SYTD - Programmierung verteilter Systeme - Microserviceprogrammierung

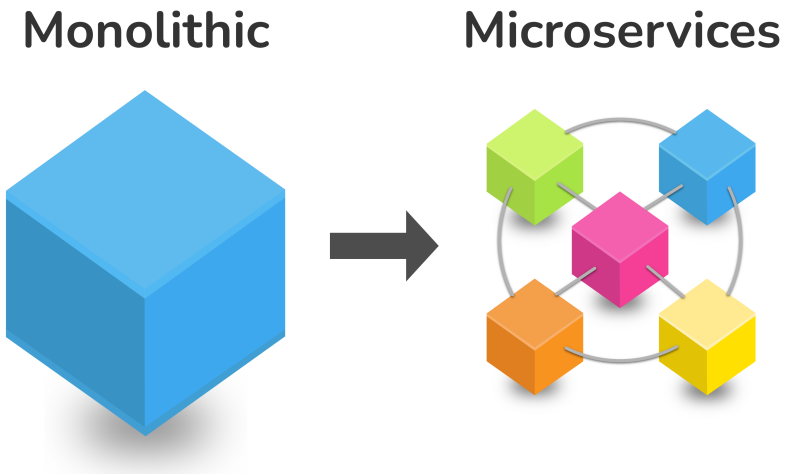
Dipl.-Ing. Paul Panhofer Bsc.

16. Oktober 2022

Microservice

Microservice

An architectural style that structures an application as a collection of independently deplayoble services.



Microservice

Monolith Advantages

- Architecture for small and midsized projects
- Toolsupport
- Great code reuse
- Easy to run locally
- Easy to debug
- Easy to build and deploy

Microservice

Monolith Disadvantages

- Scalling issues in high demand environments
- Important to code to standards - Big ball of mud
- Slow and infrequent deployments
- Long testing and stabilization periods
- Hard to scale
- Hard to adopt new technology

Microservice

Microservice Advantages

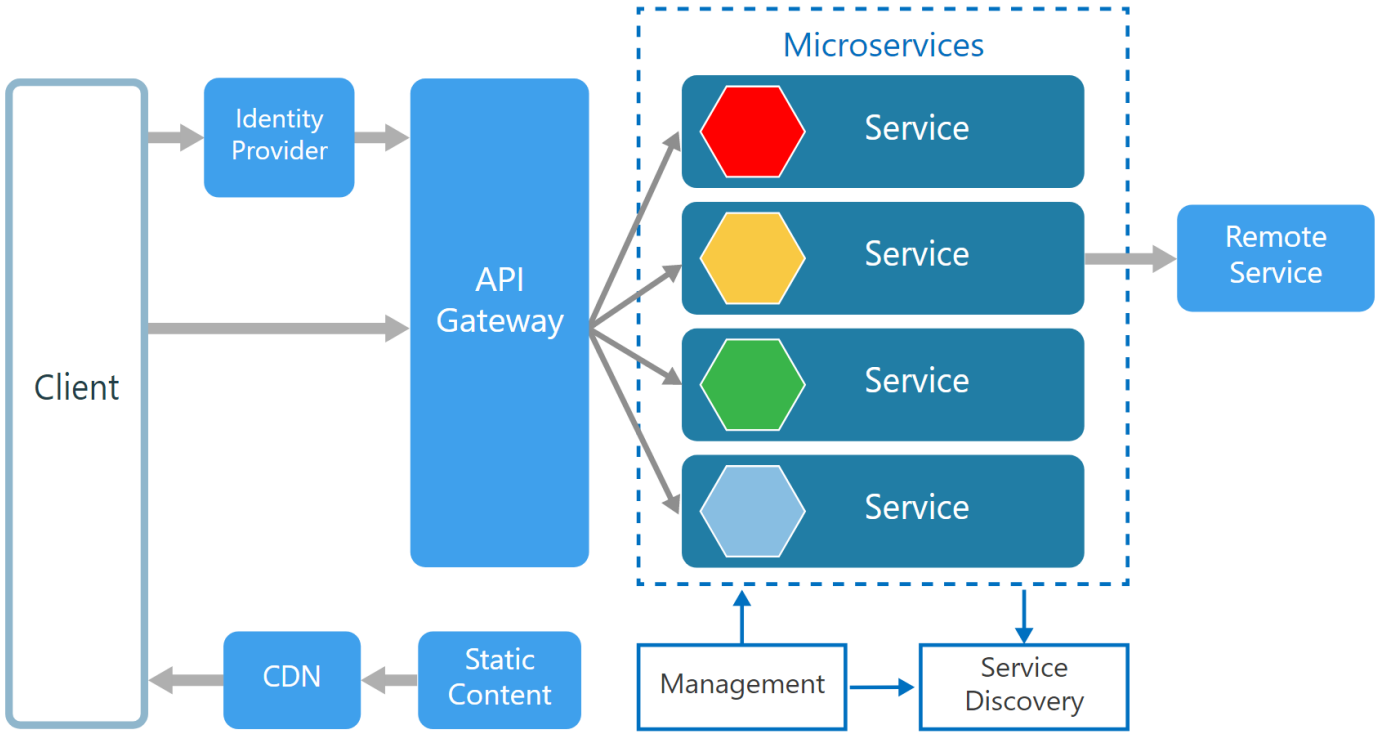
- Used in high demand environments
- Highly scalable architecture
- Service has a small, easy to understand code base
- Service quick to build
- Service independent faster deployment
- Independently scalable
- Isolation from failures
- Easy to adopt new technologies

