



Spring blog

[All Posts](#)[Engineering](#)[Releases](#)[News and Events](#)

Microservice Registration and Discovery with Spring Cloud and Netflix's Eureka

ENGINEERING | JOSH LONG | JANUARY 20, 2015 | 44 COMMENTS

The microservice style of architecture is not so much about building individual services so much as it is making the *interactions between* services reliable and failure-tolerant. While the focus on these interactions is new, the need for that focus is not. We've long known that services don't operate in a vacuum. Even before cloud economics, we knew that - in a practical world - clients should be designed to be immune to service outages. The cloud makes it easy to think of capacity as ephemeral, fluid. The burden is on the client to manage this intrinsic complexity.

In this post, we'll look at how [Spring Cloud](#) helps you manage that complexity with a service registry like Eureka and Consul and client-side load-balancing.

The Cloud's Phone Book

A service registry is a phone book for your microservices. Each service registers itself with the service registry and tells the registry where it lives (host, port, node name) and perhaps other service-specific metadata - things that other services can use to make informed decisions about it. Clients can ask questions about the service topology ("are there any 'fulfillment-services' available, and if so,

Get the Spring newsletter

☐ Yes, I would like to be contacted by The Spring and VMware for newsletters, promotions and events.

SUBSCRIBE



(Cassandra, Memcached, etc.), and that information is ideally stored in a service registry.

There [are several popular options for service registries](#). Netflix built and then open-sourced their own service registry, [Eureka](#). Another new, but increasingly popular option is [Consul](#). We'll look principally at some of the integration between Spring Cloud and Netflix's Eureka service registry.

From the [the Spring Cloud project page](#): "Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry."

Spring Cloud already supports both Eureka and Consul, though I'll focus on Eureka in this post because it can be bootstrapped automatically in one of Spring Cloud's auto-configurations. Eureka is implemented on the JVM but Consul is implemented in Go.

Installing Eureka

Standing up an instance of the Eureka service registry is easy if you have `org.springframework.boot:spring-cloud-starter-eureka-server` on your classpath.

```
package registry;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
```

[COPY](#)



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

My nominal `src/main/resources/application.yml` looks like this these days.

```
server:  
  port: ${PORT:8761}  
  
eureka:  
  client:  
    registerWithEureka: false  
    fetchRegistry: false  
  server:  
    waitTimeInMsWhenSyncEmpty: 0
```

[COPY](#)

The service's port is defaulted to the well-known 8761 if [Cloud Foundry's](#) `VCAP_APPLICATION_PORT` environment variable isn't available. The rest of the configuration simply tells this instance to not register itself with the Eureka instance it finds, because that instance is.. itself. If you run it locally, you can point a browser to `http://localhost:8761` and monitor the registry from there.

Deploying Eureka

Spring Cloud will startup a [Eureka instance with its Spring Boot auto-configuration](#). There are a couple of things to consider when deploying Eureka. First, you should *always* use a highly-available configuration in production. [The Spring Cloud Eureka sample](#) shows how to deploy it in a highly-available configuration.

Clients need to know where to find the Eureka instance. If you have DNS then that might be one option, if you're not polluting too large a global namespace. If you're [running in a Platform-as-a-Service and](#)



exposed as environment variables. You can get the effect of having a Eureka service right now, though, by using Cloud Foundry's `cf` CLI to [create a user-provided service](#).

```
cf cups eureka-service -p '{"uri":"http://host-of-your-eureka-set'
```

Point `host-of-your-eureka-setup` to a well-known host for your highly-available Eureka setup. I suspect we'll soon see a way to create Eureka as a backing service in the same way you might a PostgreSQL or Elasticsearch instance on [Pivotal Cloud Foundry](#).

Now that Eureka is up and running, let's use it to connect some services to each other!

Speak for Yourself

Spring Cloud-based services have a `spring.application.name` property. It's used to pull down configuration from the Configuration server, to identify the service to Eureka, and is referenceable in numerous other contexts when building Spring Cloud-based applications. This value typically lives in `src/main/resources/bootstrap.(yaml,properties)`, which is picked up earlier in the initialization than the normal `src/main/resources/application.(yaml,properties)`. A service with `org.springframework.cloud:spring-cloud-starter-eureka` on the classpath will be registered with the Eureka registry by its `spring.application.name`.

