MicroServices on the JVM

a practical overview (JavaOne 2014)

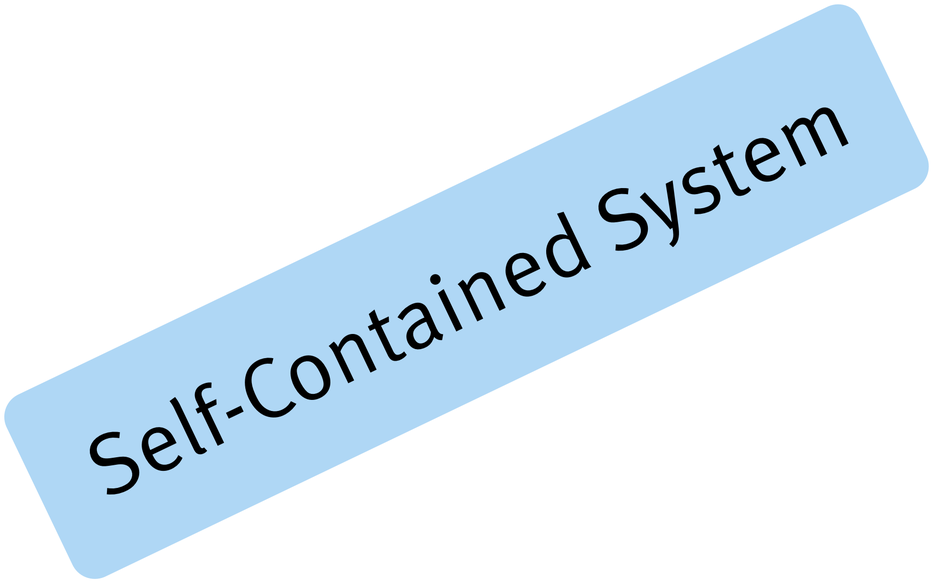
Martin Eigenbrodt — Senior Consultant
Alexander Heusingfeld — Senior Consultant

#MicroServices #buzz

- Defined by LOC?
- UI or not UI?
- SOA in new clothes?

