От сервера – клиенту, или Как это работает в другую сторону

By Vladimir Zarubin

Зачем?

Как уведомить клиента о произошедшем событии

Как предоставить клиенту своевременные данные

Как поддержать интерактивность в вашем приложении

Что хотим от клиента?

События

Клиент должен быть уведомлен о событиях, произошедших в системе

Событие требует вмешательства

Лайв-мониторинг активности

Что хотим от клиента?

Данные

Актуальность информации – путь к успеху

Об изменениях должны знать

Обработка данных – цель клиента

Что хотим от клиента?

Команды

Обратное исполнение, или RPC

Контроль и обратная связь

Распределенные процессы

Модели доставки сообщений

Pull/Imperative

Классическая модель запрос-ответ

Клиент спрашивает – сервер отвечает

Запрос инициируется в необходимое клиенту время

Push/Reactive

Модель оповещения

Сервер/прокси оповещает **подписчика** о новой доступной информации

Push-нотификация приходит клиенту по решению сервера

Модели доставки сообщений

Pull/Imperative

Polling

Long Polling

Push/Reactive

Signal R

Owin Push

Message Queue

Polling

Timers/Loops

Опрос по времени или циклу. Каждые *n* секунд делаем Get/Post и получаем текущее состояние

Scheduling

Опрос по расписанию. По бизнесу данные появляются или должны обновляться со строгой периодичностью