Microservice

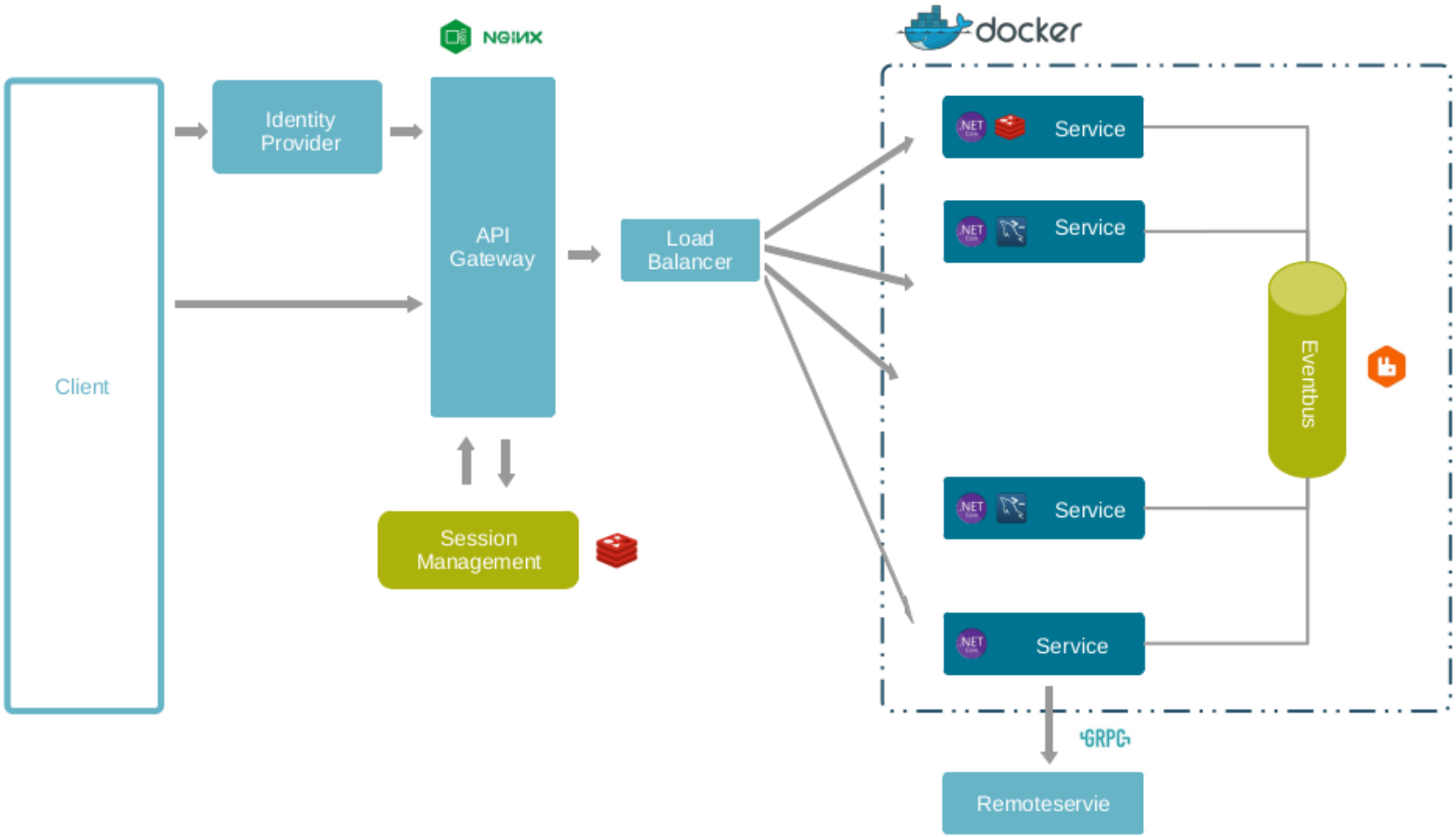
Microservice Disadvantages

- Hard to set up initial Project structure
- Adds the complexity of distributes systems
- Shared code in separate libraries
- No good tooling for distributed apps
- Releasing features across services is hard
- Distributes Transaction model - 2PC

Microservicearchitektur



Komponentenübersicht

