SYTD - gRPC Kommunikationsprotokoll

Dipl.-Ing. Paul Panhofer Bsc.

gRPC

gRPC

API - Programmschnittstelle

Eine **API** - Application Programming Interface - ist ein Satz von Befehlen, Funktionen, Protokollen und Objekten. APIs ermöglichen Anwendungen auf einfache Weise mit-einander zu kommunizieren.

gRPC

API - Programmschnittstelle

Historische Entwicklung

Zu den frühesten und bekanntesten APIs gehört die API von ebay.

- Ebay stellte seinen Nutzern einen einfachen Zugriff auf seine Seiten zur Verfügung, um Massenuploads von Inseraten zu erleichtern. (2000).
- Zwei Jahre später erschien Amazon Web Services auf der Bildfläche. Seitdem ist die Anzahl der APIs exponentiell gestiegen.

gRPC

API - Programmschnittstelle

Einsatzgebiete

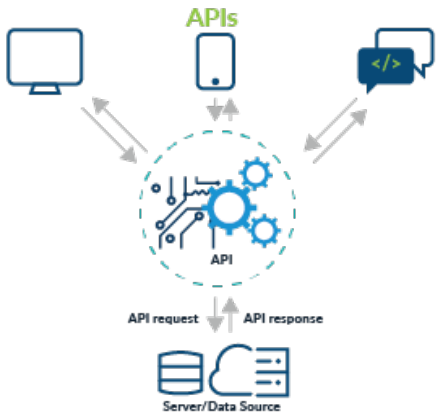
- Frontend - Backendkommunikation
- Komponentenkommunikation
- DaaS

gRPC

API - Programmschnittstelle

Einsatz: Frontend - Backendkommunikation

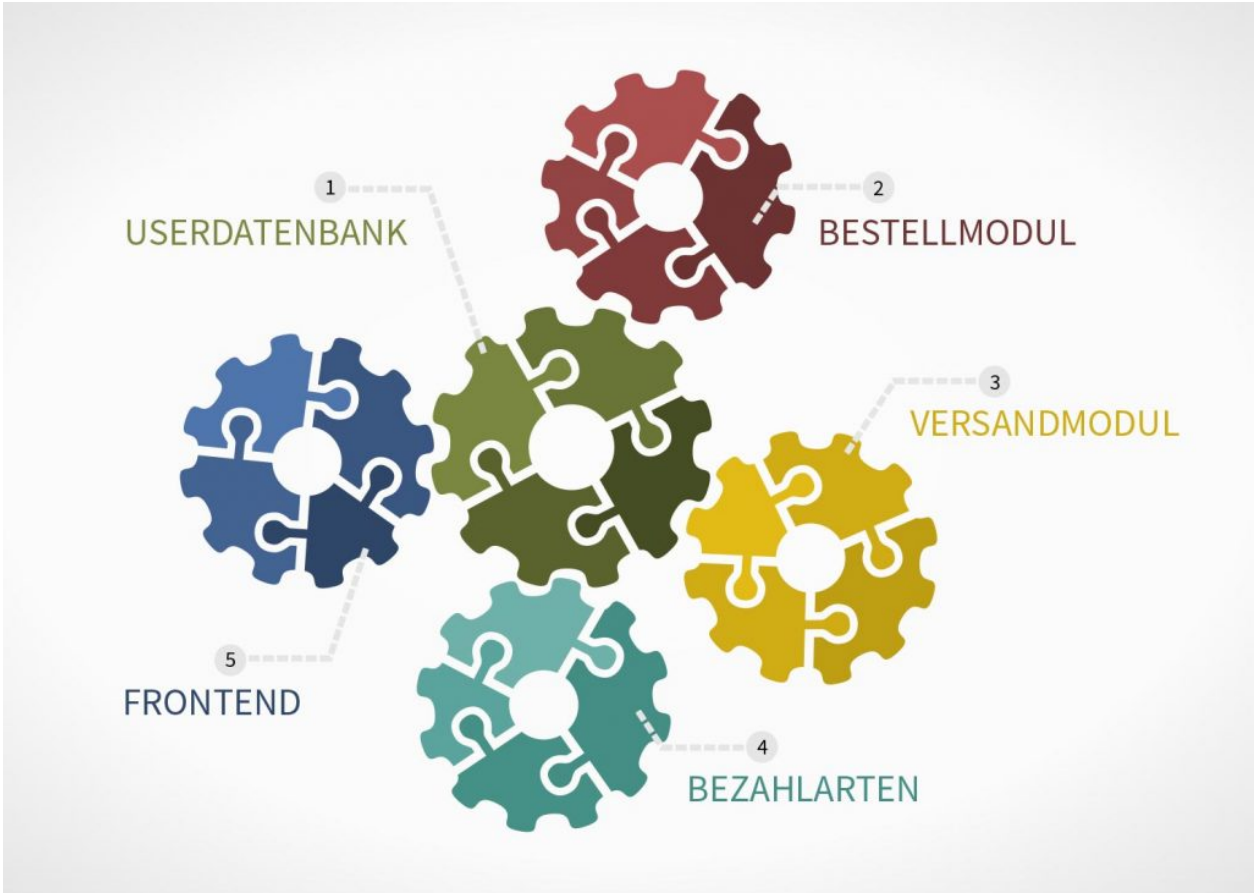
Frontend: `Angular.js, react.js, Vue.js`
Backend: `Spring, express.js, .net Core, laravel...`