The `src/main/resources/bootstrap.yaml` file for each of my services looks like this, where `my-service` is the service name that changes from service to service:

```
spring:
  application:
    name: my-service
```



`spring.application.name`, host, port, etc. You might wonder about that first bit. Spring Cloud attempts to look for it at a well-known address (`http://127.0.0.1:`), but you can change that. Here's my `src/main/resources/application.yml` for a nominal Spring Cloud microservice, though [there's no reason this couldn't live in the Spring Cloud configuration server](#). There may be many instances identifying themselves as `my-service`; Eureka will append the process' information to a list of registrations for the same ID.

COPY

```
eureka:
  client:
    serviceUrl:
      defaultZone: ${vcap.services.eureka-service.credentials.uri}

---
spring:
  profiles: cloud
eureka:
  instance:
    hostname: ${APPLICATION_DOMAIN}
    nonSecurePort: 80
```

In this configuration, the Spring Cloud Eureka client knows to connect to the Eureka instance running on localhost *if* Cloud Foundry's `VCAP_SERVICES` environment variable doesn't exist or contain valid credentials.

The bit of configuration under the `---` delimiter is for when the application is run under the `cloud` Spring profile. It's easy to set a profile using the `SPRING_PROFILES_ACTIVE` environment variable. You can configure Cloud Foundry environment variables in your `manifest.yml` or, [on Cloud Foundry Lattice](#), your [Docker file](#).

The `cloud` profile specific configuration specifically tells the Eureka client how to register the service in the discovered Eureka registry. I do this because my services don't use fixed DNS. `APPLICATION_DOMAIN` is



Click refresh on the Eureka web UI after 30 seconds (as of this writing) and you'll see your web service(s) registered.

Client-Side Load Balancing with Ribbon

Spring Cloud references other services through their `spring.application.name` value. Knowing this value can be handy in a lot of contexts when building Spring Cloud-based services.

The goal, you'll recall, is to let the *client* decide based on contextual information (which could change from client to client) which service instance it will connect to. Netflix has a Eureka-aware client-side load-balancing client called **Ribbon** that Spring Cloud integrates extensively. Ribbon is a client library with built-in software load balancers. Let's look at an example that uses Eureka directly and then uses it through the Ribbon and Spring Cloud integration.

```
package passport;

import org.apache.commons.lang.builder.ToStringBuilder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.cloud.netflix.feign.FeignClient;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.client.RestTemplate;

import java.util.List;

@SpringBootApplication
```

[COPY](#)



```

public class Application {

    public static void main(String[] args) {
        new SpringApplicationBuilder(Application.class)
            .web(false)
            .run(args);
    }
}

@Component
class DiscoveryClientExample implements CommandLineRunner {

    @Autowired
    private DiscoveryClient discoveryClient;

    @Override
    public void run(String... strings) throws Exception {
        discoveryClient.getInstances("photo-service").forEach((ServiceInstance) -> {
            System.out.println(ToStringBuilder.reflectionToString(instance));
        });
        discoveryClient.getInstances("bookmark-service").forEach((ServiceInstance) -> {
            System.out.println(ToStringBuilder.reflectionToString(instance));
        });
    }
}

@Component
class RestTemplateExample implements CommandLineRunner {

    @Autowired
    private RestTemplate restTemplate;

    @Override
    public void run(String... strings) throws Exception {
        // use the "smart" Eureka-aware RestTemplate
        ResponseEntity<List<Bookmark>> exchange =
            this.restTemplate.exchange(
                "http://bookmark-service/{userId}/bookmarks",
                HttpMethod.GET,
                null,
                new ParameterizedTypeReference<List<Bookmark>>() {},
                (Object) "mstine");

        exchange.getBody().forEach(System.out::println);
    }
}