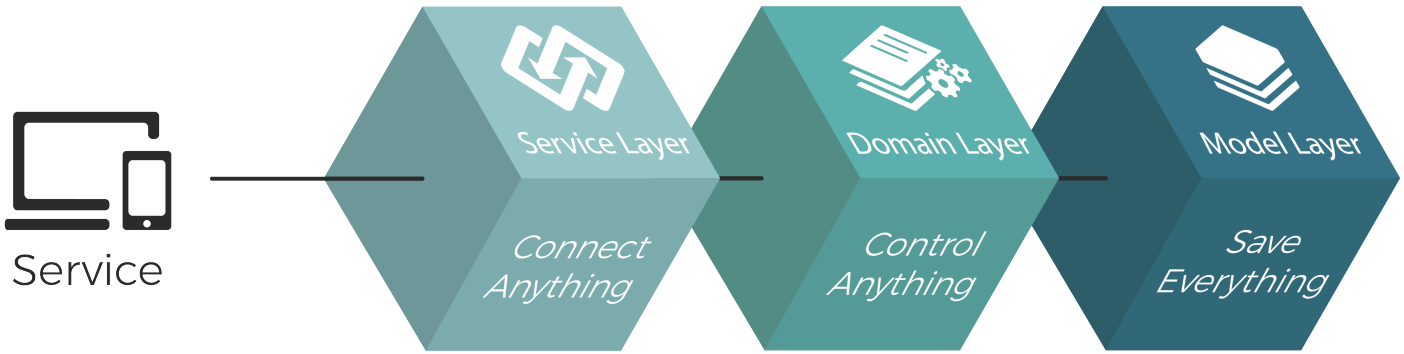


Komponentenübersicht

- API Controller: Serviceimplementierung
  Technologie: `.net Controller`
- Eventbus: Nachrichtenaustausch zwischen Servicen
  Technologie: `RabbitMQ`
- API Gateway: Schnittstelle zum Microservice
  Technologie: `Niginx`
- Session Management: Technologie: `Redis`
- Deployment: Technologie: `Docker/Kubernetes`

