Spring Cloud, Spring Boot and Netflix OSS

Spencer Gibb
twitter: @spencerbgibb
email: sgibb@pivotal.io

Dave Syer
twitter: @david_syer
email: dsyer@pivotal.io

(*Spring Boot and Netflix OSS*
or *Spring Cloud Components*)

Outline

- Define microservices
- Outline some distributed system problems
- Introduce Netflix OSS and its integration with Spring Boot
- Spring Cloud demos

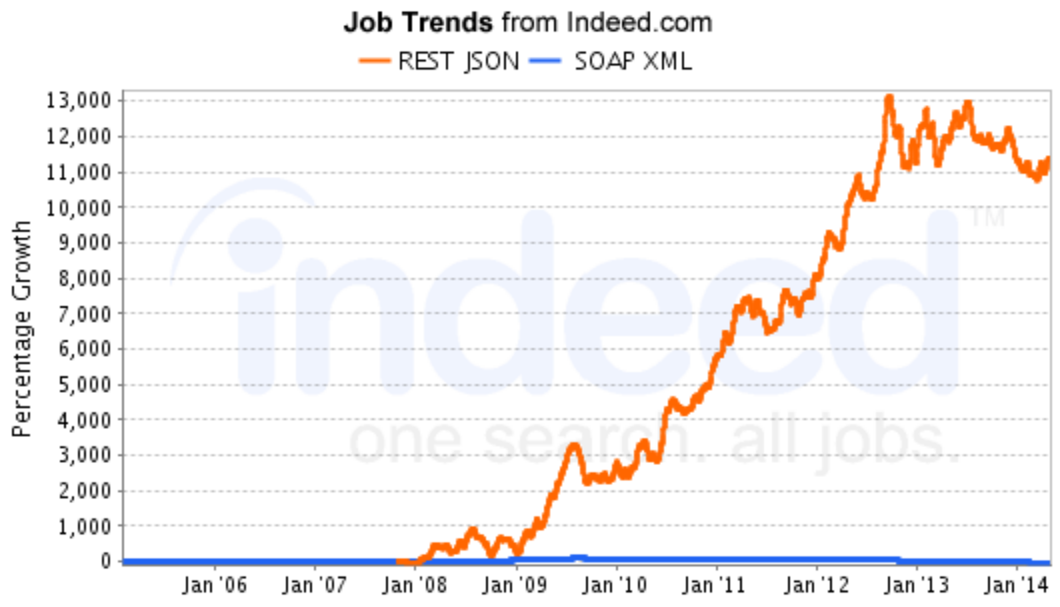
What are micro-services?

- Not monolithic :-)
- Smaller units of a larger system
- Runs in its own process
- Lightweight communication protocols
- Single Responsibility Principle
- The UNIX way

http://www.slideshare.net/ewolff/micro-services-small-is-beautiful
http://martinfowler.com/articles/microservices.html
http://davidmorgantini.blogspot.com/2013/08/micro-services-what-are-micro-services.html

Lightweight Services and REST

- There is a strong trend in distributed systems with lightweight architectures
- People have started to call them "microservices"



Spring Boot

It needs to be super easy to implement and update a service:

```
@RestController
class ThisWillActuallyRun {
    @RequestMapping("/")
    String home() {
        Hello World!
    }
}
```

and you don't get much more "micro" than that.

Cloudfoundry

Deploying services needs to be simple and reproducible

```
$ cf push app.groovy
```

and you don't get much more convenient than that.