gRPC

API - Programmschnittstelle

Einsatz: Komponentenkommunikation

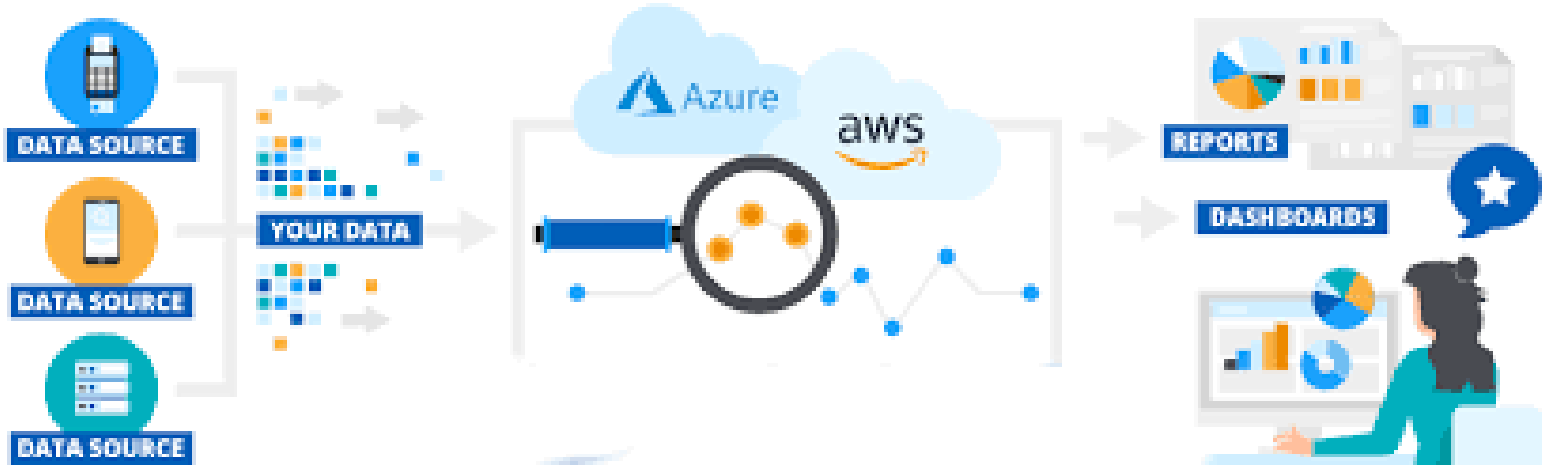


gRPC

API - Programmschnittstelle

Einsatz: DaaS

DaaS - Data as a Service - ist ein Ansatz Daten als Service in der Cloud zu Verfügung zu stellen.



gRPC

API - Programmschnittstelle

Kommunikationsprotokolle

Für die Implementierung einer API können unterschiedliche Kommunikationsprotokolle verwendet werden.

gRPC

gRPC

Kommunikationsprotokolle

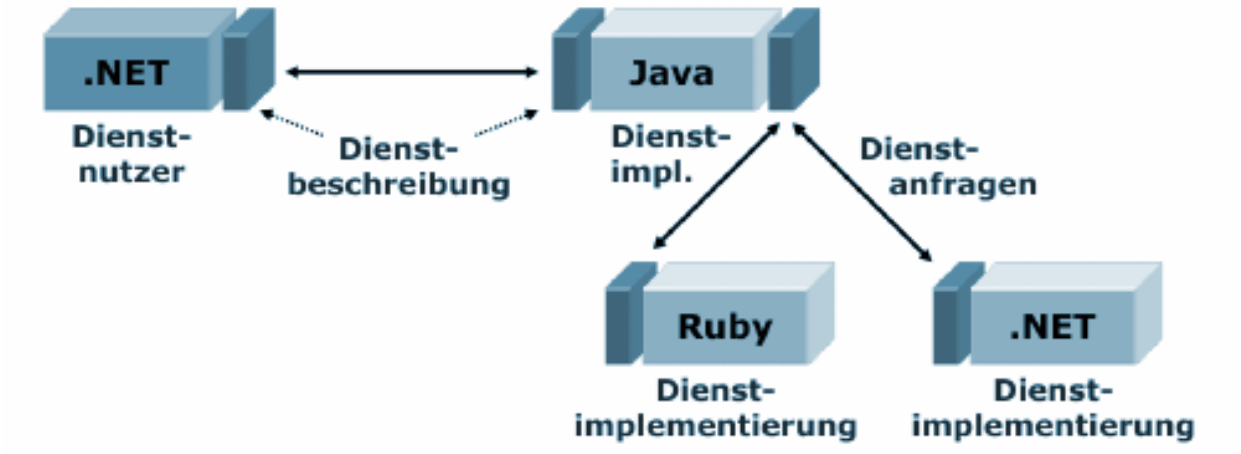
Kommunikationsprotokolle ermöglichen das **Austauschen** von Nachrichten zwischen 2 Prozessen.

