Modernizing Legacy: from Monolithic PHP to Reactive SOA

ScalaDays 2015 San Francisco

Mike White
Senior Engineer
@hootsuitemike
“There are two kinds of startups: the ones that achieve some modest traction on top of a pile of code of which they are vaguely ashamed ... and the ones that go out of business. That’s it. No third kind.”

- Dan Milstein
This talk

- What is legacy software?
- Case study: A service extraction from the monolith
- Case study: A drop-in replacement for a URL-shortener application
• Legacy code is fossilized technical debt.

• Iterate and Learn .. *in production*?!

• Modernization patterns with Scala: service extraction and replacement
A little about me

- Senior Engineer at Hootsuite
- Too many years of Java experience
- Enterprise software modernization
- Yes, COBOL and Fortran ..to JavaEE
- Loving Scala and not looking back!
A little about Hootsuite

- The world’s most widely used social relationship platform
- Founded in 2008
- Over 11 million users worldwide
- Over 2000 Enterprise customers
Web Twitter Client

Twitter Client For Web & Mobile

Multiple Social Networks

Social & Analytics Integrations

Globalization & Collaboration

Social Relationship Platform

2008

2009

2010

2011

2012

2013

2014

Hootsuite 1.0

Hootsuite 2.0

Hootsuite 5.0

iOS App

Android App

App Directory

Facebook Profiles, Pages, and Groups

LinkedIn Company Pages

Twitter Social Ads

Facebook Insights

Conversations

Security & Performance

iOS7 App

iOS8 App and Share Extension

UberVU Acquisition

Brightkit Acquisition

Zeetl Acquisition

Zeetl Acquisition
ScalaFact
2:10pm via BestScala
FVY, we only have a few "Fast Track trainings" left for #Scala & #Play at @scaladays. 2/3 of trainings are sold out! buff.ly/1DvuB9m

typesafe
1:56pm via Buffer
FVY, we only have a few "Fast Track trainings" left for #Scala & #Play at @scaladays. 2/3 of trainings are sold out! buff.ly/1DvuB9m
tweeted by scaladays and 1 others

honzam399
9 Feb 19, 12:15pm via Tweetbot for Mac
Not yet decided to go to #ScalaDays in SF? Let me make things easier with a $100 discount code Mach100.
tweeted by scaladays and 4 others

gainio
Feb 21, 2:22pm via Twitter Web Client
all set to be part of #scaladays SF. Looking forward!
tweeted by scaladays

wendevedolver
9 Feb 23, 10:56am via Twitter Web Client
*wish* I could go to #scaladays in SF! Fantastic program and talks: skillsmatter.com/conferences/70...
#scala #Play #Functional

uhAndrew
1:39pm via Hootsuite
= recursion

THE ESSENTIAL AKKC CHECKLIST
The Problem with Legacy
Legacy Code is Fossilized Technical Debt

- Maintenance
- Hosting
- Deployment
- Scaling
- Fragility
Legacy Code is Fossilized Technical Debt

- Huge Risks
- Fear, Uncertainty, Doubt
- Innovation Paralysis
- Technology itself becomes the bottleneck...
Hootsuite’s Legacy Problem

- Our PHP monolith
- Dynamically typed
- Large code base
- No clear modularity
- Low test coverage
Community.
Motivations and Requirements

- Future product integration needs
  - PaaS, APIs
- Acquisitions
- Cost

“Our hosting costs increased faster than we expected, .... Dozens of AWS instances were required to handle peak loads when we knew that a re-architected solution could significantly reduce the number of servers required to deliver functionality.”

Geordie Henderson
VP, Engineering

- Scalable!
- Composable microservices – SOA
- Integration with monolith
- Fault tolerance, partial outages
Oh, and One More Thing

ZERO

DOWNTIME
Our Engineering Strategy

• Iterative experiments leading to commitment
• Build, Measure, Learn
• Always in Production
Our decisions

- Scala/Akka
- Responsive, Resilient, Elastic, Message Driven
- Roll-our-own micro service framework
- Play Framework
Scaling the team

● Hired Java developers
● Scala training with BoldRadius
● Need to scale development practice itself
● Conway’s law
Conway’s Law

From Martin Fowler and James Lewis’ excellent “Microservices” article

Siloed functional teams...

