DEVOXX™

MicroServices on the JVM

A practical overview

Alexander Heusingfeld & Martin Eigenbrodt

innoQ

#MicroServices #buzz

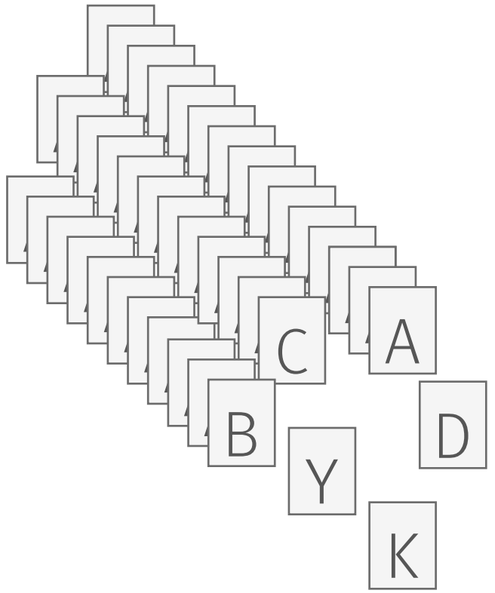
- Defined by LOC?
- UI or not UI?
- SOA in new clothes?

“Micro” Services?

“Micro” Services?

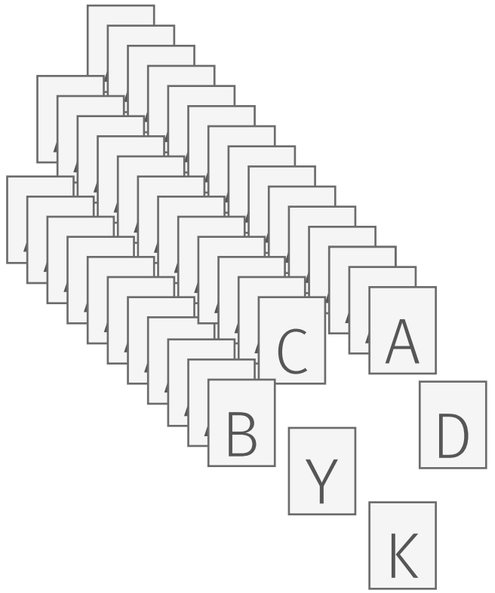
“Micro” Services?

“Micro” Services?

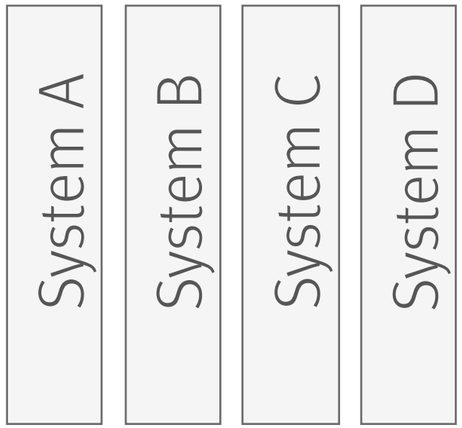


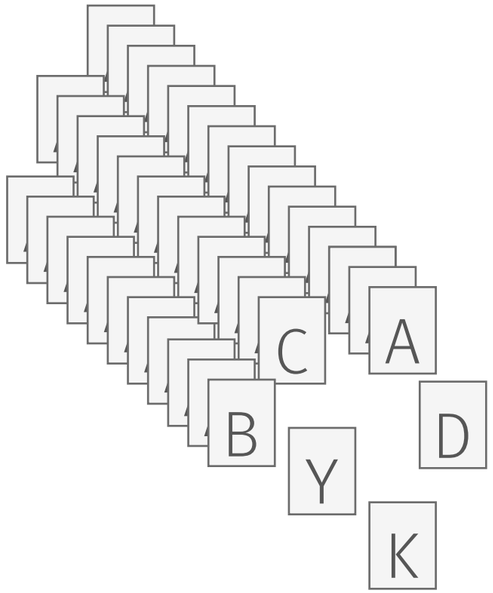
😭

“Micro” Services?