Diese Prozesse können dabei im Arbeitsspeicher unterschiedlicher Netzwerkknoten ausgeführt werden.

gRPC

Kommunikationsprotokolle

Offenheit



gRPC

Kommunikationsprotokolle

API Programmierung

Für die **Programmierung** von APIs werden unterschiedliche Protokolle verwendet:

- Rest
- gRPC
- GraphQL

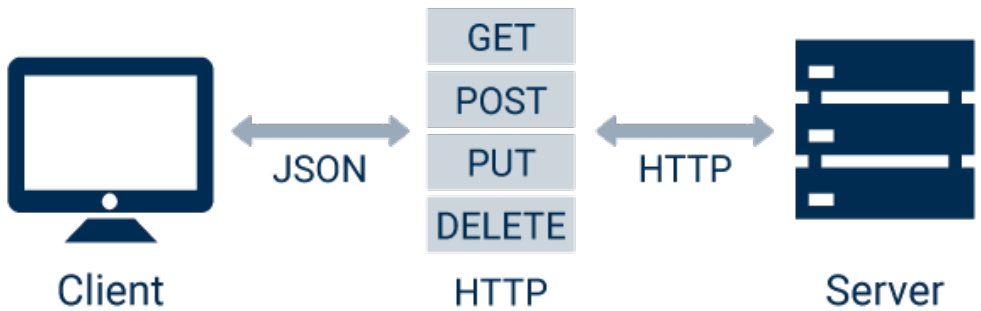
gRPC

Kommunikationsprotokolle

Rest API

Rest versteht das Internet als eine Sammlung von Ressourcen. Die REST API ermöglicht die Verwaltung von Ressourcen.

Rest verwendet zur Kommunikation das **HTTP Protokoll**.

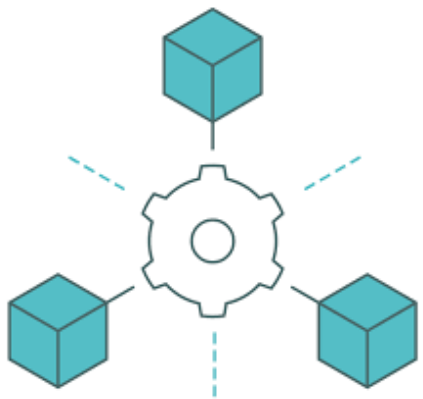


gRPC

Kommunikationsprotokolle

gRPC - Google Remote Procedure Call

gRPC ist ein Kommunikationsprotokoll zum Aufruf von Methoden/Funktionen in verteilten Systemen.



gRPC

Kommunikationsprotokolle

gRPC - Google Remote Procedure Call

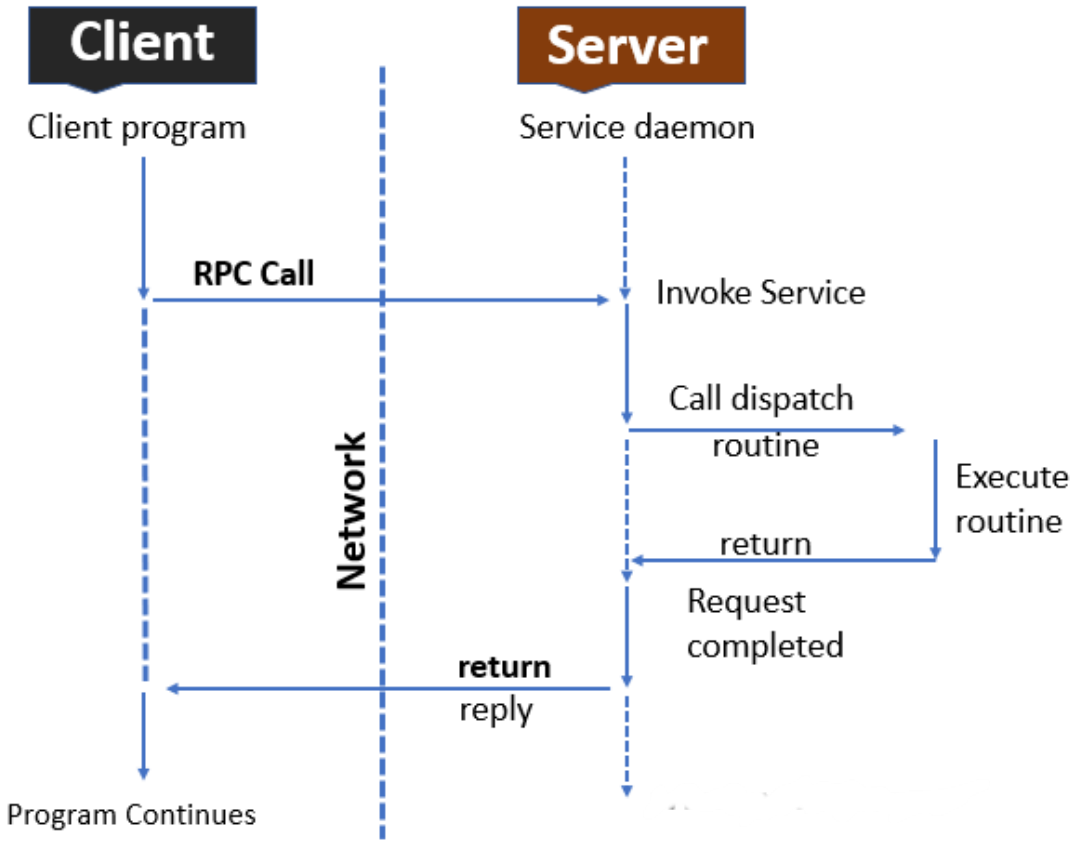
gRPC ermöglicht den Aufruf von Methoden in Objekten im Speicher eines anderen Netzwerkknotens.