... lead to silod application architectures.
Because Conway’s Law
Conway’s Law

From Martin Fowler and James Lewis’ excellent “Microservices” article

Cross-functional teams...

... organised around capabilities
Because Conway’s Law
Align the Team

- **Hootsuite Product Areas:**
  
  | **Engagement**       | Real-time listening, actions, and social workspace |
  | **Publisher**        | Content management, scheduling, and workflow      |
  | **Analytics**        | Social media insights and uberVU via Hootsuite product |
  | **Campaigns**        | Social campaigns to engage followers and attract leads |
  | **Mobile**           | iOS, Android, and mobile web Hootsuite applications |
  | **Ecosystem**        | App Directory, Integrations, and SDKs/ APIs       |
  | **Advertising**      | Social network promoted products                  |
member-service

The first service extraction from the monolith
MySQL
What we Started With

- Storage, lookup, credential validation of Hootsuite users
- Embedded SQL everywhere
- Database schema change with a partial table extraction
- 4000 requests per second
- ZERO DOWNTIME
Member Service: Why?

- Follow the dependencies!
- Start with the hardest thing
- Business drivers
How We Did It

• Create the Scala service
• Refactor PHP to use a central internal API for calls
• Create PHP SDK for calling the new service
• Darklaunch it gradually
Darklaunch is Awesome

- Super cool name for feature flagging
- Instantaneous across an entire cluster
- Hootsuite uses Consul as a distributed key/value store
- Comes in many flavours:
  - boolean
  - % random
  - % static
  - Hootsuite Organization
  - User ID list
  - Backend host name
  - Just you!
if (In_Feature::isEnabled('PUBLISH THROUGH SCUM')) {
    $clientContainer = \In\Container\Container::getInstance();
    $socialCommClient = $clientContainer->get('social_communication.client');
    $messageParams = $socialCommClient->sendPosts(...);
    $messageObj->post(...);
} else {
    $messageObj->post();
}
Our custom microservices framework

- Scala/Akka
- Custom RPC protocol
- Statsd/Sensu/Logstash/ElasticSearch
- Scalable up
- Scalable down
Our custom framework

- Puller
- Foreman
- Pusher
- Worker
- Worker
- Worker
- Worker
- Worker...
Our custom framework
endpoint("/photos", Method.Post, router,"Create a new photo <snip>") {
  case EndpointData(req, _, resp) =>
    timed(...) {
      val actionTry =
        for {
          addPhotoRequest <- AddPhotoRequest.parseAndValidate(req.body)
          newPhoto = addPhotoRequest.toPhoto
          id <- photoDao.create(newPhoto)
          photoOption <- photoDao.findById(id)
          photo = photoOption.getOrElse(throw NotFound(IdNotFound))
        }
        yield (photo, HootsuiteHash.fromId(id).get)
      val response = actionTry match {
        case Success((photo, hhash)) =>
          log.debug(...)
          resp.status200.withBody(PhotoResponse.fromPhoto(photo, hhash).toJson)
        case Failure(error) =>
          log.warning(...)
          handleError(req, resp, error)
      }
      Future.successful(reportResponse(..., response))
    }
}
```scala
endpoint("/photos", Method.Post, router,"Create a new photo <snip>") {
  case EndpointData(req, _, resp) =>
    timed(...) {
      val actionTry = for {
        addPhotoRequest <- AddPhotoRequest.parseAndValidate(req.body)
        newPhoto = addPhotoRequest.toPhoto
        id <- photoDao.create(newPhoto)
        photoOption <- photoDao.findById(id)
      } yield (photo = photoOption.getOrElse(throw NotFound(IdNotFound)))
    } yield (photo, HootsuiteHash.fromId(id).get)
    Future.successful(reportResponse(..., response))
```
endpoint("/photos", Method.Post, router,"Create a new photo <snip>") {
  case EndpointData(req, _, resp) =>
    timed(...) {
      val actionTry = for {
        addPhotoRequest <- AddPhotoRequest.parseAndValidate(req.body)
        newPhoto = addPhotoRequest.toPhoto
        id <- photoDao.create(newPhoto)
        photoOption <- photoDao.findById(id)
        photo = photoOption.getOrElse(throw NotFound(NotFound))
      } yield (photo, HootsuiteHash.fromId(id).get)