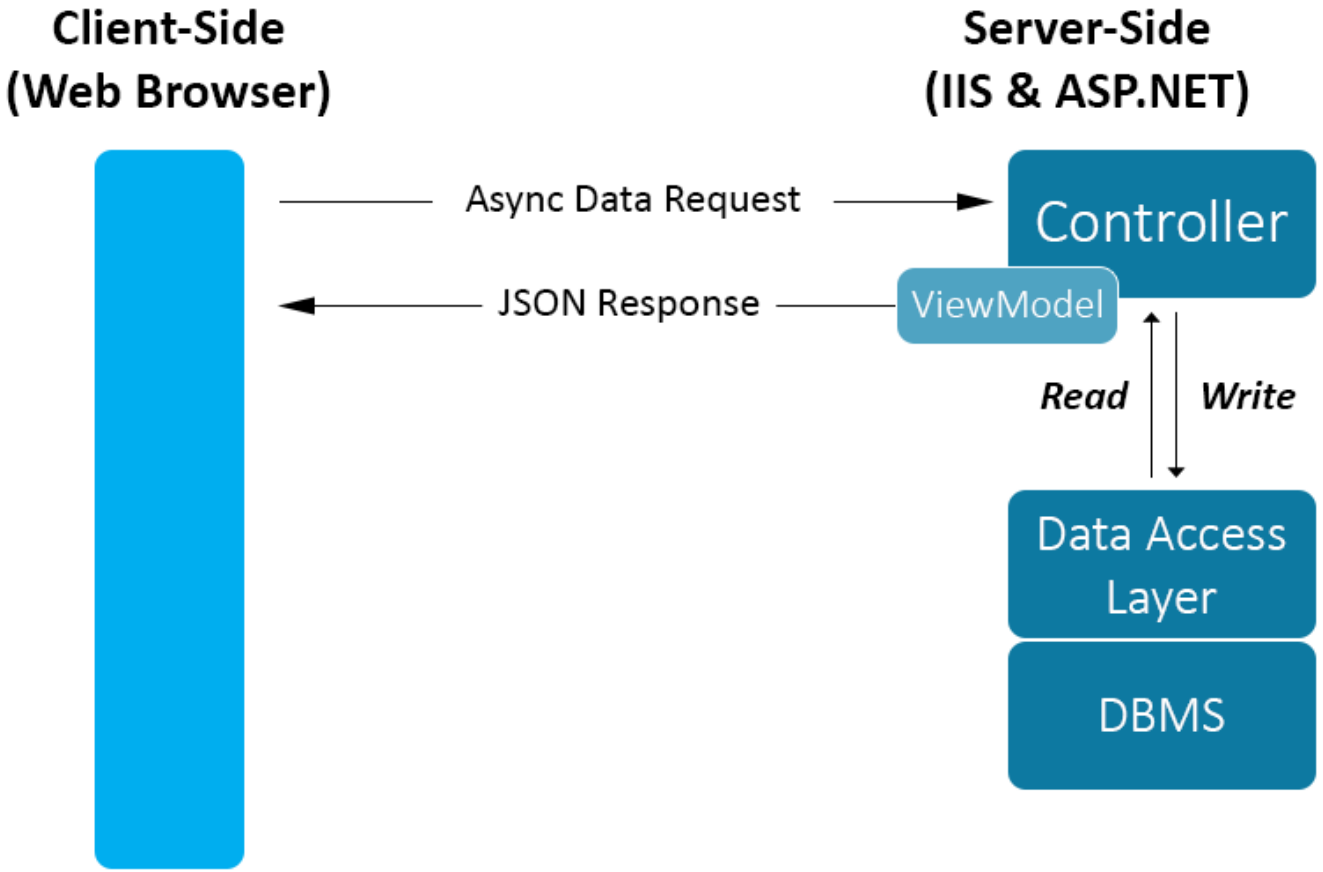
API Controller

Schichtenmodell



Controller

Kommunikationsschnittstelle



Controller

HTTP Request


```
POST /api/recipes HTTP/1.1
Host: recipes.contoso.com
User-Agent: demo/client
Accept: application/json
Content-Type: application/json
```


Controller

HTTP Response


```
HTTP/1.1 200 OK
Connection: close
Date: Mon, 18 Jan 2021 18:13:48 GMT
Content-Type: application/json; charset=utf-8
Server: Kestrel
Content-Length: 1368

[
  {
```


Controller

Implementierung

```
// Modelklasse
public class Person {

    private Guid Id { get; set; }

    [Required]
    [StringLength(30)]
    private string FirstName { get; set; }

    [Required]
    [StringLength(30)]
    private string LastName { get; set; }

}
```


Controller

Implementierung - ApiController, Route

```
/* 1.) ApiController: AC ist ein FrameworkHook der AC
 *     Objekte als Externe Schnittstelle zur Verarbeitung
 *     von HTTP Requests bereitstellt.
 */
[ApiController]
/* 2.) Route: Das Route Attribut definiert die URL der
 *     Ressource.
 */
[Route("people")]
public class PersonController : ControllerBase {...}
```


Controller

Implementierung - ControllerBase

```
[ApiController]
[Route("people")]
/* 3.) ControllerBase: ControllerBase stellt eine
 *     Implementierung des HTTP Protokolls dar.
 *     Klassen die von ControllerBase erben koennen
 *     HttpRequests entgegennehmen bzw. HttpResponses
 *     absetzen
 */
public class PersonController : ControllerBase {...}
```