Für die Kommunikation verwendet gRPC ein eigenes binäres Protokoll.

gRPC

Kommunikationsprotokolle

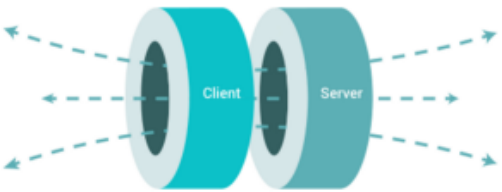
gRPC API

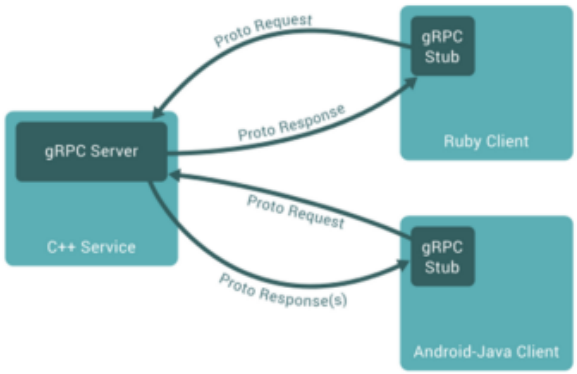


gRPC

gRPC

gRPC - Grundlagen





gRPC

gRPC - Grundlagen

gRPC ist ein Hochleistungsframework zur Programmierung von WebAPIs.

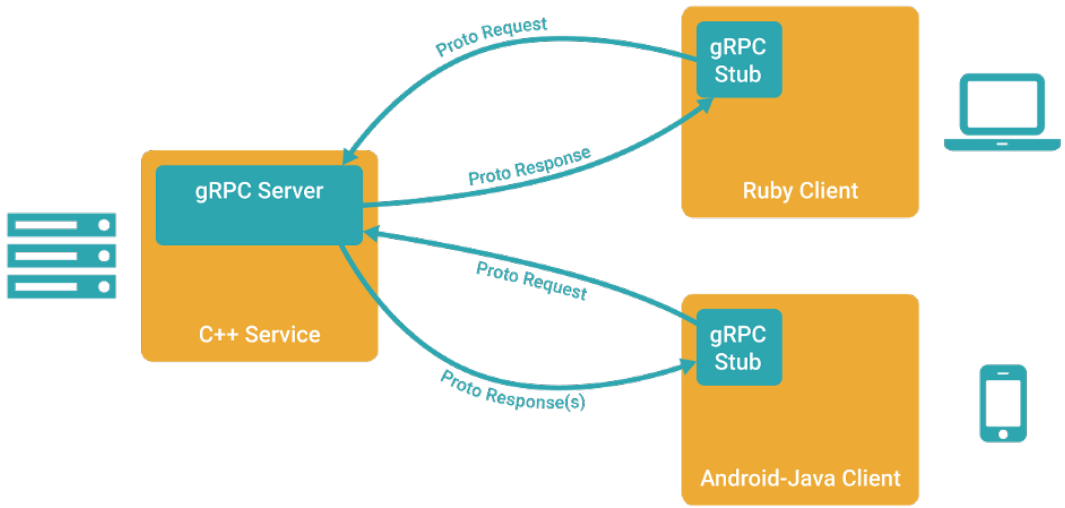
Auf Anwendungsebene optimiert gRPC das Messaging zwischen Clients und Servern.

gRPC

gRPC - Grundlagen

gRPC verwendet HTTP/2 für sein Transportprotokoll.

Zur Definition der API wird eine plattformübergreifende Interface Definition Language (IDL) verwendet.



gRPC

Protocol Buffers

Protocol Buffers - `protobuf` - ist eine Schnittstellen Beschreibungssprache die von google entwickelt wurde.

gRPC

Protocol Buffers

Protofile Struktur

```
// Proto Version
syntax = "proto3";

// Namespace
package hello;

// Schnittstellendefinition
service Greeter {
   rpc Greet(HelloRequest) returns (HelloReply) {}
}

// Datatypendefinition
message HelloRequest { string name = 1; }
message HelloReply { string message = 1; }
```


gRPC

Protocol Buffers

Protofile Struktur: Beispiel

```
// Version
import "google/protobuf/empty.proto";
syntax = "proto3";

// Interface with multiple methods
service Greeter {
   rpc Greet(HelloRequest) returns (HelloReply) {}
   rpc CreateReply(google.protobuf.Empty)
       returns (EntityReply) {}
}
```


gRPC

gRPC

Protocol Buffers

Datatype Definition - Syntax

```
// Syntax Datatypes
message <name> {
   <field rule> <type> <field name> = <parameter index>
   ...
}

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


gRPC

Protocol Buffers

Datatype Definition - Field Rule

```
// Syntax Datatypes
message <name> {
   <field rule> <type> <field name> = <parameter index>
}
```

Field Rules sind Attribute zur Beschreibung von Nachrichtenfeldern.

- repeated: list of field

gRPC

Protocol Buffers

Datatype Definition - Field Rule

```
// Syntax Datatypes
message <name> {
   <field rule> <type> <field name> = <parameter index>
}

message ProjectRequest {
    int32 id = 1;
    string title = 2;
    string description = 3;
    string token = 4;
}
```


gRPC

Protocol Buffers

Datatype Definition - Datatype

```
// Syntax Datatypes
message <name> {
   <field rule> <type> <field name> = <parameter index>
}
```

"proto"	"c#"
double	double
float	float
int32	int
bool	bool
string	string

gRPC

Protocol Buffers

Datatype Definition - Datatype

```
// Syntax Datatypes
message <name> {
   <field rule> <type> <field name> = <parameter index>
}

message SubprojectRequest {
    int32 id = 1;
    string title = 2;
    int32 focusresearch = 3 [default = 0];
}
```


gRPC

Protocol Buffers

Datatype Definition - Parameter Index