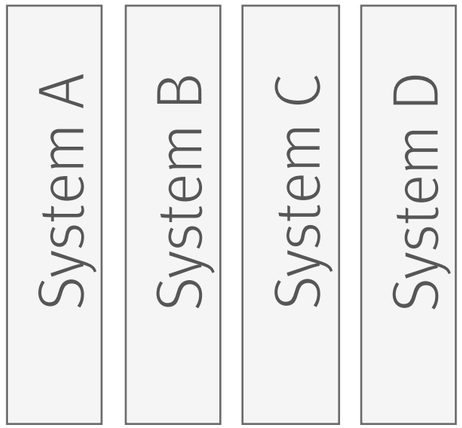


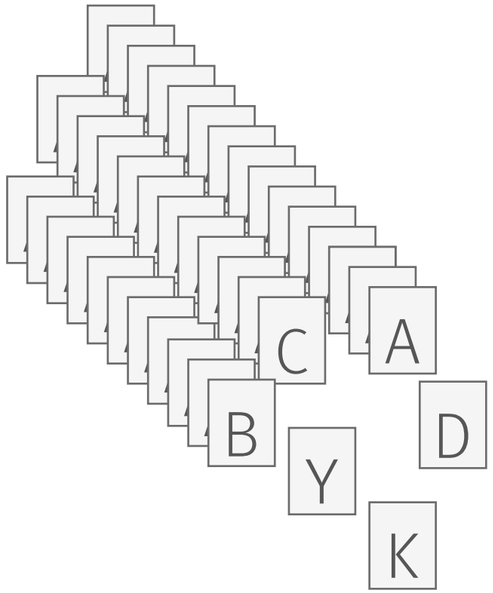
“Micro” Services?





“Micro” Services?





😭 😀 😱

Our approach

Our approach

- build around business capabilities

Our approach

- build around business capabilities
- separate runtime processes

Our approach

- build around business capabilities
- separate runtime processes
- self-contained

Our approach

- build around business capabilities
- separate runtime processes
- self-contained
- loosely coupled

Our approach

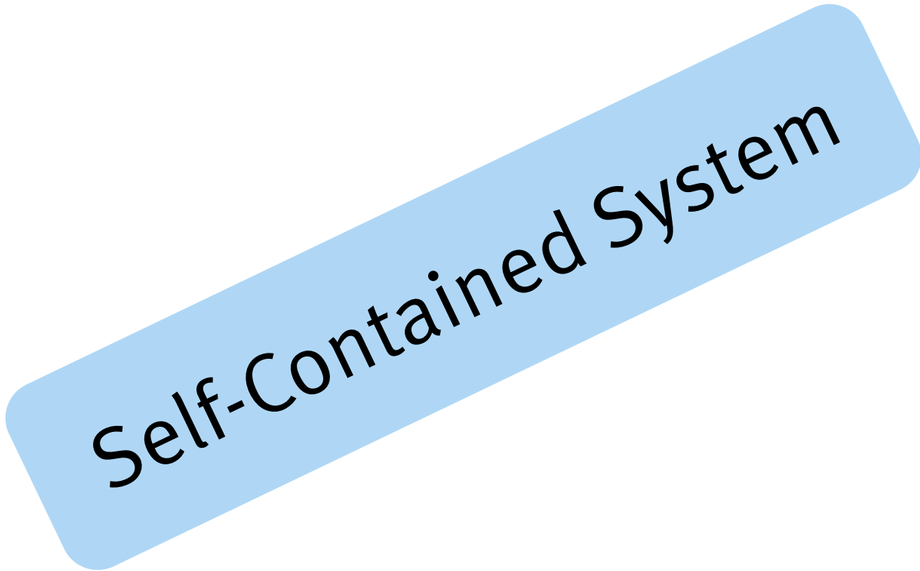
- build around business capabilities
- separate runtime processes
- self-contained
- loosely coupled
- independent life cycles