      val response = actionTry match {
        case Success((photo, hhash)) =>
          log.debug(...)
          resp.status200.withBody(PhotoResponse.fromPhoto(photo, hhash).toJson)
        case Failure(error) =>
          log.warning(...)
          handleError(req, resp, error)
      }

      Future.successful(reportResponse(..., response))
    }
}
Iterative Gradual Rollout

- Stage 0: “Old code and new service development”
- Stage 1: “Replication”
- Stage 2: “Parallel reads and updates”
- Stage 3: “Creates”
- Stage 4: “On with fallback”
- Stage 5: “Burn the safety net”
Why did this work?

- Darklaunching
- Iteration, always in production
- Measure everything
- Scala/Akka
  - Easy to develop
  - Easy to test
  - Fast!
Exciting failures!

- Cold startup vs. -XX:+TieredCompilation
- Don’t call the failing service when handling its failure
- Circuit Breaker
- Connection Pool
The Big Three

- Legacy code is fossilized technical debt
- Iterate and Learn .. in production
- Modernization patterns with Scala
The Modernization of ow.ly
What is ow.ly?

- Hootsuite’s URL shortening and image hosting service
- Public website and API
- Private API used by the Hootsuite dashboard
- URL Click Analytics
- 10 images uploaded per second
- 1000 URL redirects per second
How We Found It

Truly fossilized legacy code:

- Snowflake deployment
- No tests at all
- Old version of PHP
- Security holes
- Business requesting changes
- **Total paralysis**
What did we do?

- Frontend: Play Framework
- Backend data service: microservice using our custom framework
- Cross-team effort
- Zero planned downtime and very few bugs
- No huge scoping failures!
Mike’s Modernization Method

- Clear scoping
- Full team development, then smaller team to monitor towards 100% production
- Development best practices
  - Unit test coverage
  - Integration tests
  - Multiple environments
  - Proper monitoring, logging, alerting
  - Go to production with the M-est VP!
  - Darklaunch is awesome
Or, how we didn’t underestimate or overestimate the work by 6 months!

- Start with the user story
  - Private API
  - Public API
  - Screens
  - AJAX endpoints
  - 301 redirects

- Deprecated API versions
- Deprecated screens
- Deprecated cross-cutting features
- Deprecated technology
MMM: Scope analysis

- Only took a couple weeks
- Product owner and architect both signed off on appropriate deprecations
- No specifications
● Again, start with the story
● Look at the PHP for business logic
● Keep in mind the cross-cutting scope alterations
● Err on the side of “leave it in!” and “leave it broken!”
● Clear modularization
Let’s Look at Some Scala!

- Mix-in component pattern in Play
/** Mixin component for [[UrlGatekeeper]] */

trait UrlGatekeeperComponent {
  val urlGatekeeper: UrlGatekeeper
}

trait DefaultUrlGatekeeperComponent extends UrlGatekeeperComponent {
  override val urlGatekeeper = DefaultUrlGatekeeperComponent.urlGatekeeper
}

object DefaultUrlGatekeeperComponent {
  val urlGatekeeper = new DefaultConfigComponent
    with UrlGatekeeper
    with DefaultStatsdComponent
    with DefaultOwlyDataServiceComponent
    with SurblDnsLookup
    with ConfigWhitelist
}
import org.mockito.Mockito._
import org.scalatest.mock.MockitoSugar
import utils.urlgating.UrlGatekeeper

trait MockUrlGatekeeperComponent extends UrlGatekeeperComponent with MockitoSugar with MockReset {

  override val urlGatekeeper = mock[UrlGatekeeper](...)