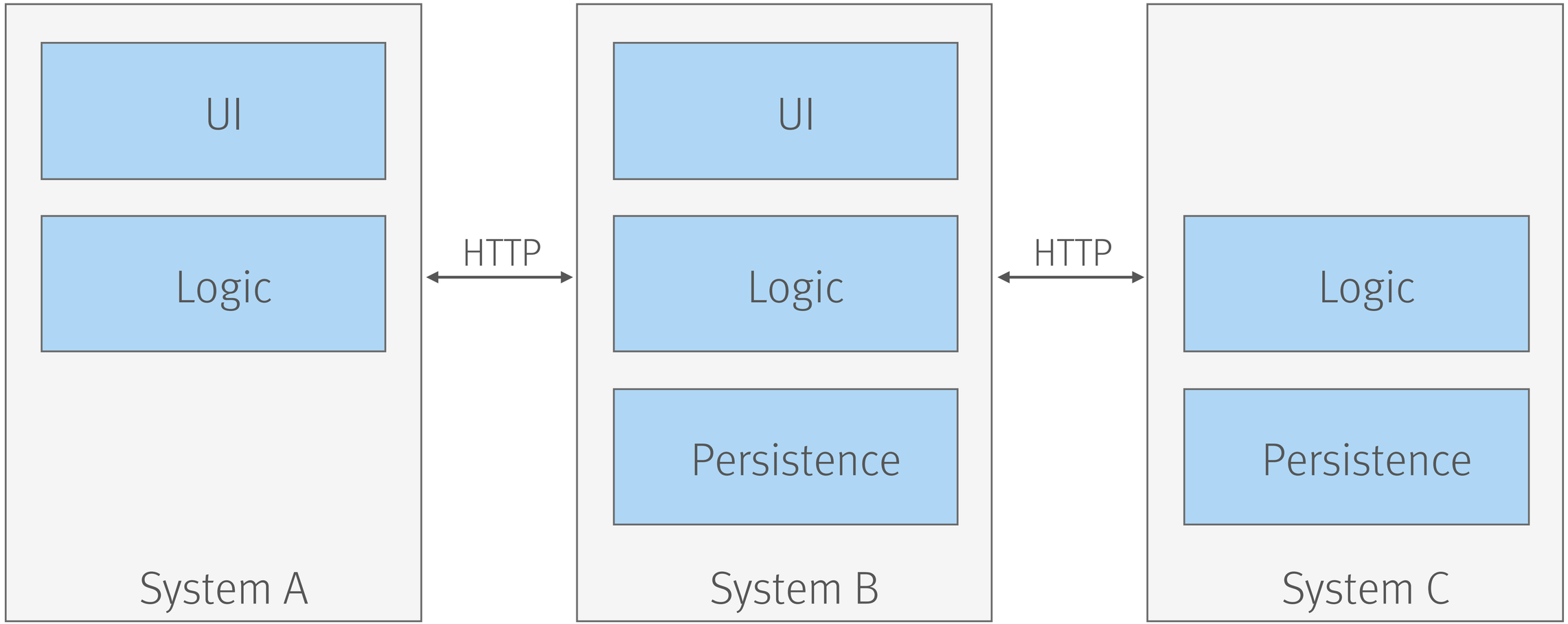
Important for us

- build around business capabilities
- separate runtime processes
- self-contained
- loosely coupled
- independent life cycles
- decentralised management



We’re not talking about…

- micro architecture
- slicing howtos
- communication protocol


UI
Logic
HTTP
UI
Logic
Persistence
HTTP
Logic
Persistence
System A
System B
System C

Challenges

- service registration & discovery
- resilience
- distributed configuration
- fast, automated deployment
- metrics