Our approach

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

Our approach

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



We’re not talking about…

We’re not talking about…

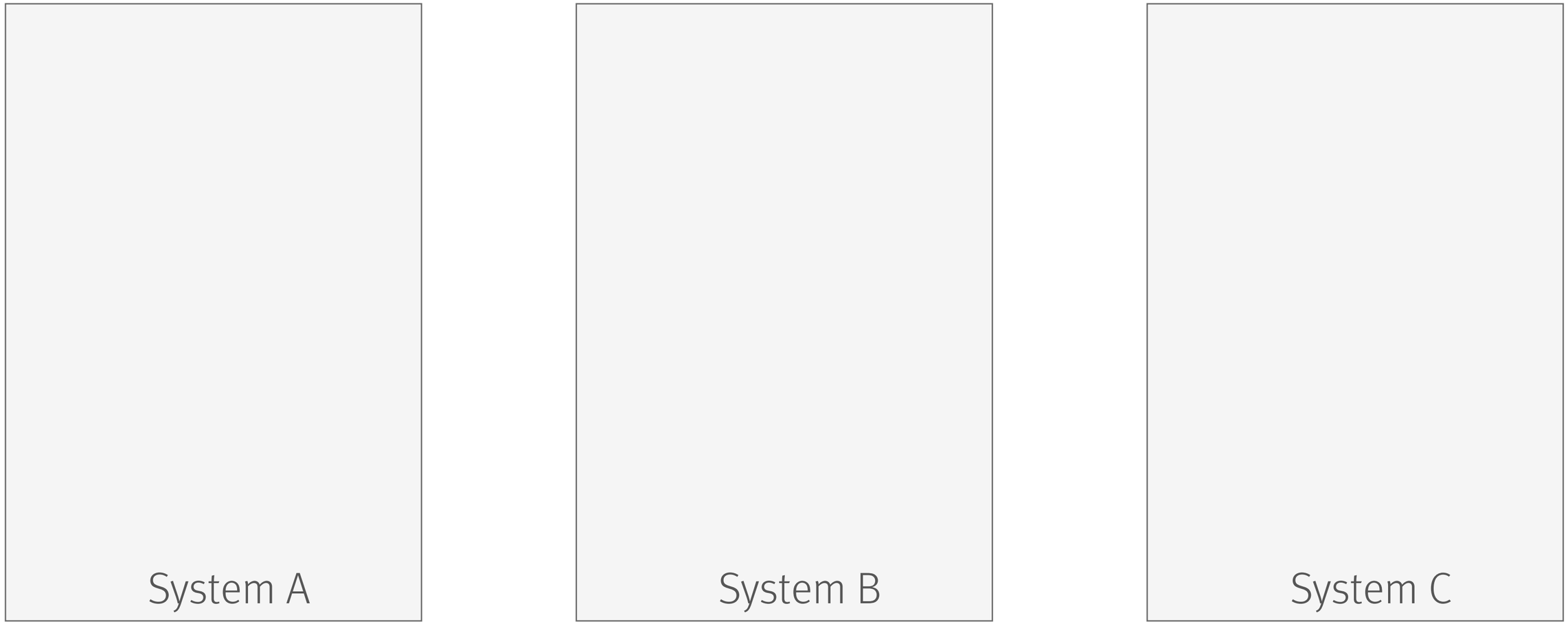
- micro architecture

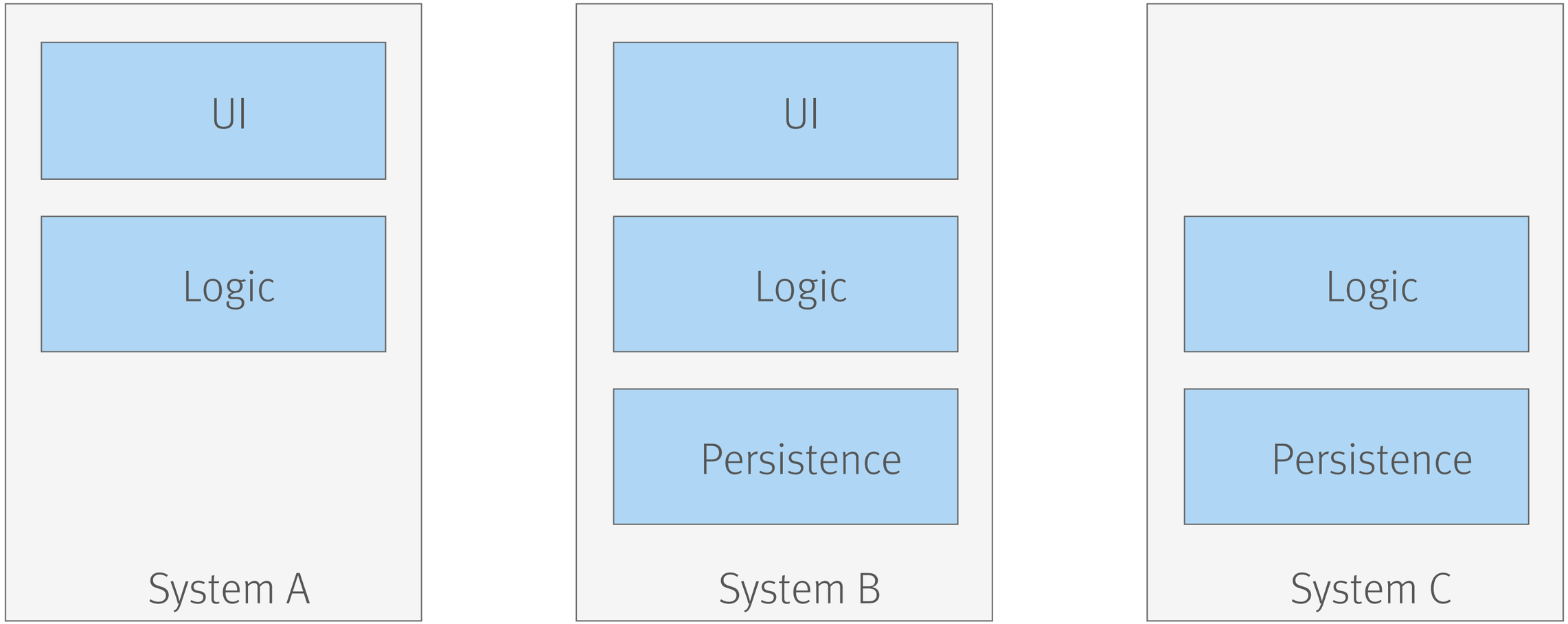
We're not talking about…

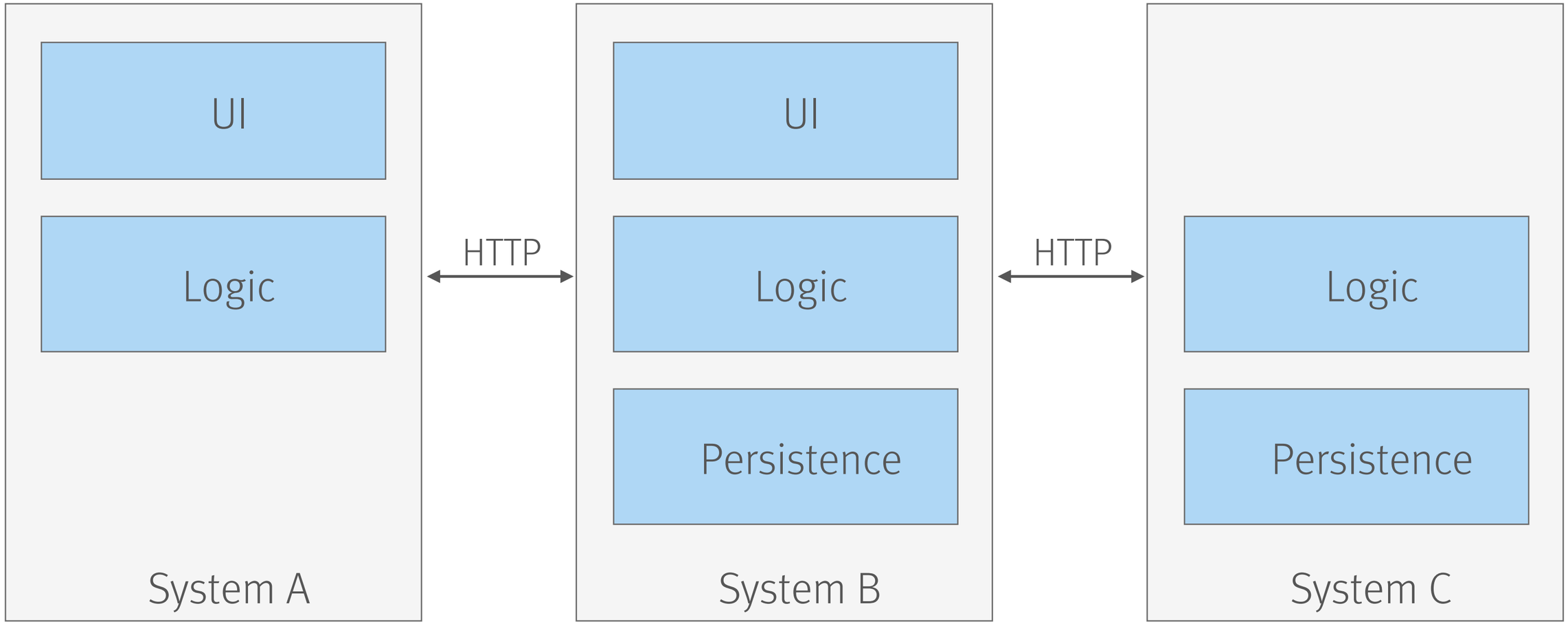
- micro architecture
- slicing howtos

We're not talking about…

- micro architecture
- slicing howtos
- communication protocol


System A
System B
System C


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


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

Challenges

Challenges

- distributed configuration

Challenges

- distributed configuration
- service registration & discovery

Challenges

- distributed configuration
- service registration & discovery
- resilience

Challenges

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

Challenges

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

Frameworks

Dropwizard

Dropwizard

- Glue Code for well known libraries
- Java

Dropwizard libraries

Dropwizard libraries

- Jetty

Dropwizard libraries

- Jetty
- Jersey

Dropwizard libraries

- Jetty
- Jersey
- Metrics

Dropwizard libraries

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