Controller

Implementierung - Call Method

```
[ApiController]
[Route("people")]
public class PersonController : ControllerBase {
    /* 1.) Damit eine bestimmte Methode in einem Contro-
     *      ller aufgerufen werden kann, muss ein HTTP
     *      Request die URL des Controllers und mit der
     *      entsprechenden HTTP Methode abgesetzt worden
     *      sein.
     */
    [HttpGet("ping")]
    public ActionResult<string> Ping(){
        ...
    }

}
```


Controller

Implementierung - HttpGet

```
[ApiController]
[Route("people")]
public class PersonController : ControllerBase {
    /* 2.) Das HttpGet Attribut definiert die HttpMethode
     *      auf die die Methode registriert ist.
     */
    [HttpGet("ping")]
    public ActionResult<string> Ping(){
        ...
    }

}
```


Controller

Implementierung - ActionResult

```
[ApiController]
[Route("people")]
public class PersonController : ControllerBase {
    /* 3.) Ueber den ActionResult wird der StatusCode
     *      des HTTP Responses gesteuert.
     */
    [HttpGet("ping")]
    public ActionResult<string> Ping(){
        ...
    }

}
```


Controller

HTTP Status Codes

Code	Status
100 - 199	Informational
200 - 299	Successful
300 - 399	Redirection
400 - 499	Client Errors
500 - 599	Server Errors

Controller

Action	Method	Success	Failure
Create	POST	201 (Created)	400 (Bad Request)
Read	GET	200 (Ok)	404 (Not Found)
Update	PUT / PATCH	204 (No Content)	404 (Not Found)
Delete	DELETE	204 (No Content)	400 (Bad Request)

Controller

Implementierung - Read

```
[ApiController]
[Route("people")]
public class PersonController : ControllerBase {

    // URL: ../people/{id}
    // HTTP METHOD: GET
    [HttpGet("{id}")]
    public async Task<ActionResult<Person>> ReadAsync(int
        id){
         var p = await personRepository.ReadAsync(id);

         if( p is null) return NotFound();

         return Ok(p);
    }
}
```


Controller

Implementierung - Create

```
[ApiController]
[Route("people")]
public class PersonController : ControllerBase {

    // URL: ../people
    // HTTP METHOD: POST
    [HttpPost]
    public async Task<ActionResult<Person>>
        CreateAsync(Person p){
         p = await personRepository.Create(p);

         return CreatedAtAction(
             nameof(Read), new {Id = p.Id}, p
         );
    }
}
```