A Business Use Case

buy something online

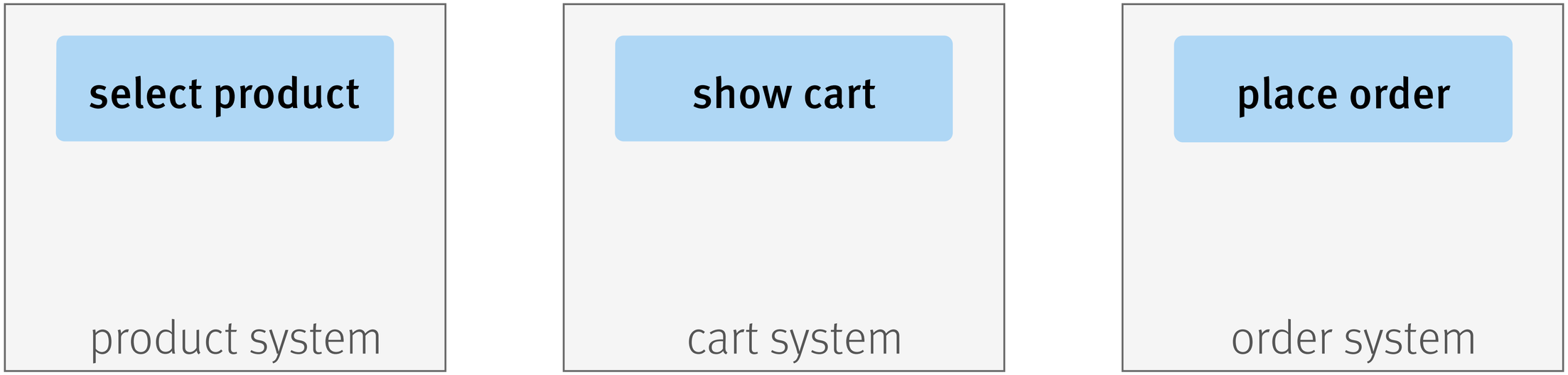
select product

show cart

place order

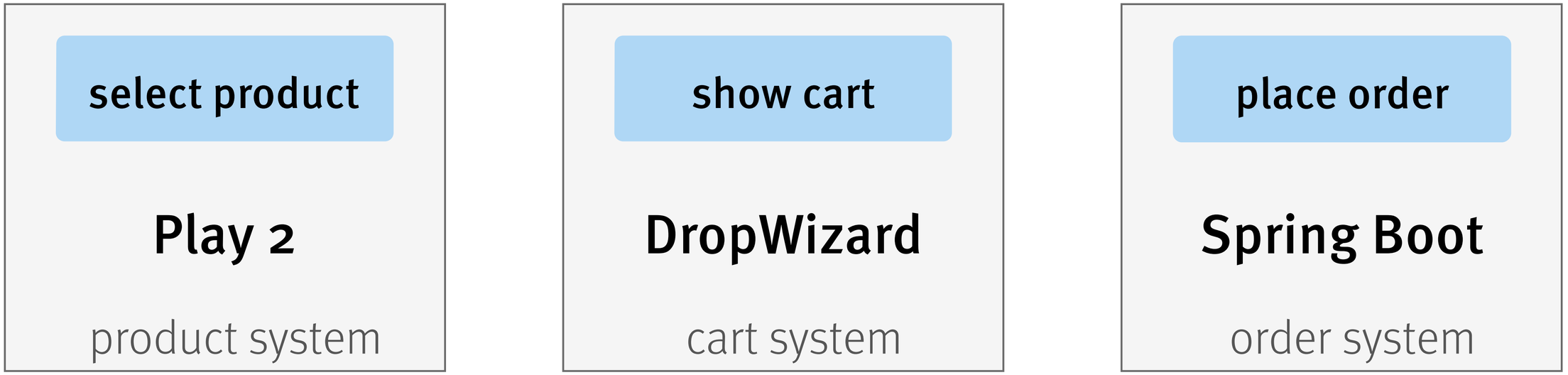
A Business Use Case

buy something online



A Business Use Case

buy something online



Dropwizard

the cart system

Dropwizard

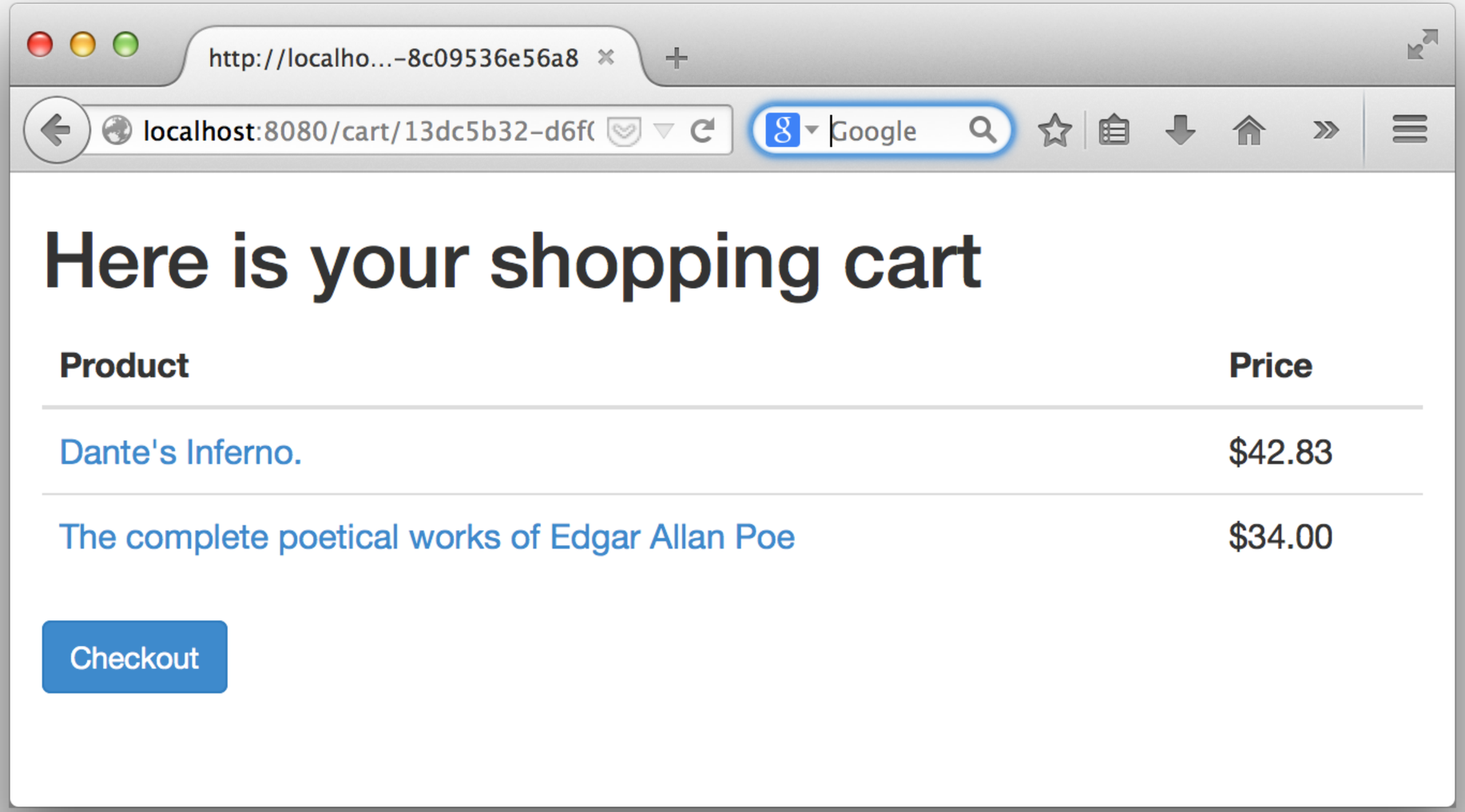
- Glue Code for well known libraries
- Java

