

Cloud Native Application Architecture Workshop

sesión

two



Building and Composing Cloud Native Applications

Building Twelve-Factor Apps with Spring Boot

<http://12factor.net>

Patterns

- Cloud-native application architectures
- Optimized for speed, safety, & scale
- Declarative configuration
- Stateless/shared-nothing processes
- Loose coupling to application environment

Twelve Factors (1/2)

- One Codebase in Version Control
- Explicit Dependencies
- Externalized Config
- Attached Backing Services
- Separate Build, Release, and Run Stages
- Stateless, Shared-Nothing Processes

Twelve Factors (2/2)

- Export Services via Port binding
- Scale Out Horizontally for Concurrency
- Instances Should Be Disposable
- Dev/Prod Parity
- Logs Are Event Streams
- Admin Processes

<http://heroku.com>

<http://cloudfoundry.org>

Microframeworks

- Dropwizard (<http://www.dropwizard.io/>)
- Spring Boot (<http://projects.spring.io/spring-boot/>)



Spring Boot

- <http://projects.spring.io/spring-boot>
- Opinionated convention over configuration
- Production-ready Spring applications
- Embed Tomcat, Jetty or Undertow
- STARTERS
- Actuator: Metrics, health checks, introspection

<http://start.spring.io>

Writing a Single Service is
Nice

But No Microservice
is an Island



Challenges of Distributed Systems

- Configuration Management
- Service Registration & Discovery
- Routing & Load Balancing
- Fault Tolerance (Circuit Breakers!)
- Monitoring



Spring Cloud

Distributed System Patterns FTW!

Configuration Management

Distributed?

**config
server!**



Config Server app.groovy

```
@Grab("org.springframework.cloud:spring-cloud-starter-bus-amqp:1.0.0.RC1")
@EnableConfigServer
class ConfigServer {  
}
```

Config Server application.yml

```
server:  
  port: 8888  
  
spring:  
  cloud:  
    config:  
      server:  
        git:  
          uri: https://github.com/mstine/config-repo.git
```

<https://github.com/mstine/config-repo/blob/master/demo.yml>

greeting: Bonjour



Config Client app.groovy

```
@Grab("org.springframework.cloud:spring-cloud-starter-bus-amqp:1.0.0.RC1")
@RestController
class BasicConfig {

    @Autowired
    Greeter greeter

    @RequestMapping("/")
    String home() {
        "${greeter.greeting} World!"
    }
}

@Component
@RefreshScope
class Greeter {

    @Value('${greeting}')
    String greeting

}
```

Config Client bootstrap.yml

```
spring:  
  application:  
    name: demo
```

Cloud
Bus!





```
curl -X POST http://localhost:8080/bus/refresh
```

Service Registration & Discovery



Foureká

producer
consumer

Eureka Service Registry

```
@GrabExclude("ch.qos.logback:logback-classic")
@EnableEurekaServer
class Eureka {
}
```

Producer

```
@EnableDiscoveryClient
@RestController
public class Application {

    int counter = 0

    @RequestMapping("/")
    String produce() {
        "{\"value\": ${counter++}}"
    }
}
```

Consumer

```
@EnableDiscoveryClient
@RestController
public class Application {

    @Autowired
    DiscoveryClient discoveryClient

    @RequestMapping("/")
    String consume() {
        InstanceInfo instance = discoveryClient.getNextServerFromEureka("PRODUCER", false)

        RestTemplate restTemplate = new RestTemplate()
        ProducerResponse response = restTemplate.getForObject(instance.homePageUrl, ProducerResponse.class)

        "{\"value\": ${response.value}}"
    }
}

public class ProducerResponse {
    Integer value
}
```

Routing & Load Balancing



Good morning

Consumer with Load Balancer

```
@Autowired  
LoadBalancerClient loadBalancer  
  
@RequestMapping("/")  
String consume() {  
    ServiceInstance instance = loadBalancer.choose("producer")  
    URI producerUri = URI.create("http://${instance.host}:${instance.port}");  
  
    RestTemplate restTemplate = new RestTemplate()  
    ProducerResponse response = restTemplate.getForObject(producerUri, ProducerResponse.class)  
  
    "{\"value\": ${response.value}}"  
}
```

Consumer with Ribbon-enabled RestTemplate

```
@Autowired  
RestTemplate restTemplate  
  
@RequestMapping("/")  
String consume() {  
    ProducerResponse response = restTemplate.getForObject("http://producer", ProducerResponse.class)  
    "{\"value\": ${response.value}}"  
}
```

Feign Client

```
@FeignClient("producer")
public interface ProducerClient {
    @RequestMapping(method = RequestMethod.GET, value = "/")
    ProducerResponse getValue();
}
```

Consumer with Feign Client

```
@SpringBootApplication
@EnableFeignClients
@EnableDiscoveryClient
@RestController
public class Application {

    @Autowired
    ProducerClient client;

    @RequestMapping("/")
    String consume() {
        ProducerResponse response = client.getValue();

        return "{\"value\": " + response.getValue() + "}";
    }

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

Fault

Tolerance

Follow us at
@augustusticks
#Kix

Circuit Breaker



Consumer app . groovy

```
@EnableDiscoveryClient
@EnableCircuitBreaker
@RestController
public class Application {

    @Autowired
    ProducerClient client

    @RequestMapping("/")
    String consume() {
        ProducerResponse response = client.getProducerResponse()

        "{\"value\": ${response.value}}"
    }

}
```

Producer Client

```
@Component
public class ProducerClient {

    @Autowired
    RestTemplate restTemplate

    @HystrixCommand(fallbackMethod = "getProducerFallback")
    ProducerResponse getProducerResponse() {
        restTemplate.getForObject("http://producer", ProducerResponse.class)
    }

    ProducerResponse getProducerFallback() {
        new ProducerResponse(value: 42)
    }
}
```

Monitoring

Augストリックス Dashboard

Hystrix Dashboard



Hystrix Dashboard

```
@Grab("org.springframework.cloud:spring-cloud-starter-hystrix-dashboard:1.0.0.RC1")  
  
import org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard  
  
@EnableHystrixDashboard  
class HystrixDashboard {  
}
```

TO THE

LAARS!