Controller

Implementierung - Update

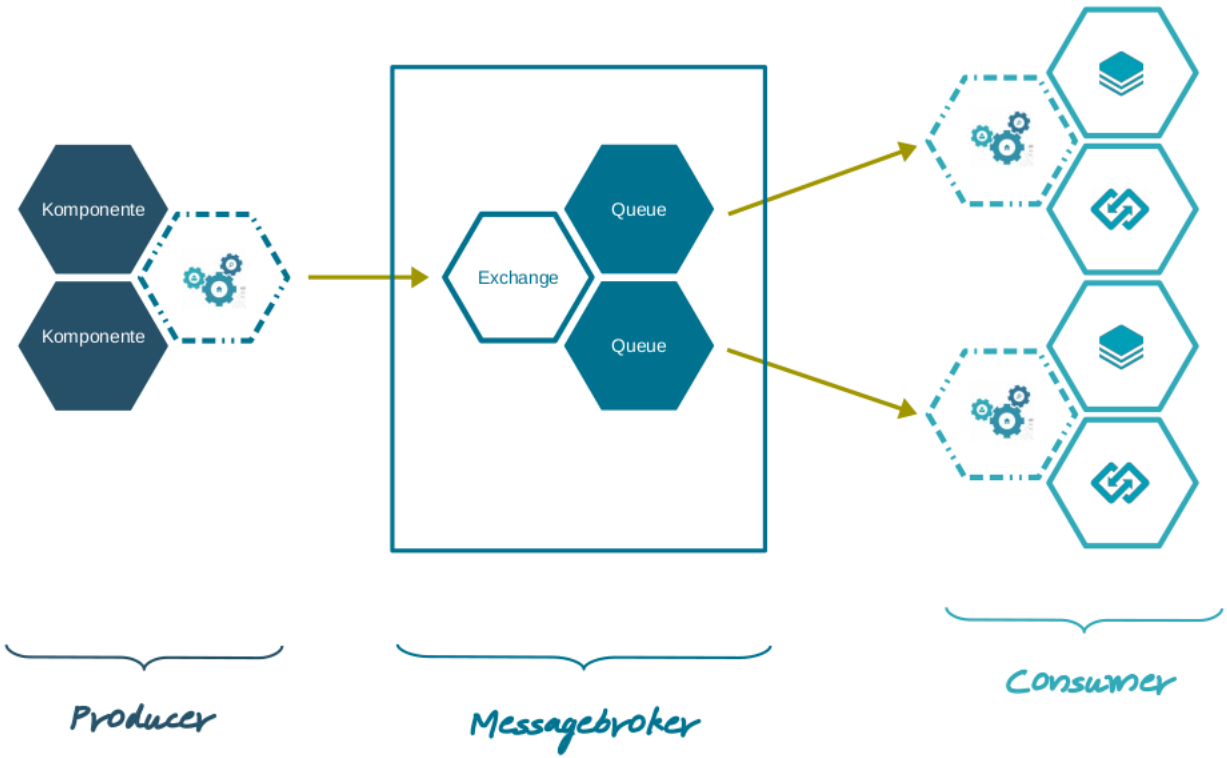
```
...
public class PersonController : ControllerBase {
    // URL: ../people/{id}
    // HTTP METHOD: PUT
    [HttpPut("{id}")]
    public async Task<ActionResult> UpdateAsync(
        int id, Person p
    ){
        var data = await personRepository.ReadAsync(id);

        if(data is null) return NotFound();
        await personRepository.UpdateAsync(p);

        return NoContent();
    }
}
```


Controller

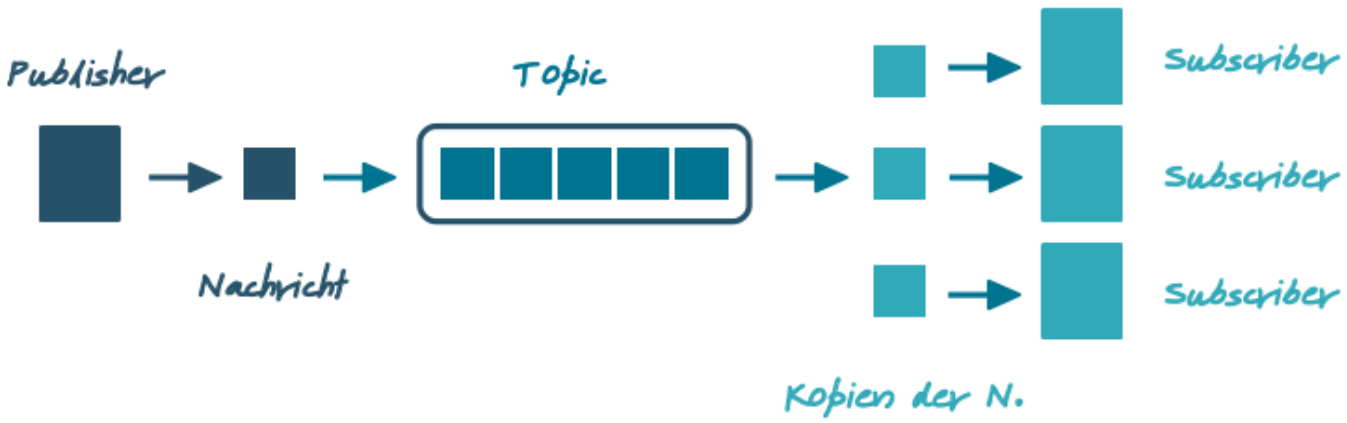
Implementierung - Delete

```
...
public class PersonController : ControllerBase {
    // URL: ../people/{id}
    // HTTP METHOD: PUT
    [HttpDelete("{id}")]
    public async Task<ActionResult> UpdateAsync(
        int id, Person p
    ){
        var data = await personRepository.ReadAsync(id);

        if(data is null) return NotFound();
        await personRepository.UpdateAsync(p);

        return NoContent();
    }
}
```