Dropwizard libraries

- Jetty
- Jersey
- Metrics
- Jackson
- Guava
- Logback
- Hibernate Validator
- Apache Http Client
- JDBI
- Liquibase
- Freemarker & Mustache
- Joda

Dropwizard libraries

- Jetty
- Jersey
- Metrics
- Jackson
- Guava
- Logback
- Hibernate Validator
- Apache Http Client
- JDBI
- Liquibase
- Freemarker & Mustache
- Joda


http://localho...-8c09536e56a8
localhost:8080/cart/13dc5b32-d6f(
Google
Here is your shopping cart
Product
Price
Dante's Inferno.
$42.83
The complete poetical works of Edgar Allan Poe
$34.00
Checkout

Controller

```
@Path ("/cart")
public class ShoppingCartResource {

    @GET
    @Path("{id}")
    @Timed
    @Produces(MediaType.TEXT_HTML)
    public  ShoppingCartView shoppingCart(@PathParam("id") String cartId,
      @Context UriInfo uriInfo) {
      ImmutableList<String> urls = dao.findItemsForCartId(cartId);
      ImmutableList<Product> products = catalog.resolveProducts(urls);
      return new ShoppingCartView(products,
          uriInfo.getAbsolutePath().toString()
          );
    }

  ...
}
```

Http Client

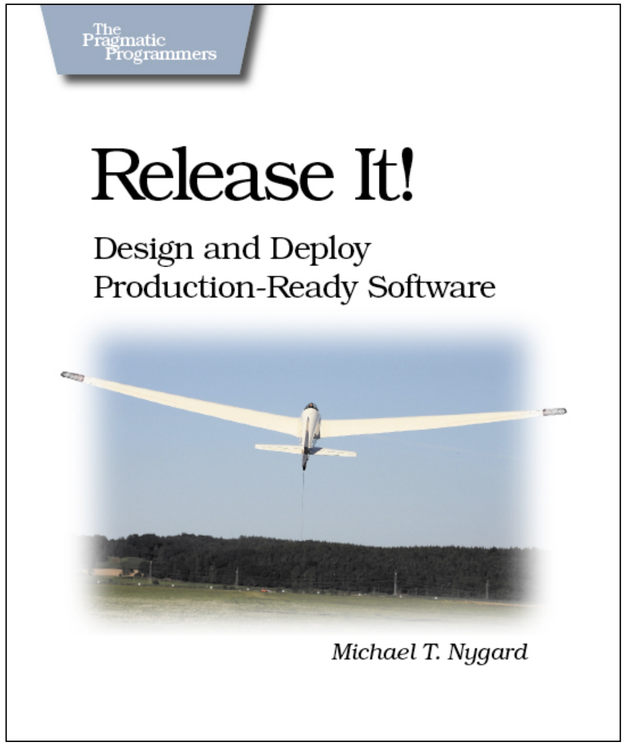
```
public ImmutableList<Product> resolveProducts(List<String> urls) {
  Builder<Product> products = ImmutableList.builder();

  for (String url : urls) {
    Product product = client.resource(url)
        .accept(MediaType.APPLICATION_JSON).get(Product.class);
    products.add(product);
  }
  return products.build();
}
```

Super Duper?