```
// Syntax Datatypes
message <name> {
   <field rule> <type> <field name> = <parameter index>
}
```

Der Parameter Index wird von Protobuf verwendet um Parameter innerhalb des Frameworks zu identifizieren.

gRPC

gRPC

Protocol Buffers

Aufzählungstypen

Zur Definition von Aufzählungstypen können in einem Protofile enums definiert werden.

gRPC

Protocol Buffers

Aufzählungstypen

```
// Syntax Datatypes
message ProjectRequest {
    string title = 1;
    string description = 2;
    EProjectType type = 3
      [ default = REQUEST_FUNDING_PROJECT ];
}

enum EProjectType {
   REQUEST_FUNDING_PROJECT = 0;
   RESEARCH_FUNDING_PROJECT = 1;
   MANAGEMENT_PROJECT = 2;
}
```


gRPC

gRPC

Protocol Buffers

record - Strukturierungstyp

Zur Definition eines Strukturierungstyps wird das Schlüsselwort message verwendet.

gRPC

Protocol Buffers

record - Strukturierungstyp

```
// Syntax: Strukturierungstyp
message ProjectRequest {
   Project project = 1;
}

message Project {
   string title = 1;
   int32 projectId = 2;
}
```


gRPC

gRPC

Protocol Buffers

Vorgefertigte Datentypen

Google stellt neben der begrenzten Zahl an Standard-datentypen eine Reihe nützlicher Datentypdefinitionen zur Verwendung in Proto Files zur Verfügung.

Bevor solche Datentypen verwendet werden können müssen sie im proto File **importiert** werden.

gRPC

Protocol Buffers

Vorgefertigte Datentypen

```
import "google/protobuf/empty.proto";
import "google/protobuf/any.proto";
...
service TraderService {
    rpc Notify(google.protobuf.Empty)
        returns (google.protobuf.Empty);
}

message HelloRequest {
    string name = 1;
    google.protobuf.any = 2;
}
```


gRPC

Protocol Buffers

Vorgefertigte Datentypen - grpc tool

Datentypen die nicht im `google.protobuf.*` Namespace liegen müssen vor ihrer Verwendung importiert werden.

Dazu wird unter c das `dotnet-grpc` Tool bereitgestellt.

```
dotnet tool install -g dotnet-grpc
```


gRPC

gRPC

Protocol Buffers

Automapper

AutoMapper werden verwendet um Daten zwischen Objekten zu kopieren.

gRPC

Protocol Buffers

Automapper

```
public class ProjectDTO {

    public int ProjectId { get; set; }

    public string Title { get; set; }

    public float Funding { get; set; }

}
```


gRPC

Protocol Buffers

Automapper

```
[Table("PROJECTS")]
public class Project {

    [Key, GeneratedValue]
    [Column("PROJECT_ID")]
    public int Id { get; set; }

    [Required, StringLength(50)]
    [Column("TITLE")]
    public string Title { get; set; }

    public float Funding { get; set; }

}
```


gRPC

Protocol Buffers

Automapper: Requirements

- Nugets: `AutoMapper`, `AutoMapper.Extensions.Microsoft.DependencyInjection`
- Program.cs: `builder.Services.AddAutoMapper(typeof(StartupBase))`

gRPC

Protocol Buffers

Automapper

gRPC

gRPC

gRPC Service

Zur Implementierung eines gRPC Services stellt .net Core den `grpc Service` Typ zur Verfügung.

gRPC Service erweitern, eine aus dem proto File generierte Klasse, die `ServiceBase`.

gRPC Service

Beispiel: proto Schnittstelle

```
syntax = "proto3";

service Greeter {
   rpc Greet(HelloRequest) returns (HelloReply) {}
}

message HelloRequest { string name = 1; }
message HelloReply { string message = 1; }
```


gRPC

gRPC Service

Beispiel: Service Implementierung

```
public class GreeterService : GreeterBase {

    public override Task<HelloReply> Greet(
        HelloRequest r, ServerCallContext context
    ) {
      return Task.FromResult(
          new HelloReply{message = $"hello {r.name}!"};
      );
    }

}
```

gRPC Service

Beispiel: Client Aufruf

```
public class Program {
   var channel = GrpcChannel.ForAddress(
      "http://localhaost:5242"
   );
   var client = new GreeterServiceClient(channel);
   var reply = client.Greet(
      new HelloRequest{ Name="Freddy" }
   );

