Spring Cloud Netflix: Service Discovery

Spring Cloud Netflix: Service Discovery

- Review Service Discovery
- Eureka Server
- Eureka Client
- Spring Cloud Services

Challenge

- Service Discovery is one of the key tenets of a microservice based architecture.
- In distributed systems, application dependencies cease to be a method call away.
- Trying to hand configure each client or use some form of convention can be very difficult to do and can be very brittle.

Where We Have Been

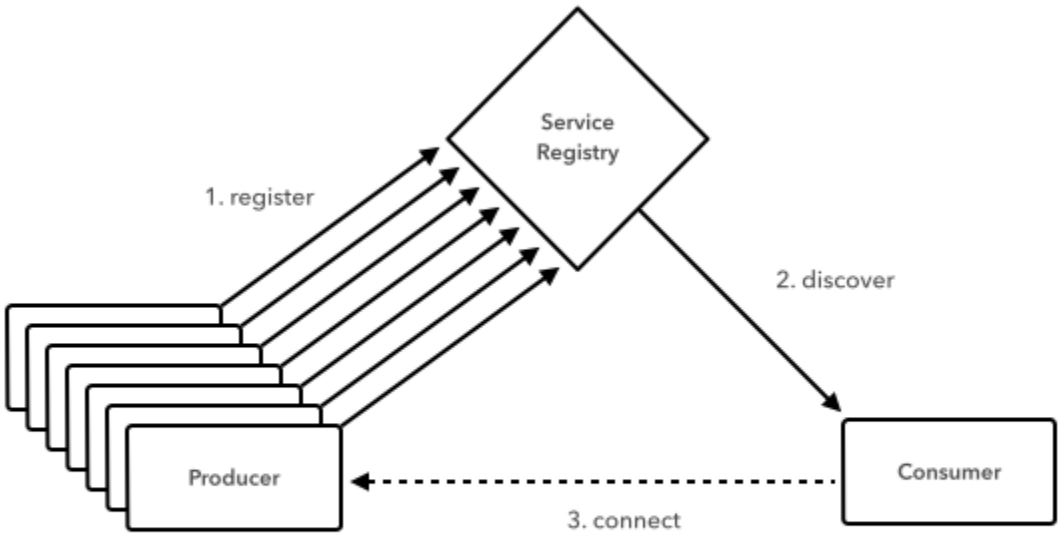
How have we discovered services in the past?

- Service Locators
- Dependency Injection
- Service Registries

Spring Cloud Netflix: Service Discovery

- Review Service Discovery
- Eureka Server
- Eureka Client
- Spring Cloud Services

Service Discovery with Spring Cloud



Include Dependency

pom.xml

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka-server</artifactId>
</dependency>
```

Eureka Server

An example Eureka server. **`@EnableEurekaServer`** annotation.

```
@SpringBootApplication
@EnableEurekaServer
public class ServiceRegistryApplication {

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

Eureka Resiliency

- Eureka does not have a backend data store. All data is kept in memory.
- Service instances in the registry all have to send heartbeats to keep their registrations up to date
- Clients also have an in-memory cache of eureka registrations (so they don't have to go to the registry for every single request to a service)

Spring Cloud Netflix: Service Discovery

- Review Service Discovery
- Eureka Server
- Eureka Client
- Spring Cloud Services

Include Dependency

pom.xml

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

Client Application

An example client application. **`@EnableDiscoveryClient`** annotation.

```
@SpringBootApplication
@EnableDiscoveryClient
public class GreetingServiceApplication {


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

}
```

Client Config

Configuration required to locate the Eureka.

`application.yml`

```
spring:
  application:
    name: fortune-service
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

- **`defaultZone`** - the default **`serviceUrl`** to reach Eureka
- **`spring.application.name`** is the name the application will use to register its service

Registering with Eureka

- When a client registers with Eureka, it provides meta-data about itself such as host and port, health indicator URL, home page etc.

- Eureka receives heartbeat messages from each instance belonging to a service. If the heartbeat fails over a configurable timetable, the instance is normally removed from the registry.

Discovery Client

The **`DiscoveryClient`** is used to locate the stores service.