Kommunikation



Kommunikation



Implementierung

Artefakte

- Configuration, Nuggets
- Eventpublisher
- Eventsubscriber
- Eventprozessor

Implementierung

Configuration

- Nuggets: `RabbitMQ.Client`

Implementierung
Configuration

- Configuration: `appsetting.json`

```
{
    ...
    "RabbitMQHost":"localhost",
    "RabbitMQPort":"5672",
    "EventBusExchange":"trigger"
}
```


Implementierung

Eventpublisher: IMessageBusClient

```
public interface IMessageBusClient {
    void Publish(string message);
}
```


Implementierung

Eventpublisher: MessageBusClient

```
using Microsoft.Extensions.Configuration;
using RabbitMQ.Cient;

public class MessageBusClient : IMessageBusClient {
   private readonly IConfiguration _configuration;
   private readonly string _exchange;

   public MessageBusClient(IConfiguration configuration){
      _configuration = configuration;

      var factory = new ConnectionFactory(){
         HostName = _configuration["RabbitMQHost"],
         Port = int.Parse(_configuration["RabbitMQPort"])
      }
      _exchange = _configuration["EventBusExchange"];
   }
}
```


Implementierung

Eventpublisher: MessageBusClient

```
public class MessageBusClient : IMessageBusClient {
    private readonly IConnection _connection;
    private readonly IModel _channel;
    private readonly string _exchange;

    public MessageBusClient(IConfiguration configuration){
            ...
            try{
                _connection = factory.CreateConnection();
                _channel = _connection.CreateModel();
                _channel.ExchangeDeclare(
                    _exchange, ExchangeType.Fanout
                );
                ...
            }catch(Exception ex){
                Console.WriteLine($"failed: {ex}");
            }
    }
```

Implementierung

Eventpublisher: MessageBusClient

```
public class AMessageBusClient : IMessageBusClient {
   private readonly IConnection _connection;
   private readonly IModel _channel;
   public MessageBusClient(IConfiguration configuration){
         ...
         try{
            ...
            _connection.ConnectionShutdown +=
                ShutDownMessageBroker;
            ...
   }

   private void ShutDownMessageBroker(
      object sender, ShutdownEventArgs arg
   ) => Console.WriteLine("Eventbus Shutdown");
}
```


Implementierung

Eventpublisher: MessageBusClient

```
using System.Text.Json;

public class MessageBusClient : IMessageBusClient {
   private readonly string _exchange;
   public AMessageBusClient(IConfiguration
       configuration){...}


   public void Publish(string message){
      var body = Encoding.UTF8.GetBytes(message);
      _channel.BasicPublish(_exchange, "", null, body);
   }
}
```


Implementierung

Eventpublisher: MessageBusClient

```
public class MessageBusClient : IMessageBusClient {
   private readonly IModel _channel;
   private readonly IConnection _connection;

   public AMessageBusClient(IConfiguration
       configuration){...}
   public void Publish(string message){...}

   public void Dispose(){
      if(_channel.IsOpen()){
         _channel.Close();
         _connection.Close();
      }
   }
}
```


Implementierung

Eventpublisher: Program

```
public class Program {
    ...
    var builder = WebApplication.CreateBuilder(args);
    builder.Services.AddSingleton<
        IMessageBusClient, MessageBusClient
    >();
    ...
}
```


Implementierung

Eventpublisher: Controller

```
public abstract class Controller ... {
      private readonly IMessageBusClient _eventBusClient;

      public Controller(
         IMessageBusClient eventBusClient
      ){
         this._eventBusClient = eventBusClient;
      }

}
```


Implementierung

Eventpublisher: Controller

```
using System.Text.Json;

public abstract class Controller<Dto> ... {
      private readonly IMessageBusClient _eventBusClient;
      public Controller(...){...}

      private void SendEvent(Dto data){
            var message = JsonSerializer.Serialize(data);
            _eventBusClient.Publish(message);
      }

}
```


Implementierung

Eventsubscriber: EventSubscriber

```
using Microsoft.Extensions.Hosting ;
public class EventSubscriber : BackgroundService {
   private readonly IConfiguration _configuration;
   private readonly IConnectin _connection;
   private readonly IEventProcessor _eventProcessor;

   public EventSubscriber(
      IConfiguration configuration,
      IEventProcessor processor
   ){
      _cofiguration = configuration;
      _eventProcessor = processor;
      Init();
      ...
   }
}
```