   Console.Writeln(reply.Message);
}
```


gRPC

gRPC

gRPC Kommunikation

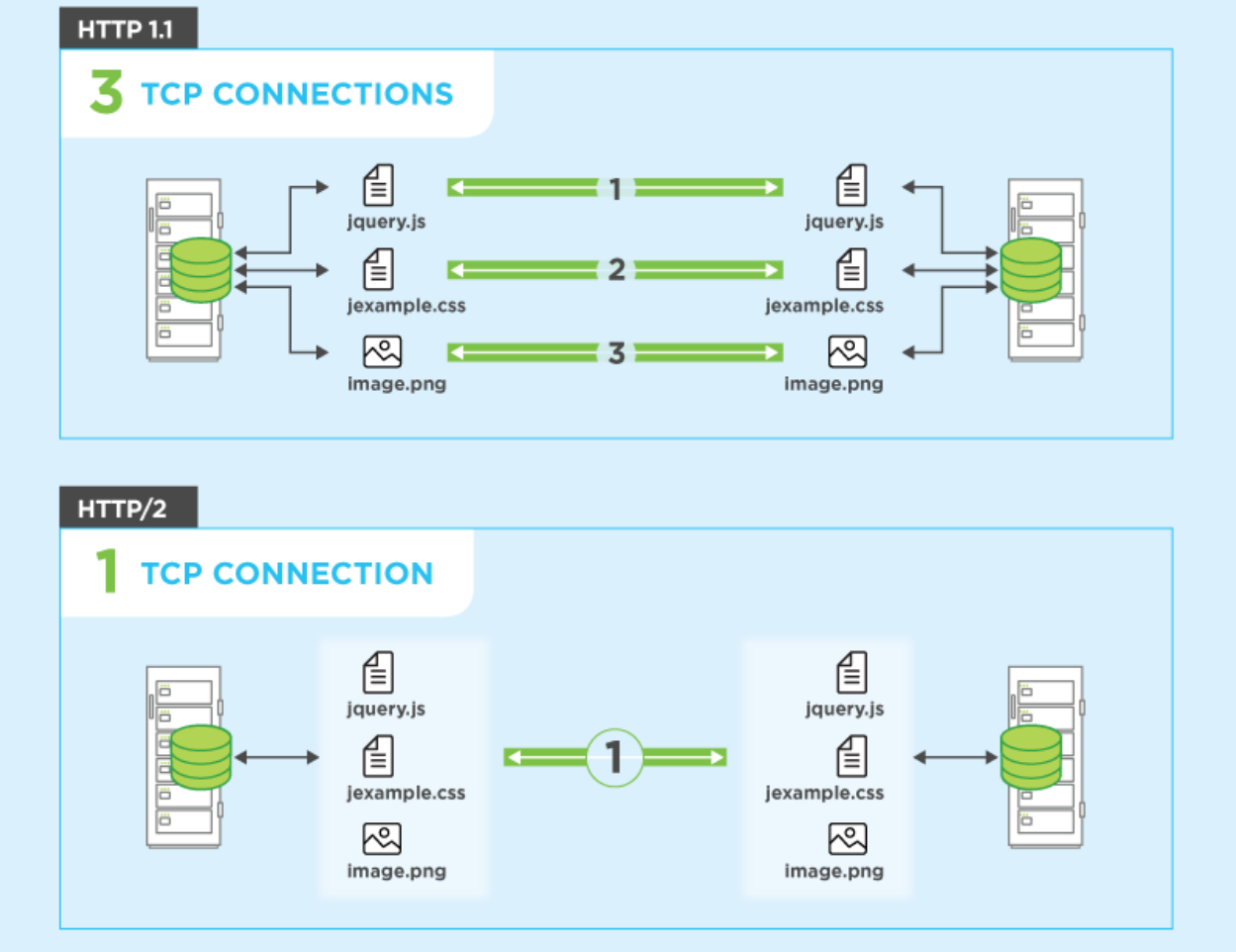
Http 1.0 vs. Http 2.0

gRPC verwendet für den Austausch von Informationen das **http 2** Protokoll.

gRPC

gRPC Kommunikation

Http 1.0 vs. Http 2.0



gRPC

gRPC Kommunikation

gRPC vs. Rest

Rest WebAPI

- ConnectionCount: TCP Connection per request
- Messageformat: Plaintext headers
- Kommunikation: Unary
- Messagedelivery: Client -> Server
- Api Methods: get/put/post ...

gRPC

gRPC Kommunikation

gRPC vs. Rest

gRPC WebAPI

- ConnectionCount: Single TCP Connection
- Messageformat: binär
- Messagedelivery: Streaming - Multiplexing
- Api Methods: Free design

gRPC

gRPC

API Typen

gRPC unterstützt für den Nachrichtenaustausch zwischen Client und Service 4 unterschiedliche Formen.

- Unary communication
- Serverside streaming
- Clientside streaming
- Bidirectional streaming

gRPC

API Typen

Unary communication

Ein Service antwortet mit einer einzelnen Nachricht auf den Request eines Clients.

Usecase: request response

gRPC

Unary communication

proto file

```
service Greet {
   rpc GreetAsync(Request) returns (Response) {}
}

message Request { ... }
message Response { ... }
```


gRPC

Unary communication

Beispiel: Service Implementierung

```
public class GreeterService : GreeterBase {

    public override Task<Response> GreetAsync(
        Request r, ServerCallContext context
    ) {
      return Task.FromResult(
          new HelloReply{message = $"hello {r.name}!"};
      );
    }

}
```


gRPC

Unary communication

Beispiel: Client Aufruf

```
public class Program {
   var channel = GrpcChannel.ForAddress(
      "http://localhaost:5242"
   );
   var client = new GreeterServiceClient(channel);
   var reply = client.Greet(
      new Request{ Name="Freddy" }
   );