Spring Boot

Spring Boot

Spring Boot

- convention over configuration approach

Spring Boot

- convention over configuration approach
- Java, Groovy or Scala

Spring Boot

- convention over configuration approach
- Java, Groovy or Scala
- self-contained jar or war

Spring Boot

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

Spring Cloud

Spring Cloud

- new umbrella project for cloud connectors

Spring Cloud

- new umbrella project for cloud connectors
- On top of Spring Boot

Spring Cloud

- new umbrella project for cloud connectors
- On top of Spring Boot
- config server for distributed configuration

Spring Cloud

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

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

Play 2

Play 2

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

Routing

Dropwizard

```
@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()
           );
  }
  ...
}
```

Spring Boot

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

    private @Autowired CounterService counterService;


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

      // save products as an order

      return "redirect:" ORDERS_RESOURCE + order.getId(); // like “/order/123”
    }
}
```

Play

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

Play

```
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))
    }
  }
  ...
}
```

Configuration

Spring Boot

```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

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

Spring Boot

```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

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

Spring Boot

- opinionated preset

```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

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

Spring Boot

- opinionated preset
- HTTP resource “/env” shows all properties

```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

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

Spring Boot

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

```
@ComponentScan
@EnableAutoConfiguration
public class OrderApp {

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

Play - Typesafe Config

Play - Typesafe Config

- Config Library used by akka, play and other

Play - Typesafe Config

- Config Library used by akka, play and other
- HOCON - JSON Data Model + syntactic sugar

Play - Typesafe Config

- Config Library used by akka, play and other
- HOCON - JSON Data Model + syntactic sugar
- override via system property