```
@Autowired
private DiscoveryClient discoveryClient;

public String serviceUrl() {
    InstanceInfo instance =
      discoveryClient.getNextServerFromEureka("STORES", false);
    return instance.getHomePageUrl();
}
```

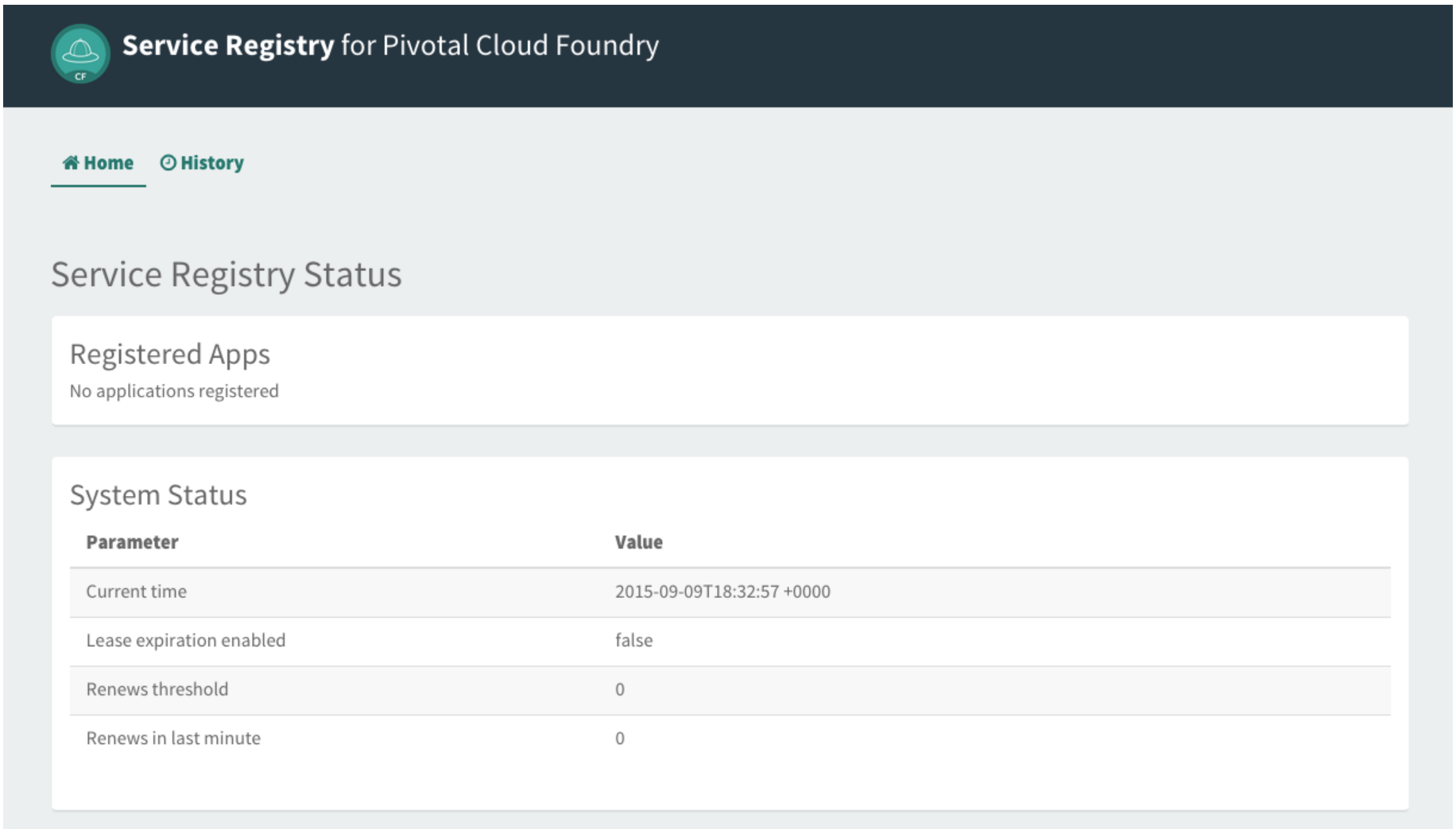
Spring Cloud Netflix: Service Discovery

- Review Service Discovery
- Eureka Server
- Eureka Client
- Spring Cloud Services

Spring Cloud Services

- Brings Spring Cloud to Pivotal Cloud Foundry
- Includes: Config Server, Service Registry & Circuit Breaker Dashboard services

Spring Cloud Services: Service Registry



Spring Cloud Services: Service Registry

1) Add dependency

```
<dependency>
  <groupId>io.pivotal.spring.cloud</groupId>
  <artifactId>spring-cloud-services-starter-service-registry</artifactId>
</dependency>
```

2) Create a Service Registry service instance

3) Bind the service to the app