   Console.Writeln(reply.Message);
}
```


gRPC

API Typen

Server streaming

Ein Service antwortet mit einem kontinuierlichem Strom von Nachrichten auf den Request eines Clients.

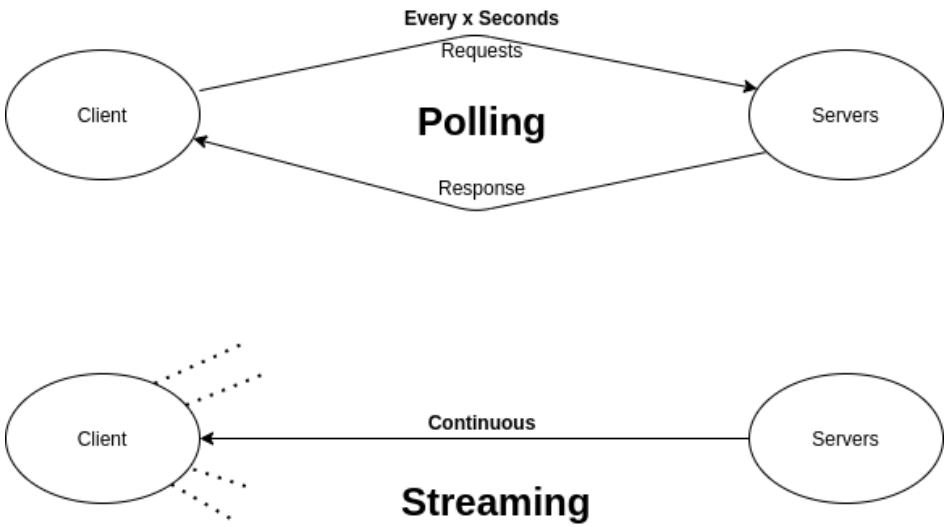
Usecase: Die Antwort des Services setzt sich aus mehreren Teilen zusammen. Das Service schickt immer dann eine Nachricht an den Client, sobald eine der Teilnachrichten berechnet ist. z.B.: live feed, chat usw.

gRPC

API Typen

Serverside streaming

Serverside Streaming wir oft auch verwendet um Polling zu verhindern.



gRPC

Serverside streaming

proto file

```
service DataService {
   rpc UpdateData(Request) returns (stream Response) {}
}

message Request { ... }
message Response { ... }
```


gRPC

Serverside streaming

Beispiel: Service Implementierung

```
public class DataService : DataServiceBase {
    public override async Task UpdateData(
        Request r,
        IServerStreamWriter<Response> responseStream
        ServerCallContext context
    ) {
      foreach(int i in Enumerable.Range(1,10)){
        await responseStream.WriteAsync(
          new Response{message = $"{i}"}
        );
      }
    }
}
```


gRPC

Serverside streaming

Beispiel: Client Aufruf

```
public class Program {
    var channel = GrpcChannel.ForAddress(
        "http://localhaost:5242"
    );
    var client = new DataServiceClient(channel);
    var reply = client.UpdateData(
        new Request{ Name="Freddy" }
    );

    await foreach (
        var data in reply.ResponseStream.ReadAllAsync
    ){
        Console.WriteLine($"calculated number: {data}");
    }
}
```


gRPC

API Typen

Clientside streaming

Ein Client schickt kontinuierlich Nachrichten an ein Service. Das Service antwortet mit einem einzelnen Signal.

Usecase: Upload, Datenübertragung von Client zu Server

gRPC

Clientside streaming

proto file

```
service DataService {
   rpc UploadAsync(stream Request) returns (Response) {}
}

message Request { ... }
message Response { ... }
```


gRPC

ClientSide streaming

Beispiel: Service Implementierung

```
public class DataService : DataServiceBase {
    public override async Task<Response> UploadAsync(
        IAsyncStreamReader<Request> requestStream,
        ServerCallContext context
    ) {
      var data = new List<Project>();
      await foreach(var request in
          requestStream.ReadAllAsync()){
        data.Add(new Project{ Id = request.Project.Id});
      }

      return new Response{message="Calculated data"};
    }
}
```


gRPC

ClientSide streaming

Beispiel: Client Aufruf

```
public class Program {
    var channel = GrpcChannel.ForAddress(
        "http://localhaost:5242"
    );
    var client = new AuctionServiceClient(channel);
    using var duplexStream = client.Bet();

    // 1.) Das Konsumieren der Nachrichten vom Server wird
    // in einen eigenen Backgroundthread verlagert.
    var receiverTask = Task.Run(
        async () => {
            ... // 3.) Receiving Content from Service
        }
    );
    // 2.) Sending Data ...
}
```


gRPC

API Typen

Bidirectional streaming

Client und Server schicken einen kontinuierlichen Strom von Nachrichten.

Usecase: Signalverarbeitung, Auction, GUIKomponente zur Datenfilterung

gRPC

Bidirectional streaming

proto file

```
service AuctionService {
   rpc Bet(stream Request) returns (stream Response) {}
}

message Request { ... }
message Response { ... }
```


gRPC

Bidirectional streaming

Beispiel: Service Implementierung

```
public class AuctionService : AuctionServiceBase {
    public override async Task Bet(
        IAsyncStreamReader<Request>  requestStream,
        IServerStreamWriter<Response> responseStream
        ServerCallContext context
    ) {
      await foreach(
        var request in requestStream.ReadAllAsync()
      ){
        var index = request.Index;
        ...
      }
    }
}
```


gRPC

Bidirectional streaming

Beispiel: Service Implementierung

```
public class AuctionService : AuctionServiceBase {
    public override async Task Bet(...) {
      await foreach(
         var request in requestStream.ReadAllAsync()
      ){
         var index = request.Index;

         foreach(int i in Enumerable.Range(1,index)){
            await responseStream.WriteAsync(
               new Response{message = $"{i}"}
            );
         }
      }
    }
}
```


gRPC

Bidirectional streaming

Beispiel: Client Aufruf

```
public class Program {
    var channel = GrpcChannel.ForAddress(
        "http://localhaost:5242"
    );
    var client = new AuctionServiceClient(channel);
    using var duplexStream = client.Bet();