```



```

class FeignExample implements CommandLineRunner {

    @Autowired
    private BookmarkClient bookmarkClient;

    @Override
    public void run(String... strings) throws Exception {
        this.bookmarkClient.getBookmarks("jlong").forEach(System.out::println);
    }

    @FeignClient("bookmark-service")
    interface BookmarkClient {

        @RequestMapping(method = RequestMethod.GET, value = "/{userId}")
        List<Bookmark> getBookmarks(@PathVariable("userId") String userId);
    }

    class Bookmark {
        private Long id;
        private String href, label, description, userId;

        @Override
        public String toString() {
            return "Bookmark{" +
                "id=" + id +
                ", href='" + href + '\'' +
                ", label='" + label + '\'' +
                ", description='" + description + '\'' +
                ", userId='" + userId + '\'' +
                '}';
        }

        public Bookmark() {}

        public Long getId() {
            return id;
        }

        public String getHref() {
            return href;
        }

        public String getLabel() {
            return label;
        }
    }
}

```




```
    }  
  
    public String getUserId() {  
        return userId;  
    }  
}
```

The `DiscoveryClientExample` bean demonstrates using the Spring Cloud common `DiscoveryClient` to interrogate the services. The results contain information like the hostname and the port for each service.

The `RestTemplateExample` bean demonstrates the auto-configured Ribbon-aware `RestTemplate` instance. Note that the URI uses a service ID, not an actual hostname. The service ID from the URI is extracted and given to Ribbon which then uses a load-balancer to pick from among the registered instances in Eureka and, finally, the HTTP call is made to a real service instance.

The `FeignExample` bean demonstrates using the Spring Cloud Feign integration. `Feign` is a handy project from Netflix that lets you describe a REST API client declaratively with annotations on an interface. In this case, we want to map the HTTP results from calls to the `bookmark-service` to the `BookmarkClient` Java interface. This mapping is configured in the `Application` class towards the top of the code page:

```
@Bean  
BookmarkClient bookmarkClient() {  
    return loadBalance(BookmarkClient.class, "http://bookmark-ser  
}
```

[COPY](#)

The URI is a service reference, not an actual hostname. It's passed through the same processing as the URI given to the `RestTemplate` in the last example.

Pretty cool, eh? You can use the more basic `DiscoveryClient` API and make a call, or use the Ribbon and Eureka-aware `RestTemplate` or



Review

- Spring Cloud supports both the Eureka and Consul service registries (and perhaps more!)
- The `DiscoveryClient` API can be used to interactively query Eureka given a service ID.
- Ribbon is a client-side load balancer
- The `RestTemplate` can substitute service IDs for hostnames in URIs and can defer to Ribbon to pick a service.
- The Netflix Spring Cloud Feign integration makes it simple to create smart, Eureka-aware REST clients that uses Ribbon for client-side load-balancing to pick an available service instance.

Where to go from Here

We've only looked at service discovery and resolution with Eureka. Most of what we talked about here applies to Consul as well and indeed Consul has some features that Netflix doesn't have.

Round-robin load-balancing is just one option. You might instead require some notion of a leader node, and leadership election. Spring Cloud aims provides support for that kind of coordination, as well.

Service registration and client-side load-balancing are just *one* of the things that Spring Cloud does to promote more resilient service-to-service calls. We have *not* looked at its support single-sign on and security, distributed locks and leadership election, reliability patterns like the circuit breaker, and much more.

The example code is all available online so don't hesitate to the check out the example on your local machine or push it to Cloud Foundry using the provided `cf.sh` script and various `manifest.yml` files.





G

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

11

Share

Best Newest Oldest**charyorde**

8 years ago

Is DiscoveryClient asynchronous and long-lived? For example, service A want to be able to get an update from service B without making a new request.

4 1 Reply • Share ›

K

Kamil

7 years ago edited

When pushing a client application to CF getting the following error:
ERR Caused by: java.lang.IllegalArgumentException: Could not resolve placeholder 'APPLICATION_DOMAIN' in string value
"\${APPLICATION_DOMAIN}"

1 1 Reply • Share ›

**Arun M R Nair**

7 years ago

The above example works fine. and i changed the main application into a war. war deployed successfully and my service listed in Eureka's dashboard. But the problem is, i cant able to access the API by calling its name that registered in eureka. Its getting a 404 error. At the same time we can access the API as normal way, by specifying ip and port. Is there any additional steps needed?

1 1 Reply • Share ›

S

Sai Krishna

➔ Arun M R Nair

7 years ago edited

Can u send me the war file or how can we create a war file

0 0 Reply • Share ›



Registration or Napster Peer to peer? Can Spring Eureka (or xyz software) Microservice does the same?

of the Apache Software Foundation in the United States and/or other countries. Java™, Java™ SE, Java™ EE, and OpenJDK™ are trademarks of Oracle and/or its affiliates. Kubernetes® is a registered trademark of the Linux Foundation in the United States and other countries. Linux® is the registered trademark of Linus Torvalds in the United States and other countries.



