Pattern: Service registry

Context

Clients of a service use either Client-side discovery (client-side-discovery.html) or Server-side discovery (server-side-discovery.html) to determine the location of a service instance to which to send requests.

Problem

How do clients of a service (in the case of Client-side discovery (client-side-discovery.html)) and/or routers (in the case of Server-side discovery (server-side-discovery.html)) know about the available instances of a service?

Forces

- Each instance of a service exposes a remote API such as HTTP/REST, or Thrift etc. at a particular location (host and port)
- The number of services instances and their locations changes dynamically. Virtual machines and containers are usually assigned a dynamic IP address. An EC2 Autoscaling Group, for example, adjusts the number of instances based on load.

Solution

Implement a service registry, which is a database of services, their instances and their locations. Service instances are registered with the service registry on startup and deregistered on shutdown. Client of the service and/or routers query the service registry to find the available instances of a service. A service registry might invoke a service instance's health check API (observability/health-check-api.html) to verify that it is able to handle requests

Examples

The Microservices Example application (https://github.com/cer/microservices-examples) is an example of an application that uses client-side service discovery. It is written in Scala and uses Spring Boot and Spring Cloud as the Microservice chassis (microservice-chassis.html). They provide various capabilities including a Netflix Eureka (https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance) service registry.

The Eureka Server is a small Spring Boot application:

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServer {

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

}
```

It is deployed using Docker:

```
eureka:
  image: java:openjdk-8u91-jdk
  working_dir: /app
  volumes:
    - ./eureka-server/build/libs:/app
  command: java -jar /app/eureka-server.jar --server.port=8761
  ports:
    - "8761:8761"
```

Other examples of service registries (or technologies that are commonly used as service registries) include:

- Apache Zookeeper (http://zookeeper.apache.org)
- Consul (https://consul.io/)
- Etcd (https://github.com/coreos/etcd)

Some systems such as Kubernetes, Marathon and AWS ELB have an implicit service registry.

Resulting context

The benefits of the Service Registry pattern include:

- Client of the service and/or routers can discover the location of service instances.

There are also some drawbacks:

- Unless the service registry is built in to the infrastructure, it is yet another infrastructure component that must be setup, configured and managed. Moreover, the service registry is a critical system component. Although clients should cache data provided by the service registry, if the service registry fails that data will eventually become out of date. Consequently, the service registry must be highly available.

You need to decide how service instances are registered with the service registry. There are two options:

- Self registration pattern (self-registration.html) - service instances register themselves.
- 3rd party registration pattern (3rd-party-registration.html) - a 3rd party registers the service instances with the service registry.

The clients of the service registry need to know the location(s) of the service registry instances. Service registry instances must be deployed on fixed and well known IP addresses. Clients are configured with those IP addresses.

For example, Netflix Eureka (https://github.com/Netflix/eureka/wiki/Configuring-Eureka-in-AWS-Cloud) service instances are typically deployed using elastic IP addresses. The available pool of Elastic IP addresses is configured using either a properties file or via DNS. When a Eureka instance starts up it consults the configuration to determine which available Elastic IP address to use. A Eureka client is also configured with the pool of Elastic IP addresses.

Related patterns

- Client-side discovery (client-side-discovery.html) and Server-side discovery (server-side-discovery.html) create the need for a service registry
- Self registration (self-registration.html) and 3rd party registration (3rd-party-registration.html) are two different ways that service instances can be registered with the service registry
- Health Check API (observability/health-check-api.html) - the service registry invokes a service instance’s health check API to verify that it is able to handle requests