Pattern: Health Check API

Context

You have applied the Microservice architecture pattern (../microservices.html). Sometimes a service instance can be incapable of handling requests yet still be running. For example, it might have ran out of database connections. When this occurs, the monitoring system should generate a alert. Also, the load balancer or service registry (../service-registry.html) should not route requests to the failed service instance.

Problem

How to detect that a running service instance is unable to handle requests?

Forces

- An alert should be generated when a service instance fails
- Requests should be routed to working service instances

Solution

A service has an health check API endpoint (e.g. HTTP `/health`) that returns the health of the service. The API endpoint handler performs various checks, such as

- the status of the connections to the infrastructure services used by the service instance
- the status of the host, e.g. disk space
- application specific logic

A health check client - a monitoring service, service registry (../service-registry.html) or load balancer - periodically invokes the endpoint to check the health of the service instance.

Examples

The Microservices Example application (https://github.com/cer/microservices-examples) is an example on an application implements a health check API. It is written in Scala and uses Spring Boot and Spring Cloud as the Microservice chassis (/patterns/microservice-chassis.html). They provide various capabilities including a health check endpoint. The endpoint is implemented by the Spring Boot Actuator (http://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html) module. It configures a `/health` HTTP endpoint that invokes extensible health check logic.

To enable a `/health` endpoint, first define actuator as a dependency:

```
dependencies {
  compile "org.springframework.boot:spring-boot-starter-actuator"
```

Second, enable Spring Boot autoconfiguration:

```
@SpringBootApplication
class UserRegistrationConfiguration {
```

At this point, your application will have a health check endpoint with default behavior.

You can customize this behavior by defining one or more Spring beans that implement the `HealthIndicator` interface:

```
class UserRegistrationConfiguration {
  @Bean
  def discoveryHealthIndicator(discoveryClient : EurekaClient ) : HealthIndicator = new DiscoveryHealthIndica
tor(discoveryClient)
```

A `HealthIndicator` must implement a `health()` method, which returns a `Health` value.

Resulting Context

This pattern has the following benefits:

- The health check endpoint enables the health of a service instance to be periodically tested

This pattern has the following drawbacks:

- The health check might not sufficiently comprehensive or the service instance might fail between health checks and so requests might still be routed to a failed service instance

Related patterns

- Service registry (../service-registry.html) - the service registry invokes the health check endpoint