Pattern: Remote Procedure Invocation (RPI)

Context

You have applied the Microservice architecture pattern (../microservices.html). Services must handle requests from the application's clients. Furthermore, services must sometimes collaborate to handle those requests. They must use an inter-process communication protocol.

Forces

- Services often need to collaborate
- Synchronous communicate results in tight runtime coupling, both the client and service must be available for the duration of the request

Problem

How do services in a microservice architecture communicate?

Solution

Use RPI for inter-service communication. The client uses a request/reply-based protocol to make requests to a service.

Examples

There are numerous examples of RPI technologies

- REST (https://en.wikipedia.org/wiki/Representational_state_transfer)
- gRPC (http://www.grpc.io/)
- Apache Thrift (https://thrift.apache.org/)

`RegistrationServiceProxy` from the Microservices Example application (https://github.com/cer/microservices-examples) is an example of a component, which is written in Scala, that makes a REST request using the Spring Framework's `RestTemplate` :

```
@Component
class RegistrationServiceProxy @Autowired()(restTemplate: RestTemplate) extends RegistrationService {

  @Value("${user_registration_url}")
  var userRegistrationUrl: String = _

  @HystrixCommand(commandProperties=Array(new HystrixProperty(name="execution.isolation.thread.timeoutInMilli
seconds", value="800")))
  override def registerUser(emailAddress: String, password: String): Either[RegistrationError, String] = {
    try {
      val response = restTemplate.postForEntity(userRegistrationUrl,
        RegistrationBackendRequest(emailAddress, password),
        classOf[RegistrationBackendResponse])
      response.getStatusCode match {
        case HttpStatus.OK =>
          Right(response.getBody.id)
      }
    } catch {
      case e: HttpClientErrorException if e.getStatusCode == HttpStatus.CONFLICT =>
        Left(DuplicateRegistrationError)
    }
  }
}
```

The value of `user_registration_url` is supplied using Externalized configuration (../externalized-configuration.html).

Resulting context

This pattern has the following benefits:

- Simple and familiar
- Request/reply is easy
- Simpler system since there in no intermediate broker

This pattern has the following drawbacks:

- Usually only supports request/reply and not other interaction patterns such as notifications, request/async response, publish/subscribe, publish/async response
- Reduced availability since the client and the service must be available for the duration of the interaction

This pattern has the following issues:

- Client needs to discover locations of service instances

Related patterns

- The Domain-specific protocol (domain-specific.html) is an alternative pattern
- The Messaging (messaging.html) is an alternative pattern
- Externalized configuration (../externalized-configuration.html) supplies the (logical) network location, e.g. URL, of the service.
- A client must use either Client-side discovery (../client-side-discovery.html) and Server-side discovery (../server-side-discovery.html) to locate a service instance
- A client will typically use the Circuit Breaker pattern (../reliability/circuit-breaker.html) to improve reliability