Play - Typesafe Config

- Config Library used by akka, play and other
- HOCON - JSON Data Model + syntactic sugar
- override via system property
- rich merge and include possibilities

DEVOXX™

Http Client

Dropwizard

```
public Product resolveProduct(String url) {
   Product product = client.resource(url)
          .accept(MediaType.APPLICATION_JSON).get(Product.class);
   return product;
}
```

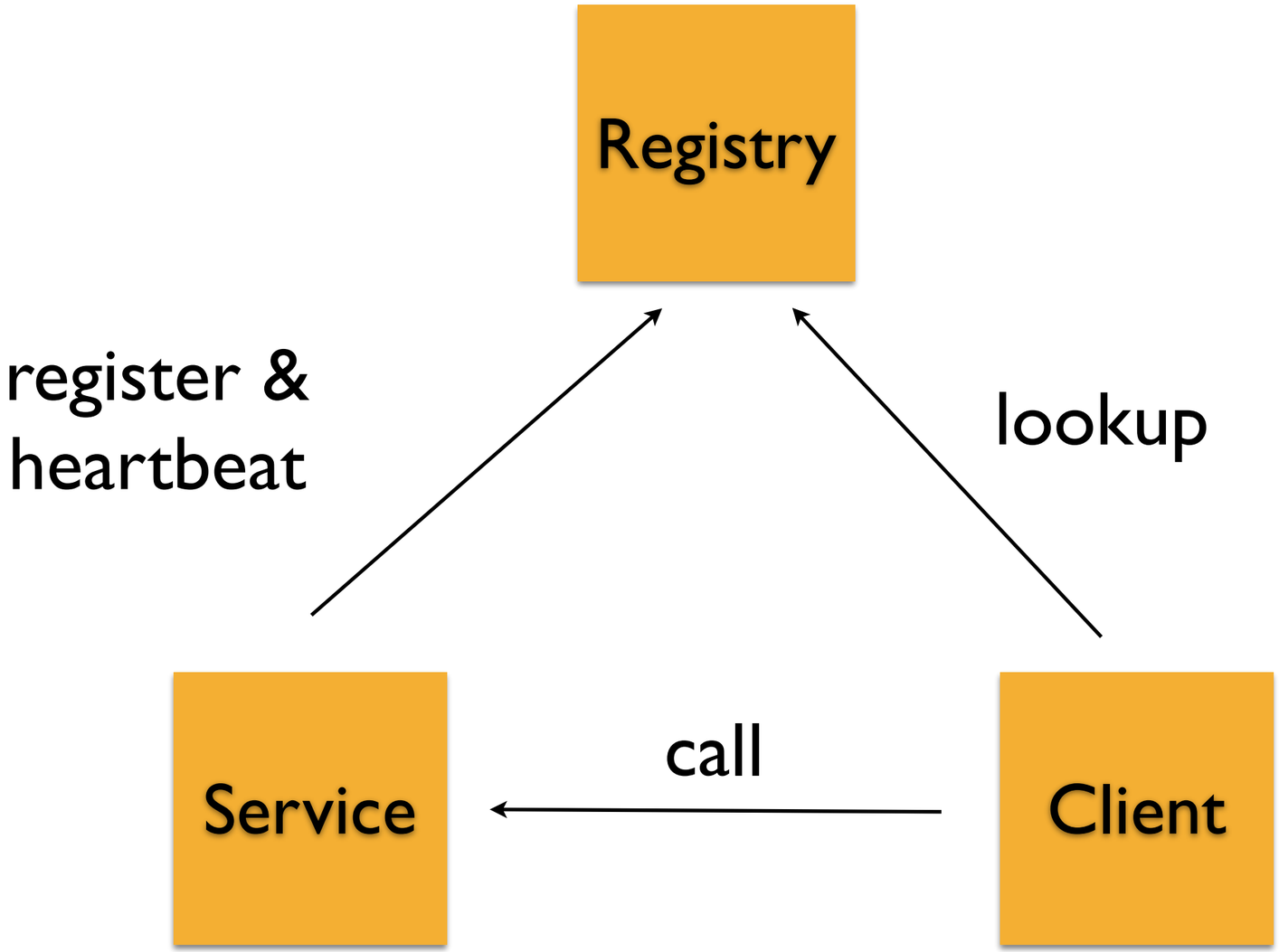
Spring Boot

```
public Product resolveProduct(String url) {
  final RestTemplate restTemplate = new RestTemplate();
  return restTemplate.getForEntity(url, Product.class);
}
```