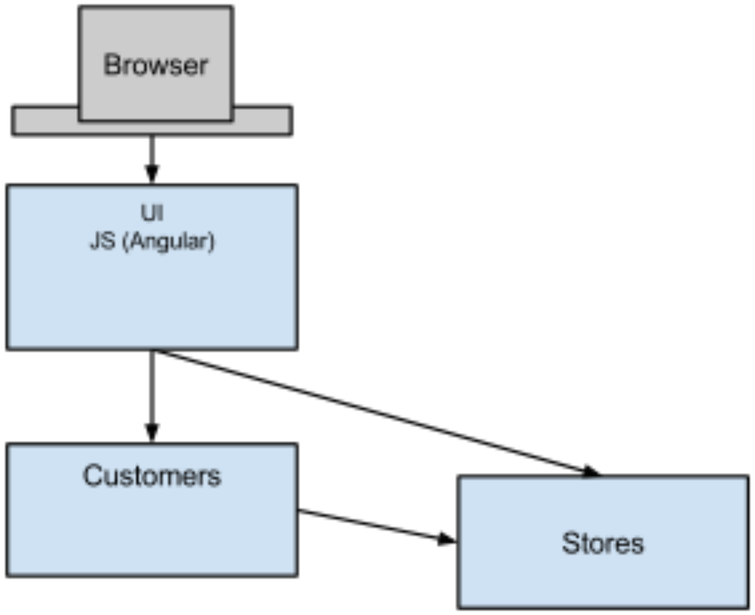
(Same argument for other PaaS solutions)

Continuous Delivery

- Microservices lend themselves to continuous delivery.
- You *need* continuous delivery

Book (Humble and Farley): http://continuousdelivery.com Netflix Blog: http://techblog.netflix.com/2013/08/deploying-netflix-api.html

Example Distributed System: Minified



No Man (Microservice) is an Island

> *It's excellent to be able to implement a microservice really easily (Spring Boot), but building a system that way surfaces "non-functional" requirements that you otherwise didn't have.*

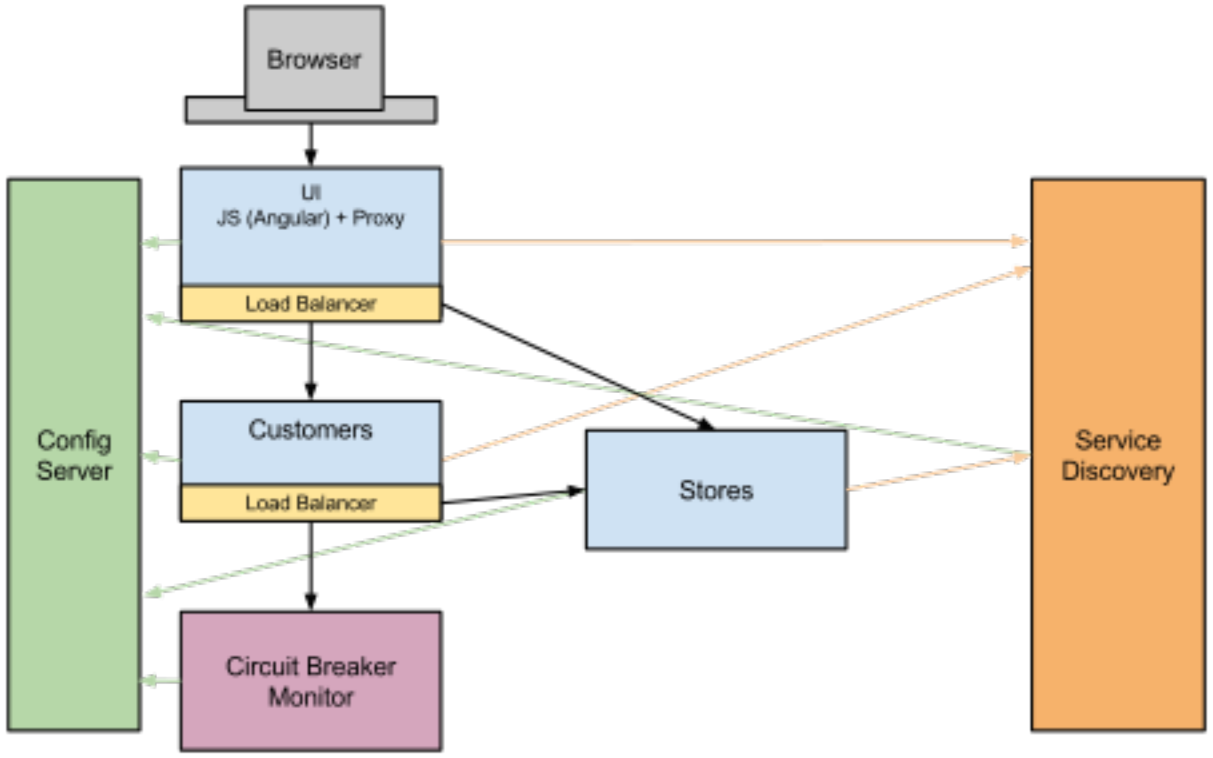
There are laws of physics that make some problems unsolvable (consistency, latency), but brittleness and manageability can be addressed with *generic*, *boiler plate* patterns.