Resilience

- isolate Failure
- apply graceful degradation
- be responsive in case of failure



Request →  *closed* → service

Request ⇄  *open* service

Request ⇄  *half-open* ⇢ service

http://en.wikipedia.org/wiki/Circuit_breaker

HYSTRIX

DEFEND YOUR APP

- Provides Command-oriented Integration of Services
- Introduces Circuit Breaker, Bulkheads and Isolation
- Decouples from Service-dependencies
- Provides metrics-facility to protect from failures

```
public class ResolveProductCommand extends TenacityCommand<Product> {

  private Client client;
  private String url;

    public ResolveProductCommand(Client client, String url) {
        super(DependencyKeys.RESOLVE_PRODUCT);
        this.client = client;
        this.url = url;
    }

    @Override
    protected Product run() throws Exception {
    Product product = client.resource(url)
        .accept(MediaType.APPLICATION_JSON).get(Product.class);
        return product;
    }

    protected Product getFallback() {
        return null
    }
}
```

```
public ImmutableList<Product> resolveProductsHystrix (List<String> urls) {
  Builder<Product> products = ImmutableList.builder();
  for (String url : urls) {
    ResolveProductCommand command = new ResolveProductCommand(client, url);
    Product product = command.execute();
    if (product != null) products.add(product);
  }
  return  products.build();
}
```

Metrics

- Integrated with Dropwizard
- @Timed on Resources
- HTTP Client is already instrumented
- JVM Data

```
"org.apache.http.client.HttpClient.cart.get-requests": {
    "count": 11,
    "max": 0.062107,
    "mean": 0.013355909090909092,
    "min": 0.00575000000000001,
    "p50": 0.009454,
    "p75": 0.010427,
    "p95": 0.062107,
    "p98": 0.062107,
    "p99": 0.062107,
    "p999": 0.062107,
    "stddev": 0.016285873488729705,
    "m15_rate": 0,
    "m1_rate": 0,
    "m5_rate": 0,
    "mean_rate": 2.9714422786532126,
    "duration_units": "seconds",
    "rate_units": "calls/second"
}
```

```
"cart.resources.ShoppingCartResource.shoppingCart": {
    "count": 22,
    "max": 0.136162,
    "mean": 0.01208109090909091,
    "min": 0.00093,
    "p50": 0.008174500000000001,
    "p75": 0.011782250000000001,
    "p95": 0.11783499999999976,
    "p98": 0.136162,
    "p99": 0.136162,
    "p999": 0.136162,
    "stddev": 0.028135302398214 26,
    "m15_rate": 1.8524577712890011,
    "m1_rate": 0.18057796798879996,
    "m5_rate": 1.315746847992022,
    "mean_rate": 0.133050618509084,
    "duration_units": "seconds",
    "rate_units": "calls/second"
}
```

Metrics

- Exposed over HTTP (as Json)
- Exposed as jmx
- Others available: stdout, csv, slf4j, ganglia, graphite

Dropwizard Roundup

- Annotation Driven
- Metrics included
- Hystrix integration

Spring Boot

the order system

Spring Boot

- convention over configuration approach
- Java, Groovy or Scala
- self-contained jar or war
- tackles dependency-hell via pre-packaging

Typical Spring build.gradle

```
dependencies {
    compile 'org.springframework:spring-core:3.2.3.RELEASE'
    compile ‘org.springframework:spring-webmvc:3.2.3.RELEASE'
// ... Jackson, Hibernate Validator, ...
    compile 'org.slf4j:slf4j-api:1.7.5'
    runtime 'org.slf4j:slf4j-log4j12:1.7.5'
    runtime 'log4j:log4j:1.2.16'
    testCompile 'org.springframework:spring-test:3.2.3.RELEASE'
    testCompile 'junit:junit:4.11'
    testCompile ‘org.hamcrest:hamcrest-library:1.3’
    testCompile ‘org.mockito:mockito-core:1.9.5’
}
```

Spring Boot build.gradle

```
dependencies {
    compile ‘org.springframework.boot:spring-boot-starter-web:1.1.6.RELEASE'
    compile ‘org.springframework.boot:spring-boot-starter-test:1.1.6.RELEASE'
}
```

