微服务架构

简要介绍

概要

一、理念

二、技术

一、理念

What is microservice ?

microservices is a software architecture style in which complex applications are composed of small, independent processes communicating with each other using language-agnostic APIs.

These services are small, highlydecoupled and focus on doing a small task, facilitating a modular approach to system-building.

Properties of microservices architecture

- The services are easy to replace
- Services are organized around capabilities, e.g., user interface front-end, recommendation, logistics, billing, etc.
- Services can be implemented using different programming languages, databases, hardware and software environment, depending on what fits best
- Architectures are symmetrical rather than hierarchical (producer - consumer)

Principles of Microservices

- Model Around Business Concepts
- Adopt a Culture of Automation
- Hide Internal Implementation Details

Principles of Microservices (cont.)

- Decentralize All the Things
- Independently Deployable
- Isolate Failure
- Highly Observable

Characteristics of a Microservice Architecture

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes

Characteristics of a Microservice Architecture (cont.)

- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

二、技术

A high performance, open source, general RPC framework that puts mobile and HTTP/2 first.

a language-neutral, platform-neutral, remote procedure call (RPC) system initially developed at Google.

Powerful IDL

Define your service using Protocol Buffers, a powerful binary serialization toolset and language.

```
message HelloRequest {
  string greeting = 1;
}

message HelloResponse {
  string reply = 1;
}

service HelloService {
  rpc SayHello(HelloRequest) returns
  (HelloResponse);
}
```

Libraries in ten languages

Automatically generate idiomatic client and server stubs for your service in a variety of languages.

gRPC has libraries in:

C, C++, Java, Go, Node.js,Python, Ruby, Objective-C,PHP and C#.

HTTP/2

Building on the HTTP/2 standard brings many capabilities such as bidirectional streaming, flow control, header compression, multiplexing requests over a single TCP connection and more.

These features save battery life and data usage on mobile devices while speeding up services and web applications running in the cloud.

gRPC Motivation

Google has been using a single general-purpose RPC infrastructure called Stubby to connect the large number of microservices running within and across our data centers for over a decade. Our internal systems have long embraced the microservice architecture gaining popularity today. Having a uniform, cross-platform RPC infrastructure has allowed for the rollout of fleet-wide improvements in efficiency, security, reliability and behavioral analysis critical to supporting the incredible growth seen in that period.

Stubby has many great features - however, it's not based on any standard and is too tightly coupled to our internal infrastructure to be considered suitable for public release. With the advent of SPDY, HTTP/2, and QUIC, many of these same features have appeared in public standards, together with other features that Stubby does not provide. It became clear that it was time to rework Stubby to take advantage of this standardization, and to extend its applicability to mobile, IoT, and Cloud use-cases.

gRPC Design Principles

Payload Agnostic - Different services need to use different message types and encodings such as protocol buffers, JSON, XML, and Thrift; the protocol and implementations must allow for this. Similarly the need for payload compression varies by use-case and payload type: the protocol should allow for pluggable compression mechanisms.

Streaming - Storage systems rely on streaming and flow-control to express large data-sets. Other services, like voice-to-text or stock-tickers, rely on streaming to represent temporally related message sequences.

Blocking & Non-Blocking - Support both asynchronous and synchronous processing of the sequence of messages exchanged by a client and server. This is critical for scaling and handling streams on certain platforms.

gRPC Design Principles (cont.)

Cancellation & Timeout - Operations can be expensive and long-lived - cancellation allows servers to reclaim resources when clients are well-behaved. When a causal-chain of work is tracked, cancellation can cascade. A client may indicate a timeout for a call, which allows services to tune their behavior to the needs of the client.

Lameducking - Servers must be allowed to gracefully shut-down by rejecting new requests while continuing to process in-flight ones.

RPC life cycle

Unary RPC

Server streaming RPC

Client streaming RPC

Bidirectional streaming RPC

Deadlines

RPC termination

Hello gRPC how to

- Create a protocol buffers schema that defines a simple RPC service with a single Hello World method.
- Create a server that implements this interface in your favourite language (where available).
- Create a client in your favourite language (or any other one you like!) that accesses your server.

gRPC releases Beta, opening door for use in production environments

The gRPC team is excited to announce the immediate availability of gRPC Beta. This release marks an important point in API stability and going forward most API changes are expected to be additive in nature. This milestone opens the door for gRPC use in production environments.

Summary

一、理念－microservice

二、技术－gRPC

Reference

Articles :

https://en.wikipedia.org/wiki/Microservices

http://martinfowler.com/articles/microservices.html

http://www.grpc.io/

Books:

Building Microservices by Sam Newman 2015

Wen Zhenglin
wenzhenglin@bjjdsy.com.cn
2016-3-3

Thank you !