Emergent features of micro-services systems

Coordination of distributed systems leads to boiler plate patterns

- Distributed/versioned configuration
- Service registration and discovery
- Routing
- Service-to-service calls
- Load balancing
- Circuit Breaker
- Asynchronous
- Distributed messaging

Example: Coordination Boiler Plate



Bootification

How to bring the ease of Spring Boot to a micro-services architecture?

- Netflix OSS
- Consul
- etcd
- zookeeper
- custom
- doozerd
- ha proxy
- nginx
- Typesafe Config

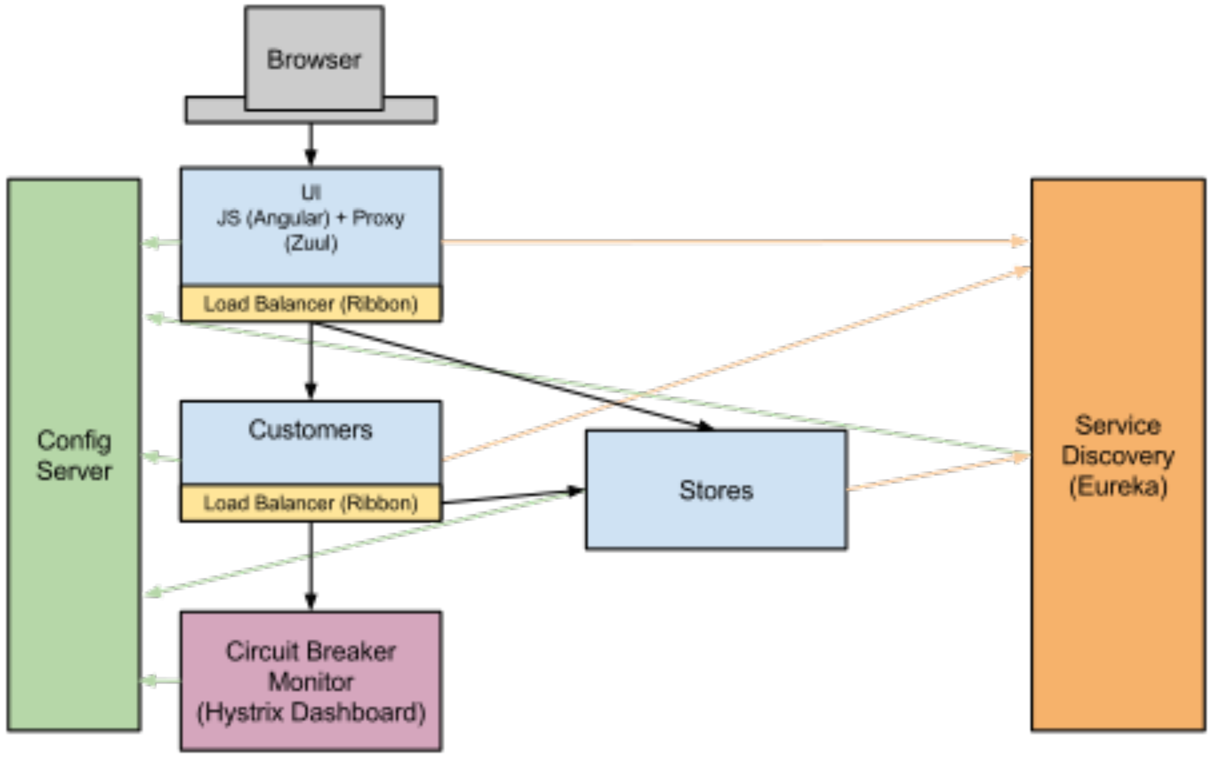
and many more... what to choose?