  override def resetMocks() {
    reset(urlGatekeeper)
    super.resetMocks()
  }
}
val controller = new TestConfigComponent
  with PrivateApiShortUrlController
  with MockOwlyDataServiceComponent
  with MockStatsdComponent
  with MockBlacklistComponent
  with MockUrlGatekeeperComponent
  with MockTwitter11ApiComponent
  with MockImageResizerComponent
  with MockS3Component

it should "200 and successfully load a single url" in {

  val req = FakeRequest("GET", "/api/hs/expandUrl?...")
  val result = call(controller.expandUrl(), req)

  status(result) should equal(OK)
  contentAs.Json(result).as[ ] should equal( )

}
Play Framework UX

- Stole some UX engineers from another team
- Zero training on Play
- Grunt framework for asset pipeline and CDN publishing

“The Play framework allowed us to quickly become productive in a new technology stack using our existing tools and pipeline. Any UX engineer can do now a tour of duty on the ow.ly team.”

Steve Mynett
Senior Front End Engineer, Hootsuite
Make your links manageable.

Paste your URL, shrink it, and share. Shrink URL

Measure your success. Pair Ow.ly with Hootsuite and get deep analytics with each link you share. Try Hootsuite for FREE
Anecdote - Gearman integration for ow.ly analytics

- Fortunate to have a rich Java library that we could integrate
- Wrapped in an Actor for easy implementation of different integration semantics
  - Fire-and-forget
  - Reliable RPC
  - Bulkheading
  - Load-shedding
  - Buffering
def shortenUrl = Action.async(parse.tolerantText)(timedFuture(...) { request =>
  statsd.inc(...)

  val requestParams = getArrayNotationRequestParams(request)
  authenticatedWithMemberApiKey(requestParams, request.uri) { apiMember =>
    val longUrlStringParam = requestParams.getString("longUrl")
    val isStaticUrlParam = requestParams.getString("type")

    val response = for {
      longUrlString <- validateLongUrlParam(longUrlStringParam)
      maybeUrl <- validUrl(longUrlString)
      url <- isUrlBlocked(urlGatekeeper, maybeUrl)
      maybeOwlyMemberId = apiMember.memberId.map(_.toLong).orElse(Some(0L))
      addShortUrlReq = makeShortUrlAddRequest(apiMember, url, isStaticUrl, maybeOwlyMemberId)
      shortUrl <- statsd.timedFuture(...)(owlyDataService.addShortUrl(...))
    } yield {
      Logger.info(...)
      statsd.inc(...)
      Ok(result)
    }
  }
}
def shortenUrl = Action.async(parse.tolerantText)(timedFuture(...) { request =>
statsd.inc(...)
val requestParams = getArrayNotationRequestParams(request)
authenticatedWithMemberApiKey(requestParams, request.uri) { apiMember =>
val longUrlStringParam = requestParams.getString("longUrl")
val isStaticUrlParam = requestParams.getString("type")
val response = for {
  longUrlString <- validateLongUrlParam(longUrlStringParam)
  maybeUrl <- validUrl(longUrlString)
  url <- isUrlBlocked(urlGatekeeper, maybeUrl)
  maybeOwlyMemberId = apiMember.memberId.map(_.toLong).orElse(Some(0L))
  addShortUrlReq = makeShortUrlAddRequest(apiMember, url, isStaticUrl, maybeOwlyMemberId)
  shortUrl <- statsd.timedFuture(...)(owlyDataService.addShortUrl(...))
}
yield {
  Logger.info(...)
  statsd.inc(...)
  Ok(result)
}
})
def shortenUrl = Action.async(parse.tolerantText)(timedFuture(...) { request =>
  statsd.inc(...)
  val requestParams = getArrayNotationRequestParams(request)
  authenticatedWithMemberApiKey(requestParams, request.uri) { apiMember =>
    val longUrlStringParam = requestParams.getString("longUrl")
    val isStaticUrlParam = requestParams.getString("type")