Play

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

Service Discovery


Registry
register &
heartbeat
lookup
Service
call
Client

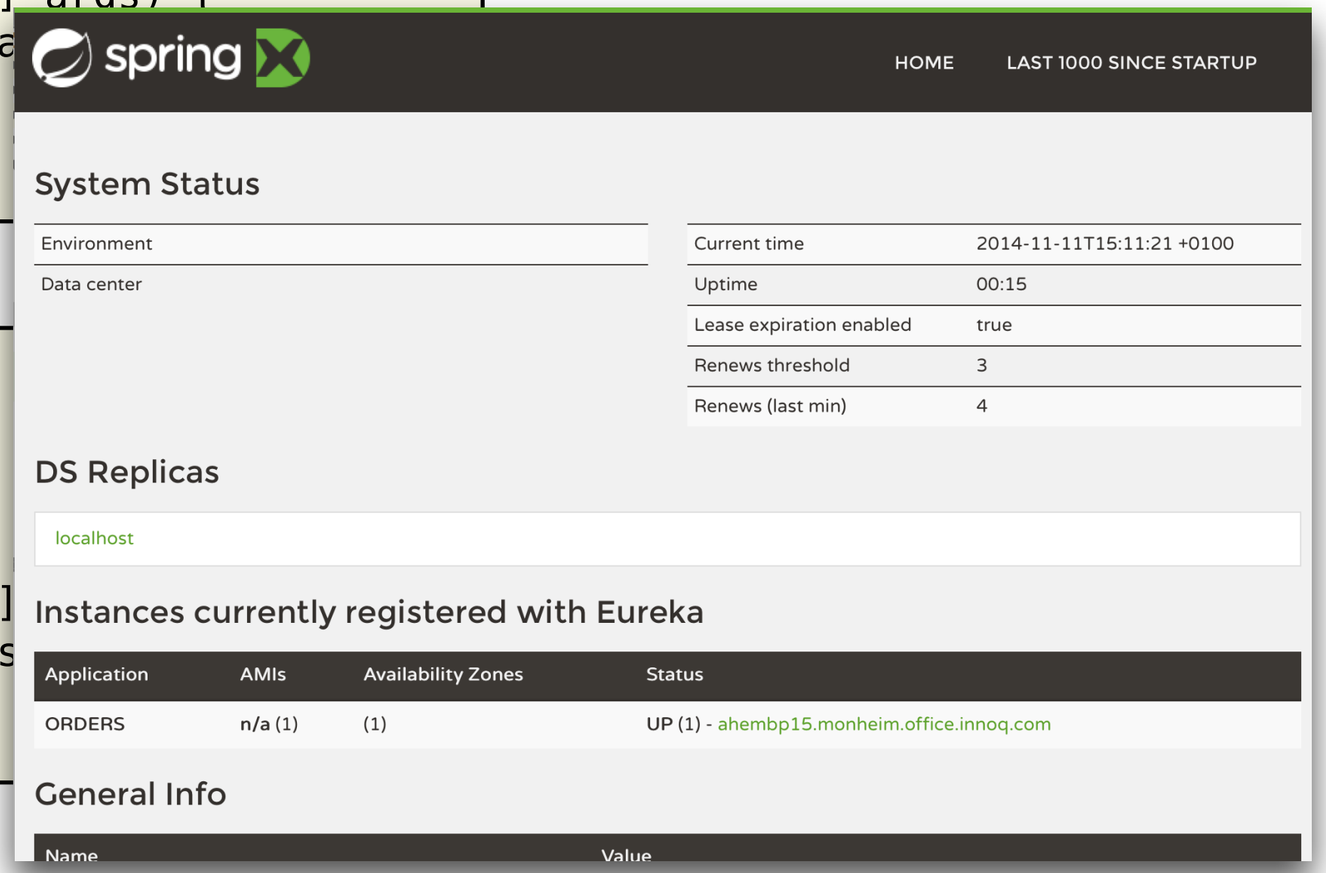
```
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaServer
public class EurekaServerApp {

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