Netflix OSS

- Eureka
- Hystrix & Turbine
- Ribbon
- Feign
- Zuul
- Archaius
- Curator
- Asgaard
- ...

Mikey Cohen Netflix edge architecture, http://goo.gl/M159zi

Example: Spring Cloud and Netflix



Configuration Server

- Pluggable source
- Git implementation
- Versioned
- Rollback-able
- Configuration client auto-configured via starter

Spring Cloud Configuration Server

- Supports applications `<appname>.properties`
- Supports environments `<appname>-<envname>.yml`
- Default environment `application.properties` applies to all applications and environments

DEMO

Config Client

Consumers of config server can use client library as Spring Boot plugin

Features:

- Bootstrap `Environment` from server
- POST to /env to change `Environment`
- `@RefreshScope` for atomic changes to beans via Spring lifecycle
- POST to /refresh
- POST to /restart

Environment Endpoint

- POST to /env
- Re-binds `@ConfigurationProperties`
- Resets loggers if any logging.level changes are detected
- Sends `EnvironmentChangeEvent` with list of properties that changed

Refresh Endpoint

- POST to /refresh
- Re-loads configuration including remote config server
- Re-binds @ConfigurationProperties
- Resets @RefreshScope cache

RefreshScope

- Annotate @Beans
- Atomic updates during /refresh

DEMO

```
@EnableConfigurationProperties(MyProps)
public class Application {

  @Autowired
  private MyProps props

  @RefreshScope
  @Bean
  public Service service() {
    new Service(props.name)
  }
}
```

Restart Endpoint

- POST to `/restart` closes application context and refreshes it
- Probably more useful in development than production (leaks?)
- Disabled by default

Encrypted Properties

- Authenticated clients have access to unencrypted data.
- Only encrypted data is stored in git.
- Support for server side or client side decryption

DEMO

Discovery: Eureka

- Service Registration Server
- Highly Available
- In AWS terms, multi Availability Zone and Region aware

Eureka Client

- Register service instances with Eureka Server
- `@EnableEurekaClient` auto registers instance in server
- Eureka Server
- Eureka Client

```
@EnableEurekaClient
public class Application {
}
```

DEMO

Circuit Breaker: Hystrix

- latency and fault tolerance
- isolates access to other services
- stops cascading failures
- enables resilience
- circuit breaker pattern
- dashboard

Release It!: https://pragprog.com/book/mnee/release-it

Declarative Hystrix

- Programmatic access is cumbersome
- `@HystrixCommand` to the rescue
- `@EnableHystrix` via starter pom
- Wires up spring aop aspect

DEMO

Hystrix Synchronous

```
private String getDefaultMessage() {
  return "Hello World Default";
}

@HystrixCommand(fallbackMethod="getDefaultMessage")
public String getMessage() {
  return restTemplate.getForObject(/*...*/);
}
```

Hystrix Future

```
@HystrixCommand(fallbackMethod="getDefaultMessage")
public Future<String> getMessageFuture() {
  return new AsyncResult<String>() {
    public String invoke() {
      return restTemplate.getForObject(/*...*/);
    }
  };
}

//somewhere else
service.getMessageFuture().get();
```

Hystrix Observable

```
@HystrixCommand(fallbackMethod="getDefaultMessage")
public Observable<String> getMessageRx() {
  return new ObservableResult<String>() {
    public String invoke() {
      return restTemplate.getForObject(/*...*/);
    }
  };
}

//somewhere else
helloService.getMessageRx().subscribe(new Observer<String>() {
    @Override public void onCompleted() {}
    @Override public void onError(Throwable e) {}
    @Override public void onNext(String s) {}
});
```

Circuit Breaker Metrics

- Via actuator `/metrics`
- Server side event stream `/hystrix.stream`
- Dashboard app via `@EnableHystrixDashboard`
- More coming...

