Pattern: Self Registration

Context

You have applied either the Client-side Service Discovery pattern (client-side-discovery.html) or the Server-side Service Discovery pattern (server-side-discovery.html). Service instances must be registered with the service registry (service-registry.html) on startup so that they can be discovered and unregistered on shutdown.

Problem

How are service instances registered with and unregistered from the service registry?

Forces

- Service instances must be registered with the service registry on startup and unregistered on shutdown
- Service instances that crash must be unregistered from the service registry
- Service instances that are running but incapable of handling requests must be unregistered from the service registry

Solution

A service instance is responsible for registering itself with the service registry. On startup the service instance registers itself (host and IP address) with the service registry and makes itself available for discovery. The client must typically periodically renew its registration so that the registry knows it is still alive. On shutdown, the service instance unregisters itself from the service registry.

This is typically handled by a Microservice chassis framework (microservice-chassis.html)

Examples

The Microservices Example application (https://github.com/cer/microservices-examples) is an example of an application that uses self-registration. It is written in Scala and uses Spring Boot and Spring Cloud as the Microservice chassis (microservice-chassis.html). The application uses the Eureka (https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance) Service Registry (service-registry.html), which is a Netflix OSS (http://netflix.github.io/) component.

Service registration is configured using the `@EnableEurekaClient` on a Java config class:

```
@Configuration
@EnableEurekaClient
class EurekaClientConfiguration {
```

This annotation causes the service instance to be registered with Eureka.

Resulting context

The benefits of the Self Registration pattern include the following:

- A service instance knows its own state so can implement a state model that’s more complex than UP/DOWN, e.g. STARTING, AVAILABLE, …

There are also some drawbacks:

- Couples the service to the Service Registry
- You must implement service registration logic in each programming language/framework that you use to write your services, e.g. NodeJS/JavaScript, Java/Scala, etc.
- A service instance that is running yet unable to handle requests will often lack the self-awareness to unregister itself from the service registry

Related patterns

- Service Registry (service-registry.html) - an essential part of service discovery
- Client Side Discovery (client-side-discovery.html) - one way that a service instance is discovered
- Server Side Discovery (server-side-discovery.html) - another way a service instance is discovered
- Microservice chassis (microservice-chassis.html) - self registration is the responsibility the microservice chassis framework
- 3rd Party Registration (3rd-party-registration.html) is an alternative solution

Tweet

Follow @MicroSvcArch