academia

- Static Analysis of Dalvik Bytecode for Malware Detection
- University of Utah, advisor: Matt Might

/me
why you should (not) listen

- This is my first conference talk
- I'm not an expert in Scala nor typed FP
- Lessons from building an idiomatic Scala open-source FP library
- Library is not Java friendly

Some(Context)

ruby + rails

...a love story


true luv 4ever!!1


tests will keep us 2gether :) :) :)


elho? ping? ack? ... silence


servicesus minimus
servii? servi?

now you've just moved complexity

orchestration

- compose sessions
- allow for plugabble/modular authentication
- abstract away service discovery for clients

context level++

github.com/lookout/ngx_borderpatrol

Library Design

why inheritance isn't good enough

```
trait Mammal {
  // implement mammal
}

case object Human extends Mammal
case object Cat extends Mammal
```

inheritance for end users

define a *general* type and silently accept more *specific* types (subtypes)

shoe store

```
trait ShoeStore {
  def get(m: Mammal): Kicks
}
```

others want our shoes

```
case object Bird extends Animal
case object Lizard extends Reptile
case object Velociraptor extends Bird with Lizard
```

:(

interfaces

define a *specific* description and silently allow more *general* types

icanhazshoes?

- Anything with `Feet` can (probably) wear shoes
- Support `Reptile`, `Bird`, `Mamamal`, but not `ImperialMetrics`

adapter pattern

adapter pattern

interface membership is determined at *definition*

monkey patching

type classes

Abstracting

web sessions

```
trait Session[A] {
  val id: SessionId
  val data: A
}

trait SessionId {
  val expiry: Time
  val entropy: Seq[Byte]
  val secret: Secret
  val signature: Seq[Byte]
}
```

interface

```
type K
type Get[B]: K => Store => Option[B],
type Put[B]: K => B => Store => Unit
```

A !=? B

laws

- `A => B`
- `B => Option[A]`

define the general law

```
trait Encoder[A, B] {
  def encode(a: A): B
  def decode(b: B): Option[A]
}
```

define a less general law

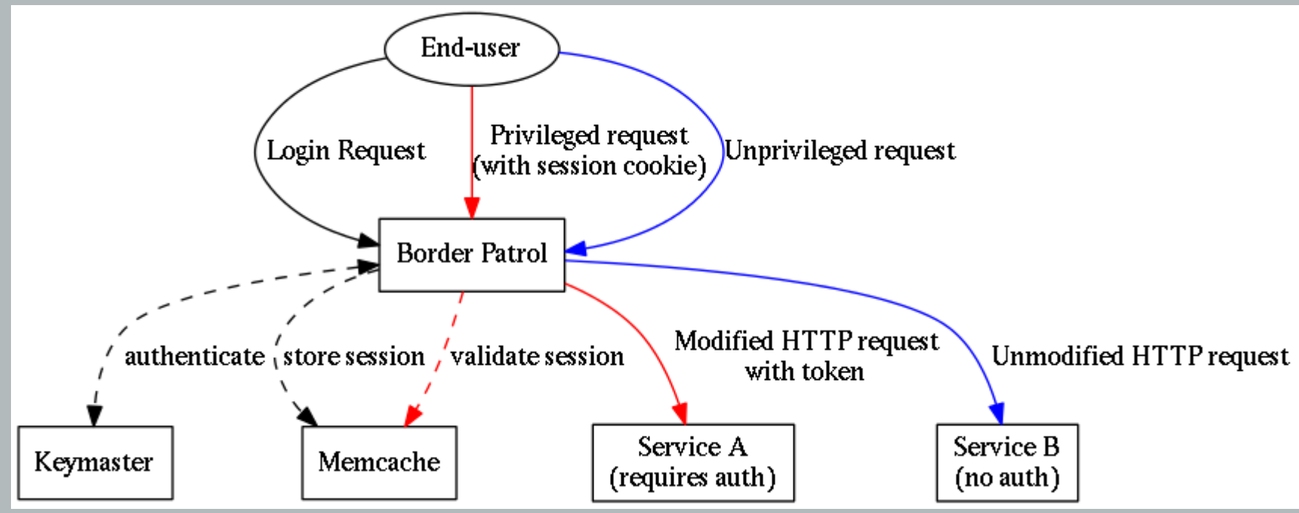
Buf

```
type EncodeSession[A] = Encoder[A, Buf]

// or

trait EncodeSession[A] {
  def encode(data: A): Buf
  def decode(buf: Buf): Option[A]
}
```

defaults


End-user
Login Request
Privileged request
(with session cookie)
Unprivileged request
Border Patrol
authenticate
store session
validate session
Modified HTTP request
with token
Unmodified HTTP request
Keymaster
Memcache
Service A
(requires auth)
Service B
(no auth)