    val response = for {

      longUrlString <- validateLongUrlParam(longUrlStringParam)
      maybeUrl <- validUrl(longUrlString)
      url <- isUrlBlocked(urlGatekeeper, maybeUrl)
      maybeOwlyMemberId = apiMember.memberId.map(_.toLong).orElse(Some(0L))
      addShortUrlReq = makeShortUrlAddRequest(apiMember, url, isStaticUrl, maybeOwlyMemberId)
      shortUrl <- statsd.timedFuture(...)(owlyDataService.addShortUrl(...))
      result = Json.toJson(PublicUrlResponse.fromShortUrlAndRoot(...))
    } yield {
      Logger.info(...)
      statsd.inc(...)
      Ok(result)
    }
  }
})
def shortenUrl = Action.async(parse.tolerantText)(timedFuture(...) { request =>
  statsd.inc(...)
  val requestParams = getArrayNotationRequestParams(request)
  authenticatedWithMemberApiKey(requestParams, request.uri) { apiMember =>
    val longUrlStringParam = requestParams.getString("longUrl")
    val isStaticUrlParam = requestParams.getString("type")

    val response = for {
      longUrlString <- validateLongUrlParam(longUrlStringParam)
      maybeUrl <- validUrl(longUrlString)
      url <- isUrlBlocked(urlGatekeeper, maybeUrl)
      maybeOwlyMemberId = apiMember.memberId.map(_.toLong).orElse(Some(0L))
      addShortUrlReq = makeShortUrlAddRequest(apiMember, url, isStaticUrl, maybeOwlyMemberId)
      shortUrl <- statsd.timedFuture(...)(owlyDataService.addShortUrl(...))
    } yield {
      Logger.info(...)
      statsd.inc(...)
      Ok(result)
    }
  }
})
/**
 * Generic handler to define the recover on the Result Future.
 */

private def doRecover(request: Request[String], response: Future[Result], 
  response.recover {
    case ValidationException(errMsg, logMsg, _) =>
      statsd.inc(stat(400))
      Logger.warn(s"$tag: ValidationException: $errMsg $logMsg")
      BadRequest(makeApiErrorJson(request, errMsg))
    case NotFoundException(errMsg, _, _) =>
      statsd.inc(stat(404))
      NotFound(makeApiErrorJson(request, errMsg))
    case InvalidDataException(codes, _) if hasOneOfErrorCode(codes, OwlyDataServiceErrorCodes.InvalidHash) =>
      // Legacy compatibility: 404 if owlydata reports invalid hash
      statsd.inc(stat(404))
      NotFound(makeApiErrorJson(request, ApiErrorMessages.InvalidShortUrl))
    case InvalidDataException(codes, _) =>
      Logger.error(s"Uncaught InvalidDataException from owlydata: ${codes.mkString(" ", ")}" )
      statsd.inc(stat(500))
      InternalServerError(makeApiErrorJson(request, s"Error getting stats"))
    case e: Exception =>
      Logger.error(s"$tag got ${e.getMessage}", e)
      statsd.inc(stat(500))
      InternalServerError(makeApiErrorJson(request, s"Error getting stats"))
  }
}
How Scala/Akka/Play Made This Successful

- Darklaunching
- Iteration (always in production!)
- Measure everything
- Play framework!
Exciting Failures!

- PHP serialization!
- Single threaded DDoS bots!
Legacy code is fossilized technical debt

Iterate and Learn .. in production

Modernization patterns with Scala
Thank You!

Mike White
@hootsuitemike

More Info:
code.hootsuite.com