```
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaServer
public class EurekaServerApp {

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

```
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaClient
public class OrdersApp {

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

```java
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaServer
public class EurekaServerApp {

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

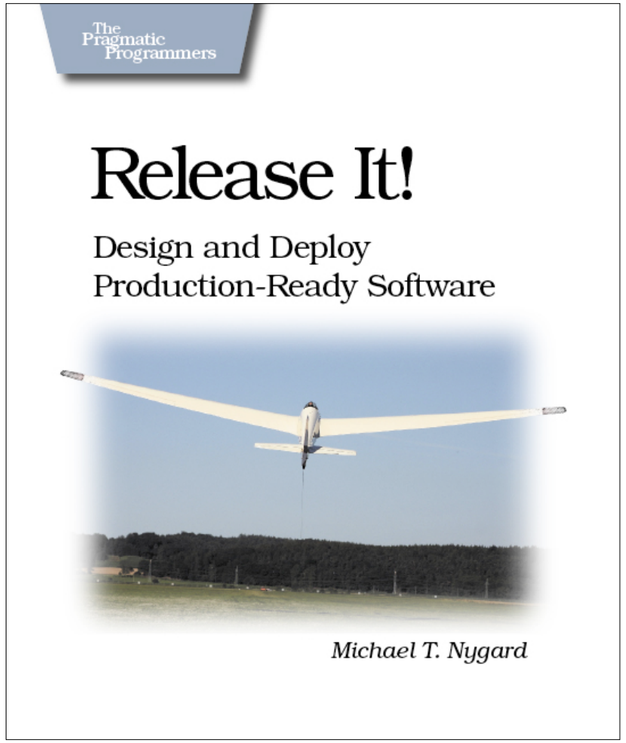
```java
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaClient
public class OrdersApp {

    public static void main(String[]
        SpringApplication.run(Orders
    }
}
```



Resilience

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

Hystrix & Dropwizard

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

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

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

Hystrix & Dropwizard

```
ResolveProductCommand command = new ResolveProductCommand(client, url);
Product product = command.execute();
```

Spring Cloud Hystrix

```
@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

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

```
@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

```
@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);
}
```

Play - 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() }
}
```

Deployment

Spring Boot - Packaging

```
./gradlew distZip
./gradlew build
```

Spring Boot - Packaging

```
./gradlew distZip
./gradlew build
```

ZIP + shell-script

Spring Boot - Packaging

```
./gradlew distZip
./gradlew build
```

ZIP + shell-script

executable JAR

Play - Packaging

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

Metrics

Dropwizard

- “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": 0.005750000000000001,
        "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"
    }
```

```
oppingCartResource.shoppingCart": {
2,
8109090909091,
,
4500000000001,
2250000000001,
499999999976,
2,
2,
62,
813530239821426,
8524577712890011,
8057796798879996,
        "m5_rate": 1.315746847992022,
        "mean_rate": 0.133050618509084,
        "duration_units": "seconds",
        "rate_units": "calls/second"
    }
```

Dropwizard Metrics

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

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

DEVOXX TM

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,
    ...
}
```

Conclusion

Take Aways

Take Aways

- MicroServices aren't micro!

Take Aways

- MicroServices aren't micro!
- Only "micro" regarding business scope

Take Aways

- MicroServices aren't micro!
- Only "micro" regarding business scope
- micro services = distributed systems deep dive

Take Aways

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

Thanks for your attention

- Questions?

Martin Eigenbrodt, @eigenbrodtm
Alexander Heusingfeld, @goldstift

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

innoQ