Implementierung

Eventsubscriber: EventSubscriber

```
public class EventSubscriber : BackgroundService {
   private readonly IModel _channel;
   public EventSubscriber(...){...Init();...}

   private void Init(){
      var factory = new ConnectionFactory(){
         HostName = _configuration["RabbitMQHost"],
         Port = int.Parse(_configuration["RabbitMQPort"])
      }
      _connection = factory.CreateConnection();
      _exchange = _configuration["EventBusExchange"];
      _channel = _connection.CreateModel();

      );
   }
}
```


Implementierung

Eventsubscriber: EventSubscriber

```
public class EventSubscriber : BackgroundService {
   private readonly IModel _channel;
   public EventSubscriber(...){...Init();...}

   private void Init(){
      ...
      _channel.ExchangeDeclare(_exchange,
          ExchangeType.Fanout);
      _queueName = _channel.QueueDeclare().QueueueName;
      _channel.QueueBind(_queueName, _exchange, "");
   }
}
```


Implementierung

Eventsubscriber: EventSubscriber

```
public class EventSubscriber : BackgroundService {
   protected override Task
       ExecuteAsync(CancelleationToken token){
         stoppingToken.ThrowIfCancellationRequested();

         var consumer = new
             EventingBasicConsumer(_channel);
         consumer.Received += (ModuleHandle, ea) => {
            var body = ea.Body;
            var message =
                Encoding:UTF8.GetString(body.ToArray());

            eventProcessor.ProcessEvent(message);
            ...
         };
   }
}
```


Implementierung

Eventsubscriber: EventSubscriber

```
public class EventSubscriber : BackgroundService {
   protected override Task
       ExecuteAsync(CancelleationToken token){
      ...
      _channel.BasicConsume(_queueName, true, consumer);
      return Task.CompletedTask;
   }
}
```


Implementierung

Eventsubscriber: Programm

```
public class Program {
    ...
    services.AddHostedService<EventSubscriber>();

}
```


Implementierung

EventProcessor: IEventProcessor

Register EventProcessor as Singelton

```
public interface IEventPorcessor{
   void ProcessEvent(string message);
}
```


Implementierung

EventProcessor: EventProcessor

```
using Micorsoft.Extensins.DependencyInjection;
public class EventProcessor : IEventProcessor {
   private readonly IServiceScopeFactory _scopeFactory;
   public EventProcessor(
      IServiceScopeFactory scopeFactory
   ){
      this._scopeFactory = scopeFactory;
   }
}
enum Eventype {
   Undetermend, CreatedProject, UpdatedProject;
}
```


Implementierung

EventProcessor: EventProcessor

```
public class EventProcessor : IEventProcessor {
   private readonly IServiceScopeFactory _scopeFactory;
   public EventProcessor(...){...}

   private EventType DetermineEvent(string message){
      var event =
          JsonSerializer.Deserilaize<Dto>(message);
      EventType eventType;

      return Enum.TryParse<EventType>(event.type, out
          eventType);
   }
}
enum Eventype {
   Undetermend, CreatedProject, UpdatedProject;
}
```


Implementierung

EventProcessor: EventProcessor

```
public class EventProcessor : IEventProcessor {
   public EventProcessor(...){...}
   public void ProcessEvent(string message){
      var eventType = DetermineEvent(message);
      // Execute Event
   }
}
```


Implementierung

EventProcessor: EventProcessor

```
public class EventProcessor : IEventProcessor {
    public EventProcessor(...){...}
    public void ProcessEvent(string message){
        var eventType = DetermineEvent(message);
        // Init Event
        // Execute Event
    }
}
```


Implementierung

EventProcessor: EventProcessor

```
public class EventProcessor : IEventProcessor {
   public EventProcessor(...){...}
   public void ProcessEvent(string message){
      // Init Event
      using(var scope = _scopeFactory.CreateScope()){
         var repo = scope.ServiceProvider
            .GetRequiredServie<IRepository<Alora>>();
         var dataDTO =
             JsonSerializer.Deserialize<...>(message)
         var entity = Map(dataDTO);
         repo.Method(entity);
      }
   }
}
```


Implementierung

EventProcessor: IEventProcessor

```
pub
```


Implementierung

Configuration

- Nuggets: `RabbitMQ.Client`