Externalized Configuration

```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

    public static void main(String[] args) {
        SpringApplication.run(OrderApp.class, args);
    }
}
```

- opinionated preset
- HTTP resource “/autoconfig” shows all properties
- overwrite via application.properties or CLI parameter

Spring Boot Metrics

- Prepackaged Spring Boot starter module
- enables HTTP resources for metrics
- configurable via application.properties

http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#production-ready

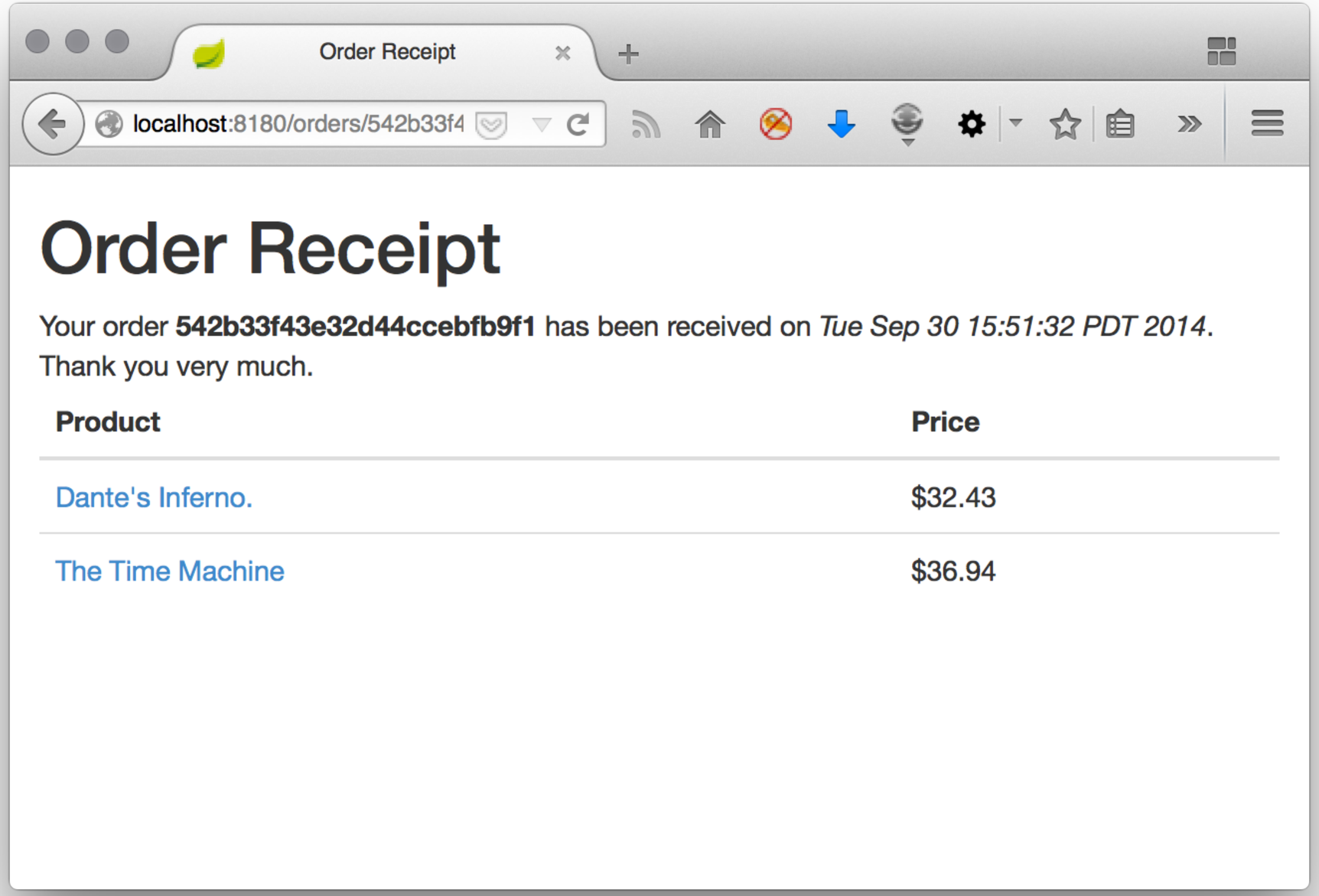
Using a counter metric in your Java code...

```
counterService.increment("checkouts.withproducts." + productUris.size());
```

...will display it in the /metrics JSON

```
GET /metrics HTTP/1.1
{
    "counter.checkouts.withproducts.3": 4,
    ...
}
```

How does it look on the inside?