    // 1.) Das Konsumieren der Nachrichten vom Server wird
    // in einen eigenen Backgroundthread verlagert.
    var receiverTask = Task.Run(
        async () => {
            ... // 3.) Receiving Content from Service
        }
    );
    // 2.) Sending Data ...
}
```


gRPC

Bidirectional streaming

Beispiel: Client Aufruf

```
public class Program {
    ...
   var receiverTask = Task.Run(
      async () => {
          ... // 3.) Content
      }
   );
   // 2.) Sending Data to Service
   await duplexStream.RequestStream.WriteAsync(
      new Request{Index = 5};
   );
   await duplexStream.RequestStream.WriteAsync(...);
   await receiverTask;
}
```


gRPC

Bidirectional streaming

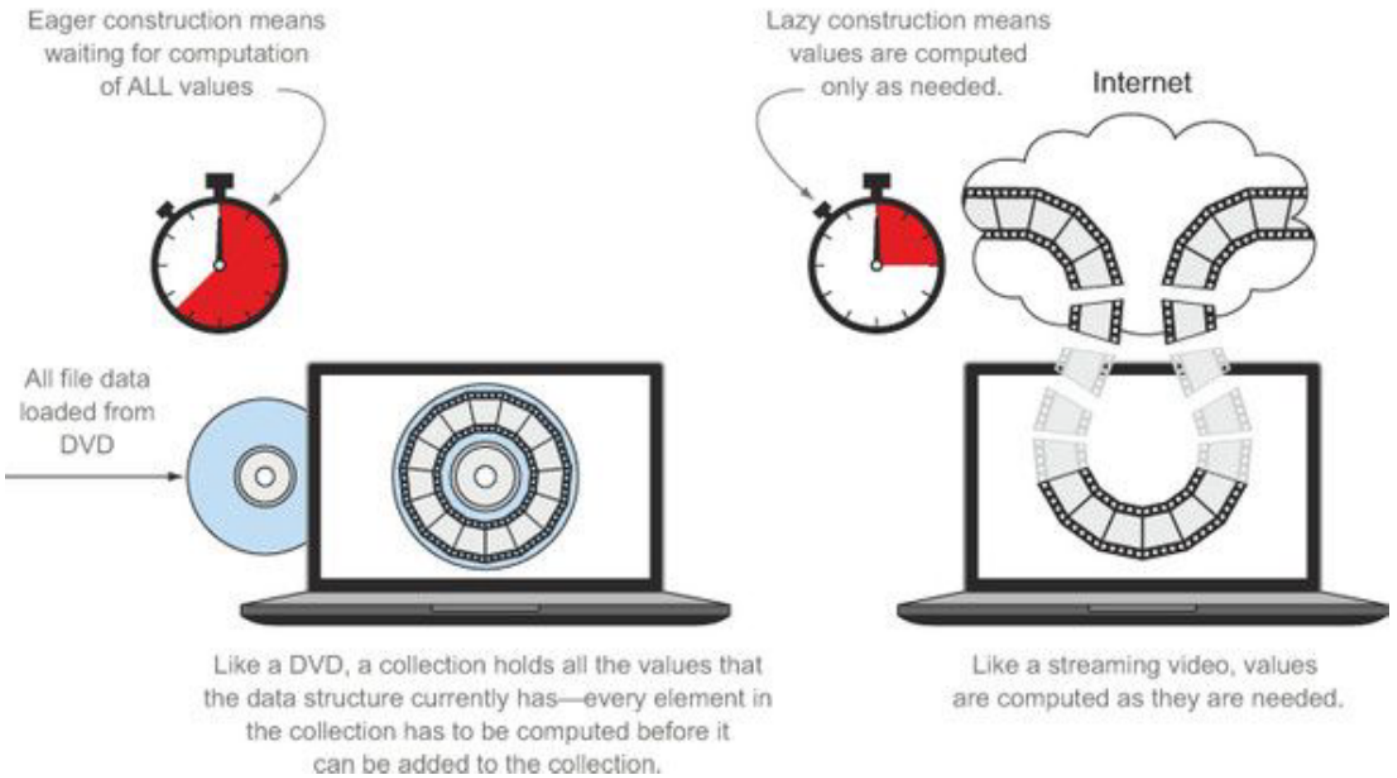
Beispiel: Client Aufruf

```
public class Program {
    ...
     // Receiving Data from Service
    var receiverTask = Task.Run(
        async () => {
            await foreach(var response in
                duplexStream.ResponseStream.ReadAllAsync()){
              Console.WriteLine($"index: {response.Message}")
            }
        }
    );
    ...
}
```


gRPC

API Typen

Stream vs. Unary



gRPC

1. Prinzipien verteilter Programmierung
   - API - Programmschnittstelle
   - Kommunikationsprotokolle
   - gRPC - Grundlagen
2. IDL - Protocol Buffers
   - Protofile Struktur
   - Message Definition
   - enum - Aufzählungstypen
   - record - Strukturierungstyp
   - Datentyp Definition
   - Automapper
3. Service Implementierung
   - gRPC Service/Client
4. Nachrichtenaustausch
   - Http 2.0
   - API Typen
   - Scalability

gRPC

Scalability

Als Skalierbarkeit wird die Fähigkeit eines Systems definiert trotz wachsender Last seinen Dienst aufrechthalten zu können.

- Server: Async
- Client: Async or Blocking

gRPC