GRPC

Connector Alternativen zu JSON over HTTP

Was wir bis jetzt wissen

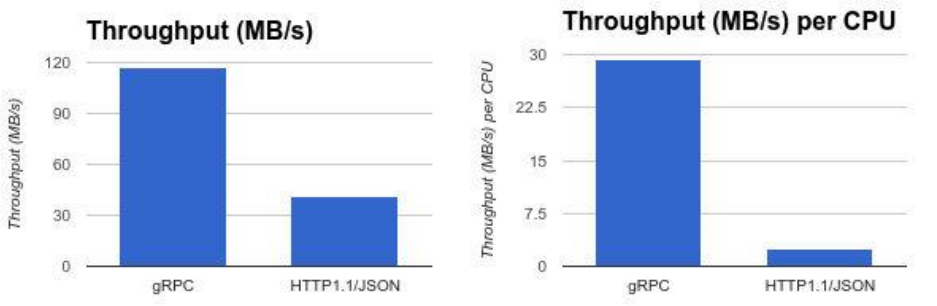
- JSON-over-HTTP mit dem REST Architecture Style
  - Wir denken ressourcen-orientiert

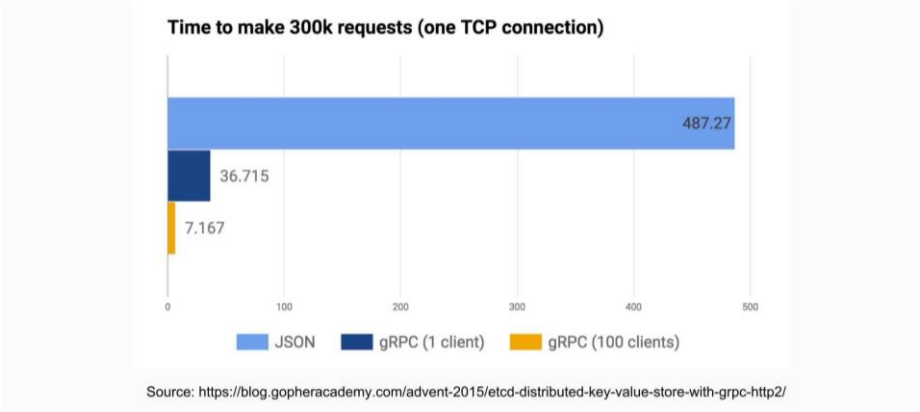
Was ist GRPC?

- gRPC is a modern open source **high performance RPC framework** that can run in any environment. It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication. It is also applicable in last mile of distributed computing to connect devices, mobile applications and browsers to backend services.

Performancevergleich zu HTTP 1.1

- gRPC nutzt HTTP2
- Auch Kleinvieh macht Mist – Easy-Win Optimierungen sind die Besten





Source: https://blog.gopheracademy.com/advent-2015/etcd-distributed-key-value-store-with-grpc-http2/

RPC

- Wikipedia:
- RPC ist eine Möglichkeit, ein Client-Server-Modell zu implementieren. Die Kommunikation beginnt, indem der Client eine Anfrage an einen bekannten Server schickt und **auf die Antwort wartet**. In der Anfrage gibt der Client an, welche Funktion mit welchen Parametern ausgeführt werden soll. Der Server bearbeitet die Anfrage und schickt die Antwort an den Client zurück. **Nach Empfang der Nachricht kann der Client seine Verarbeitung fortführen**.
- Beim Einsatz von RPC können durch **Kommunikationsfehler** unterschiedliche Fehlerkonstellationen auftreten, die beachtet und bearbeitet werden müssen.

Übung 3

- Die Solution enthält das Hello-World von GRPC
- Die wichtigsten Pakete
  - Google Protobuf
  - Grpc
  - Grpc.Tools
- Die wichtigste Datei: protodefitinion.proto (IDL)

Aufgabe 1

- Erstellen ein `OrderService`
- Erstellen sie eine Methode `CreateOrder`, welche den Parameter `CreateOrderRequest` hat und den Rückgabewert `CreateOrderResponse`
  - `CreateOrder` gibt den Inhalt der Bestellung auf der Console aus
- `CreateOrderRequest` besteht aus einer `UserId` (int32) und einem Array von `Items`
  - Ein `Item` besteht aus einer `Id` (int32) und einem `Price` (double)
- Der `CreateOrderResponse` besteht aus einer `Id` (int32) und gibt pro Bestellung immer die nächst höhere Zahl aus
- https://grpc.io/docs/quickstart/csharp/

Aufgabe 1.1

- Machen Sie zwei exemplarische Aufrufe von `CreateOrder` mit beliebigen Werten
- Geben Sie in der Server Implementierung den Aufruf in die Console aus

Aufgabe 2

- gRPC kann auch Responses streamen
- Erweitern Sie das Service um eine Methode `PriceChanges` mit dem Parameter `NoParams` (leere Message)
  - Returniert einen `returns (stream Item)`
  - `PriceChanges` gibt randomisiert alle Sekunden eine Preisänderung zurück – der Client gibt diese aus
- Optional: Verwenden Sie eine `RX.NET` im Server, um die Änderungen außerhalb der Klassen (in einem eigenen Thread) übergeben zu können
- https://grpc.io/docs/tutorials/basic/csharp/

RX.NET Hint

```
await subject
   .ForEachAsync(async value =>
   {
     await responseStream.WriteAsync(value);
   },
   context.CancellationToken);
```

Aufgabe 2 Anmerkung

- Aufgabe 1 & 2 sind unabhängig (es wird nur die Entität der Aufgabe 1 wiederverwendet)
- Man returniert randomisiert ein Item (id, price) alle X Sekunden:
  - Sekunde 1: Item (234, 123.3434)
  - Sekunde 2: Item (212, 234.1234)
  - …
- Geben Sie die empfangenen Werte auf dem Client aus, bis dieser beendet wird

Was Sie aus der Übung mitnehmen sollten

- JSON over HTTP ist nicht das einzige Tool
  - Gleich wie RabbitMQ nicht das einzige Tool ist
- Analysieren Sie die nicht-funktionalen Anforderungen genau und lernen Sie zu verstehen, was Sie brauchen
  - Muss ich die Komponenten zeitlich / örtlich entkoppeln?
  - Muss der Request gespeichert / gebuffert werden? (E.g. Bestellung)
  - Welche Performance brauche ich?
  - Will ich überhaupt ressource-orientiert denken?