DEMO

Metric Aggregation: Turbine

- Aggregator for Hystrix data
- Pluggable locator
- Static list
- Eureka

Ribbon

- Client side load balancer
- Pluggable transport
- Protocols: http, tcp, udp
- Pluggable load balancing algorithms
- Round robin, “best available”, random, response time based
- Pluggable source for server list
- Static list, Eureka!

Feign

- Declarative web service client definition
- Annotate an interface
- Highly customizable
- Encoders/decoders
- Annotation processors (Feign, JAX-RS)
- Logging
- Supports Ribbon and therefore Eureka

Feign cont.

- Auto-configuration
- Support for Spring MVC annotations
- Uses Spring MessageConverter’s for decoder/encoder

DEMO

Feign cont.

```
public interface HelloClient {
  @RequestMapping(method = RequestMethod.GET,
                  value = "/hello")
  Message hello();

  @RequestMapping(method = RequestMethod.POST,
                  value = "/hello",
                  consumes = "application/json")
  Message hello(Message message);
}
```

Routing: Zuul

- JVM based router and filter
- Similar routing role as httpd, nginx, or CF go router
- Fully programmable rules and filters
- Groovy
- Java
- any JVM language

How Netflix uses Zuul

- Authentication
- Insights
- Stress Testing
- Canary Testing
- Dynamic Routing
- Service Migration
- Load Shedding
- Security
- Static Response handling
- Active/Active traffic management

Spring Cloud Zuul Proxy

- Store routing rules in config server
  `zuul.proxy.route.customers: /customers`
- uses `Hystrix->Ribbon->Eureka` to forward requests to appropriate service

```
@EnableZuulProxy
@Controller
class Application {
  @RequestMapping("/")
  String home() {
    return 'redirect:/index.html#/customers'
  }
}
```

DEMO

Configuration: Archaius

- Client side configuration library
- extends apache commons config
- extendible sources
- Polling or push updates

```
DynamicStringProperty myprop = DynamicPropertyFactory.getInstance()
      .getStringProperty("my.prop");
someMethod(myprop.get());
```

Archaius: Spring Environment Bridge

- Auto-configured
- Allows Archaius `Dynamic*Properties` to find values via Spring `Environment`
- Existing Netflix libraries configured via `application.{properties,yml}` and/or Spring Cloud Config Server

Spring Cloud Bus

- Lightweight messaging bus using spring integration abstractions
  - spring-amqp, rabbitmq and http
  - other implementations possible
- Send messages to all services or...
- To just one applications nodes (ie just service x) `?destination=x`
- Post to `/bus/env` sends environment updates
- Post to `/bus/refresh` sends a refresh command

DEMO

Spring Cloud Starters

spring-cloud-starter	spring-cloud-starter-hystrix
spring-cloud-starter-bus-amqp	spring-cloud-starter-hystrix-dashboard
spring-cloud-starter-cloudfoundry	spring-cloud-starter-turbine
spring-cloud-starter-eureka	spring-cloud-starter-zuul
spring-cloud-starter-eureka-server	

Links

- http://github.com/spring-cloud
- http://github.com/spring-cloud-samples
- http://blog.spring.io
- http://presos.dsyer.com/decks/cloud-boot-netflix.html
- Twitter: @spencerbgibb, @david_syer
- Email: sgibb@pivotal.io, dsyer@pivotal.io