Order Receipt
localhost:8180/orders/542b33f4
Order Receipt
Your order 542b33f43e32d44ccebfb9f1 has been received on Tue Sep 30 15:51:32 PDT 2014.
Thank you very much.
Product
Price
Dante's Inferno.
$32.43
The Time Machine
$36.94

```
@Controller
@RequestMapping(value = ORDERS_RESOURCE)
public class OrderController {

    private @Autowired CounterService counterService;
    private @Autowired OrderRepository repository;

   @RequestMapping(method = RequestMethod.POST)
    public String checkoutCart(@RequestParam(value = "products") List<String> productUris) {
       counterService.increment("checkouts.withproducts." + productUris.size());

       final List<Product> products = productUris.stream()
                    .map(productUri -> CompletableFuture.supplyAsync(() -> resolveProducts(productUri)))
                    .map(CompletableFuture::join)
                    .filter(res -> res.second.hasBody())
                    .map(this::responsePairToProduct)
                    .collect(Collectors.toList());

       final Order savedOrder = repository.save(new Order(products));
       return "redirect:" ORDERS_RESOURCE + savedOrder.getId();
    }

   private Pair<String, ResponseEntity<Product>> resolveProducts(String productUri) {
        final RestTemplate restTemplate = new RestTemplate();
       return new Pair(productUri, restTemplate.getForEntity(productUri, Product.class));
    }
}
```

Resilience

```
final List<Product> products = productUris.stream()
            .map(productUri -> CompletableFuture.supplyAsync(() -> resolveProducts(productUri)))
            .map(CompletableFuture::join)
            .filter(res -> res.second.hasBody())
            .map(this::responsePairToProduct)
            .collect(Collectors.toList());
```

```
private Pair<String, ResponseEntity<Product>> resolveProducts(String productUri) {
    final RestTemplate restTemplate = new RestTemplate();
   return new Pair(productUri, restTemplate.getForEntity(productUri, Product.class));
}
```

Resilience

```
final List<Product> products = productUris.stream()
            .map(productUri -> CompletableFuture.supplyAsync(() -> resolveProducts(productUri)))
            .map(CompletableFuture::join)
            .filter(res -> res.second.hasBody())
            .map(this::responsePairToProduct)
            .collect(Collectors.toList());
```

Resilient?

```
private Pair<String, ResponseEntity<Product>> resolveProducts(String productUri) {
    final RestTemplate restTemplate = new RestTemplate();
   return new Pair(productUri, restTemplate.getForEntity(productUri, Product.class));
}
```

Spring Cloud

- new umbrella project for cloud connectors
- On top of Spring Boot
- config server for distributed configuration
- Eureka annotations for service-discovery
- Hystrix annotations for resilience enhancements

Spring Cloud Hystrix

wrapped in command -› async!

```
@HystrixCommand(fallbackMethod = “fallbackProduct")
private Pair<String, ResponseEntity<Product>> resolveProduct(String productUri) {
    final RestTemplate restTemplate = new RestTemplate();
    return new Pair(productUri, restTemplate.getForEntity(productUri, Product.class));
}
```

```
private Pair<String, ResponseEntity<Product>> fallbackProduct(String productUri) {
    final Product product = new Product(productUri, null, BigDecimal.ZERO);
    final ResponseEntity<Product> response = new ResponseEntity<Product>(product, PARTIAL_CONTENT);
    return new Pair(productUri, response);
}
```

Spring Cloud Hystrix

method reference? :(

```
@HystrixCommand(fallbackMethod = “fallbackProduct")
private Pair<String, ResponseEntity<Product>> resolveProduct(String productUri) {
    final RestTemplate restTemplate = new RestTemplate();
    return new Pair(productUri, restTemplate.getForEntity(productUri, Product.class));
}


private Pair<String, ResponseEntity<Product>> fallbackProduct(String productUri) {
    final Product product = new Product(productUri, null, BigDecimal.ZERO);
    final ResponseEntity<Product> response = new ResponseEntity<Product>(product, PARTIAL_CONTENT);
    return new Pair(productUri, response);
}
```