Captain Heavy

6 years ago

I had a particularly nasty time getting this type of microservice architecture working using the Feign Client pattern. My problems and solution can be found here: <https://stackoverflow.com/q...>

0 0 Reply • Share ›

R

Richard Valdivieso

7 years ago

I am trying to implement eureka in cloud foundry, but without look. Can eureka oss be implemented on CF?

0 0 Reply • Share ›



Pieter_H

→ Richard Valdivieso

7 years ago

Yep! Eureka has been fully integrated into PCF. Check this out: <https://blog.pivotal.io/piv...>

0 0 Reply • Share ›

R

Richard Valdivieso

→ Pieter_H

7 years ago

But that is a pay service, right? I meant to create a simple web app with the eureka annotations and push it to cloud foundry

0 0 Reply • Share ›



William Witt

→ Richard Valdivieso

7 years ago

I'm putting together a tutorial using Josh Long's Cloud Native talk but on PCFDev (It will include using Eureka). The actual tutorial is a few days out, but my initial notes are here:

<https://unamanic.blogspot.c...>

0 0 Reply • Share ›



John

7 years ago

How would the application vml look for Eureka if you were to set it up

How would the application.yml look for Eureka if you were to set it up for peer to peer for Cloud Foundry

0 0 Reply • Share ›

K

kk_spring

7 years ago

Hi -

Need help on ribbon load balancing with Eureka. I would like to use rest template as below.

could someone assist?

Here is my client
 SpringBootApplication
 @EnableAutoConfiguration(exclude = {
 DataSourceAutoConfiguration.class })
 @ComponentScan
 @EnableDiscoveryClient
 public class cachingClient {
 public static void main(String[] args) {
[SpringApplication.run](#)(cachingClient.class, args);
 }
 }

[see more](#)

0 0 Reply • Share ›



Hameister

7 years ago edited

I am not sure, but I think the auto configuration of the RestTemplate in the class RestTemplateExample, does not work since March 2015...:
<https://github.com/spring-c...>

0 0 Reply • Share ›



psycience

7 years ago

I'm getting a refused connection between my eureka client and server
<http://stackoverflow.com/qu...>

0 0 Reply • Share ›

L

lipinggm

7 years ago

We would like to implement HTTPS for our Eureka-Enabled Client by adding following properties inside application property file:
 server.port: 8020
 eureka.instance.nonSecurePort: 8020
 eureka.instance.nonSecurePortEnabled: false

```
eureka.instance.nonSecurePortEnabled: false
eureka.instance.securePortEnabled: true
eureka.instance.securePort: 8021
eureka.instance.preferIpAddress: true
```

But when we check the service info link: only <http://xxxx:8020/info> works, neither secure port link <http://xxxx:8021> or <https://xxxx:8021> worked. I am wondering how Eureka handles https vs http. Any examples available?
Any help is highly appreciated.

0 0 Reply • Share ›



Josh Long

→ lipinggm



7 years ago

Hi, the amazing Dr. Syer pointed us to

<http://projects.spring.io/s...>

Which explains how to register secure applications

0 0 Reply • Share ›

L

lipinggm



7 years ago

On Eureka Dashboard, under section "Instances currently registered with Eureka", besides 'Application', 'Status', ... Is there anyway that Eureka dashboard could display other information such as if application instance is SSL enabled or not.

0 0 Reply • Share ›



Josh Long

→ lipinggm



7 years ago

pull requests are welcome ;-) we're just using the Netflix Eureka .jsp - make changes to that to show whatever you want.

0 0 Reply • Share ›

L

lipinggm

→ Josh Long



7 years ago

Thank you so much Josh.