EncodeSessionRequest

```
implicit object EncodedSessionRequest
    extends EncodeSession[httpx.Request] {

  def encode(data: httpx.Request): Buf =
    Buf.ByteArray.Owned(data.encodeBytes())

  def decode(buf: Buf): Option[httpx.Request] =
    Try { Request.decodeBytes(Buf.ByteArray.Owned.extract(buf)) }.toOption
}
```

use the type class

```
trait SessionStore {
  def put[A : EncodeSession](session: Session[A]): Future[Unit]
  def get[A : EncodeSession](id: SessionId): Future[Option[Session[A]]]
}
```

a simple memcached implementation

```
case class MemcachedSessionStore(store: memcachedx.BaseClient[Buf])
    extends SessionStore {
  def put[A : EncodeSession](session: Session[A]): Future[Unit] =
    store.set(session.id, 0, 60, implicitly[EncodeSession[A]](session.data))

  def get[A : EncodeSession](id: SessionId): Future[Option[Session[A]]]
    store.get(id).map {opt =>
      opt.flatMap {buf =>
        Session(id, implicitly[EncodeSession[A]].unapply(buf))
      }
    }
}
```

what just happened?

more advanced fun-time

Session[A] + Functor[F[_]] + Free Monad

:) yesplz! (:

auth as a type class

Primitives

Finagle

Service

```
type Service[Req, Rep] = Req => Future[Rep]
```

Filter

```
type Filter[ReqIn, RepOut, ReqOut, RepIn] =
  (Req, Service[ReqOut, RepIn]) => Future[RepOut]
```

Finch

Router / RequestReader / ResponseBuilder

```
type Router[A] = Request => Option[A]

type RequestReader[A] = Request => Future[A]

type ResponseBuilder[A] = (A, EncodeResponse[A]) => Response
```

Composability

Router Combinator

```
type Router[A] = Request => Option[A]
```

Combinator DSL

```
// GET /users/:id/tickets/:id
get("users" / int("userId") / "tickets" / int("ticketId"))
```

Using these primitives

Router for Path based Service

```
http://example.com/:service/...
```

Router for Subdomain based default Service

```
http://service.example.com/
```

Path based Service

lookout.com/:service/ => :service

```
implicit val pathMap = Map(("ent" -> "enterprise"), ("my" -> "consumer"))

def validatePath(path: String)(implicit services: Map[String, String]): Option[String] =
  services.get(path)

object servicePath extends Extractor("service", validate(_))
```

Default subdomain based

```
case class DefaultServiceDomain(g: String => Option[String])
    extends Extractor[String]("defaultService", identity) {

  import Router._

  override def apply(input: Input): Option[(Input, () => Future[String])] =
    for {
      h <- input.request.host
      s <- g(h)
    } yield (input.drop(1), () => Future.value(s))
}

val domainMap = Map(("www" -> "consumer"), ("enterprise" -> "enterprise"))
val subdomain = DefaultServiceDomain(domainMap.get(_))
```

Compose them

```
val serviceRouter = servicePath | subdomain

  if (:service in path) then:
    mapping(:service)
  if (:service in subdomain) then:
    default(:service)
  else
    halt with 400
  end
```

w00t! w00t!

What this looks like

```
val endpoints = (
  (Get / serviceRouter /> AuthenticatedEndpoint) :+:
  (Post / "login" /> LoginService))
)

val server = Httpx.serve("localhost:8080", endpoints.toService)
```

Things bp does

- Sessions
- Authentication (Basic, internal)
- Encryption for data at rest

Things bp is about to do

- CSRF via double submit cookie
- Periodic secret key generation

Things you can help with

- github.com/lookout/borderpatrol/issues
- Build your own abstractions on Finagle/Finch

Thank you

@kuhnhausen | github.com/trane | blog.errstr.com