Spring Boot Roundup

- focussed on Spring ecosystem
- externalized configuration
- metrics
- Spring Cloud still beyond bleeding edge

Play 2

the product catalog

Play 2

- Java or Scala
- based on Akka
- strong async support

Your micro Bookstore

localhost:9000

Google

Best Seller

EDGE OF ETERNITY

by Ken Follett

NY-Times Api

Our Books

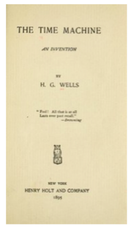
Persistence

Dante's Inferno.

33.65

Dante Alighieri

Add to shopping cart

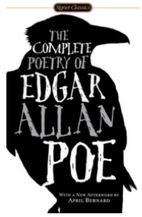


The Time Machine

49.36

H. G. Wells

Add to shopping cart

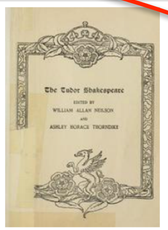


The complete poetical works of Edgar Allan Poe

46.22

Edgar Allan Poe

Add to shopping cart



The tragedy of Romeo and Juliet.

24.68

William Shakespeare

Add to shopping cart

Your micro Bookstore

localhost:9000

Google

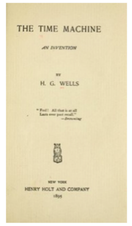
Our Books

Dante's Inferno.

45.12

Dante Alighieri

Add to shopping cart

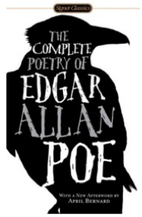


The Time Machine

40.14

H. G. Wells

Add to shopping cart

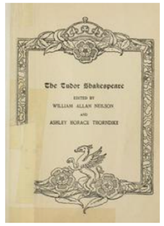


The complete poetical works of Edgar Allan Poe

20.88

Edgar Allan Poe

Add to shopping cart



The tragedy of Romeo and Juliet.

37.22

William Shakespeare

Add to shopping cart

Routing

```
# Routes
#  List of all books
GET     /                                   controllers.BooksController.books
# A specific Book
GET    /books/:id          controllers.BooksController.book(id:String)
```

Controller

```
object BooksController extends Controller {

  def books = Action.async { implicit request =>
    val bestsellerFuture = Bestseller.getBestseller
    val booksFuture = Books.books
    for {
      bestseller <- bestsellerFuture
      books <- booksFuture
    } yield {
      Ok(views.html.books(books.values.toList, bestseller))
    }
  }
  ...
}
```

Calling an external Service

```
WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }
```

Handle Failure

```
WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }.recover { case e => List() }
```

Circuit Breaker

```
val apiUrl = "..."
val breaker =
  CircuitBreaker(Akka.system().scheduler,
    maxFailures = 5,
    callTimeout = 2.seconds,
    resetTimeout = 1.minute)

def getBestseller : Future[List[Bestseller]] = {
  breaker.withCircuitBreaker(
    WS.url(apiUrl).get.map {
      response => response.json.as[List[Bestseller]]
    }).recover { case e => List() }
}
```

Packaging

```
sbt dist
sbt debian:packageBin
sbt rpm:packageBin
```

Play roundup

- smart syntax for async http, recovery, circuit breaker in scala
- easy packaging
- no build in support for metrics
- no build in support service discovery or distributed config

Conclusion

Take Aways

- MicroServices aren't micro!
- Only "micro" regarding business scope
- micro service = distributed systems deep dive
- no framework is a silver bullet

Thank you!

Questions?

Comments?

Martin Eigenbrodt, @eigenbrodtm
Alexander Heusingfeld, @goldstift

https://www.innoq.com/en/timeline/

#con4952

innoQ
www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0

Ohlauer Straße 43
10999 Berlin
Germany
Phone: +49 2173 3366-0

Robert-Bosch-Straße 7
64293 Darmstadt
Germany
Phone: +49 2173 3366-0

Radlkoferstraße 2
D-81373 München
Germany
Telefon +49 (0) 89 741185-270

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116