We want to implement HTTPS for our Eureka-Enabled Service by adding following properties to the Eureka Client's properties file:

```
server.port
```

```
eureka.instance.nonSecurePort: 8020
```

```
eureka.instance.nonSecurePortEnabled: false
```

```
eureka.instance.securePortEnabled: true
```

```
eureka.instance.securePort: 8021
```

```
eureka.instance.preferIpAddress: true
```


But when we check the service info link: only
http://xxx:8020/info works, neither secure port
link http://xxx:8021 or https://xxx:8021 worked. I
am wondering how Eureka handles https vs http.
Any examples available?
Any help is highly appreciated.
Liping

0 0 Reply • Share ›

D**disqus_05J0W3UW0X**

8 years ago

my HTTPS enabled microapp/service is getting registered in Eureka with HTTP based URL. I tried adding the
eureka.instance.securePortEnabled=true and
eureka.instance.secureVirtualHostName property to my service's
yaml file but still no luck. How do we register secure/ssl enabled
service (Spring Boot) in Eureka ?

0 0 Reply • Share ›

**ractive**

8 years ago

Would be nice if this blog post could be updated using the newest
spring-cloud versions.

0 0 Reply • Share ›

M**Marcin Jarzab**

8 years ago

I tried to play little bit with the sample but passport/[Application.java](#) is
not valid because FeignConfiguration class is not present in the
classpath. I tried to examine the source code
and this class is not available:
<https://github.com/spring-c...>

symbol: class FeignConfiguration
[ERROR] location: package org.springframework.cloud.netflix.feign
[ERROR] /var/share/test-workspace/service-registration-and-
discovery/passport-service/src/main/java/passport/[Application.java](#)

0 0 Reply • Share ›

**Josh Long**

➔ Marcin Jarzab

8 years ago

Hi, I've updated the code in GitHub and in the example. git
pull and mvn clean install accordingly.

0 0 Reply • Share ›

A**Andrei Pop**



8 years ago

Is it possible to do the same client side load balancing with the OAuth2RestTemplate? Looking at the sample you provided with the RestTemplate, is what I need to do but need oauth support. Thanks

0 0 Reply • Share ›

**Dave Syer**

➔ Andrei Pop



8 years ago edited

Yes it is possible. You need to use the `RibbonInterceptor` in your `RestTemplate`. It's created for you, so you can just autowire it. (Code: [https://github.com/spring-c...](https://github.com/spring-cloud/spring-cloud-netflix)

0 0 Reply • Share ›

**Pascal Gehl**

8 years ago

"Spring Cloud already supports [...] Consul" I was enable to find Consul related documentation in <http://projects.spring.io/s...>

Do you plan to add a chapter on it ?

0 0 Reply • Share ›

**Spencer Gibb**

➔ Pascal Gehl



8 years ago

[https://github.com/spring-c...](https://github.com/spring-cloud/spring-cloud-netflix) is currently a developer preview. Official documentation will be added later.

0 0 Reply • Share ›

**Georgios Andrianakis**

➔ Spencer Gibb

8 years ago edited

Looks good!

I am sort of confused however on where the placement of Spring Cloud Consul will be inside the Spring Cloud ecosystem.

Will one be able to substitute Eureka with Consul and have everything else working as is, or does the use of Spring Cloud Consul mean that one would have to abandon Spring Cloud Netflix?

Thanks!

0 0 Reply • Share ›

**Spencer Gibb**

➔ Georgios Andrianakis

8 years ago

It has support for netflix things like zuul and ribbon, but it replaces eureka for

discovery and git for config. I'm looking at it like Spring Data and the different databases that Spring Data supports

1 0 Reply • Share ›



Georgios Andrianakis

→ Spencer Gibb

8 years ago

Most impressive!

That's exactly what I was hoping for, having a Service Registry implementation being interchangeable while all the other moving parts remain intact!

Great work! Thanks for the project and the prompt reply!

0 0 Reply • Share ›



Pascal Gehl

→ Spencer Gibb

8 years ago

Thanks I'll take a look.

0 0 Reply • Share ›

S

Shree Prakash

6 years ago

I have written a micro-service with spring boot and registered in eureka server . I want to make service discovery in node.js for discovering same registered service. How do I proceed ?

0 1 Reply • Share ›



Hitesh Bargujar

7 years ago

if you are using [spring.application.name](#) as the name to be registered with Eureka, then do not use eureka.instance.appName or if at all using make sure both of them are exactly the same. Though the registration goes good, discovery using `org.springframework.cloud.client.discovery.DiscoveryClient` fails `getInstances(service)` return empty