Polling

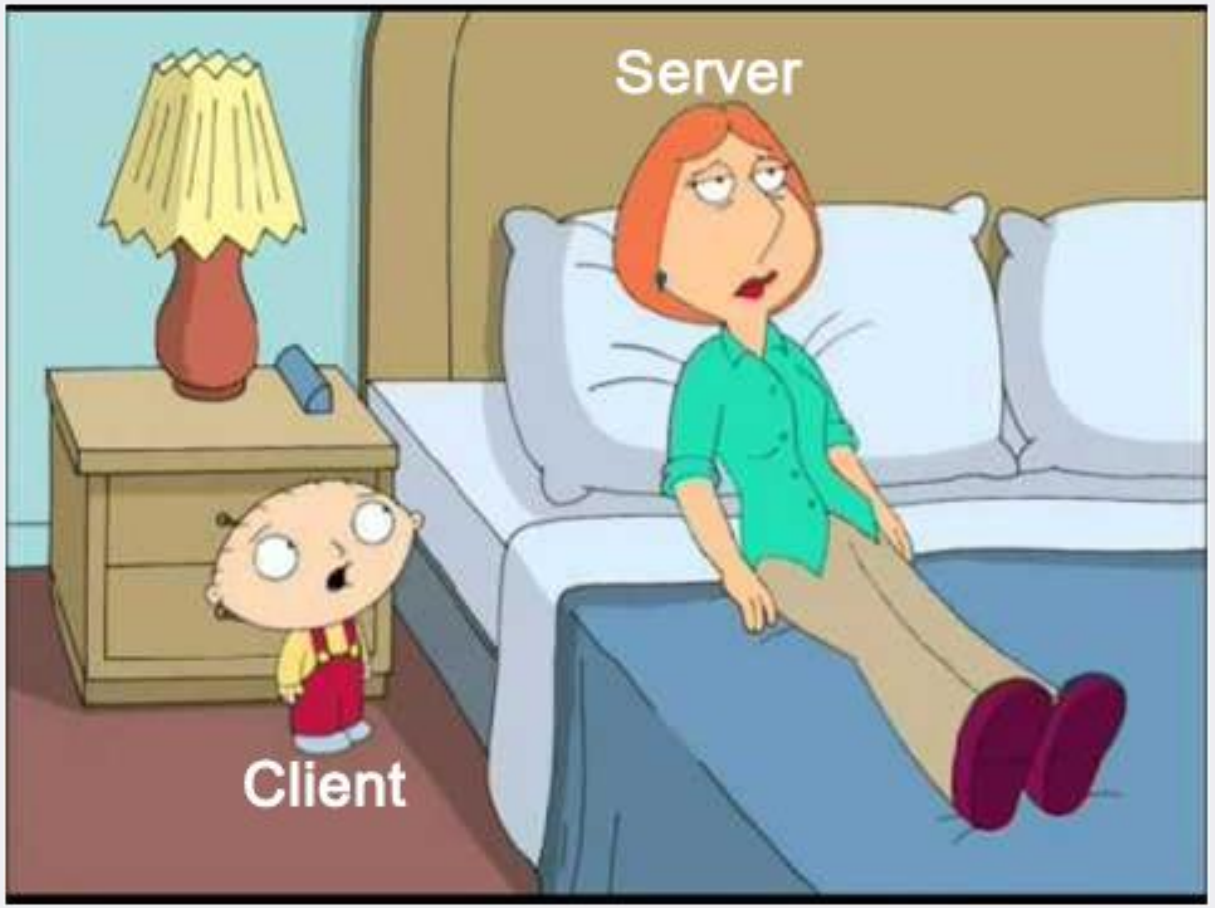
Event/Navigation Base

При загрузке новой страницы/блока/окна получаем состояние сервера

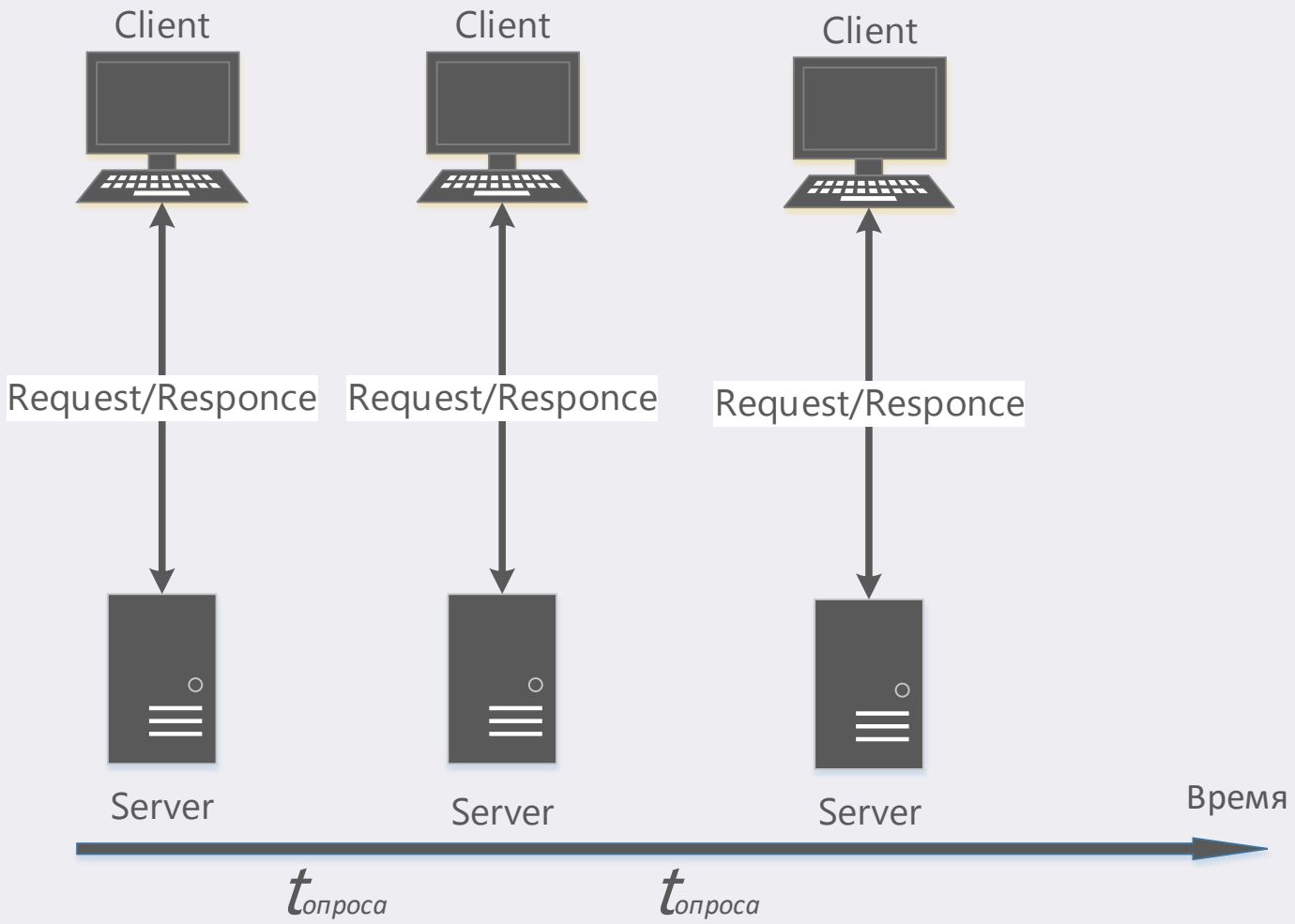
Manual

Нажми на кнопку – увидишь результат. Старый и проверенный refresh/F5

Polling



Polling


Client
Client
Client
Request/Responce
Request/Responce
Request/Responce
Server
Server
Server
Время
$t_{опроса}$
$t_{опроса}$

Polling

Плюсы

- Простой и наглядный флоу response/request
- Достаточно Http Client
- Реализуем независимо от сервера/клиента

Минусы

- Дискретизация состояния
- «Паразитная» нагрузка сервера (Request -> Process -> Sql -> Response)
- Нет единой точки входа
- Состояние клиента сложно мониторить
- Сложно разделять состояния при их наличии

Polling C#

```
Observable.Interval(TimeSpan.FromSeconds(5))
    .Subscribe(async (l) =>
    {
        string currentTemp = await "http://localhost:9650/temp/current".GetStringAsync();

        Console.WriteLine(currentTemp);
    });

Console.ReadLine();
```

Polling JS

```
<script type="text/javascript">
    function RequestData()
    {
        fetch('http://localhost:9650/temp/current')
            .then(function(response) {
                                    return response.text();
            })
            .then(function(val) {
                document.getElementById("tempHolder").innerText = val;
            })
    }

    setInterval(RequestData, 4*1000);
</script>
```

Long Polling

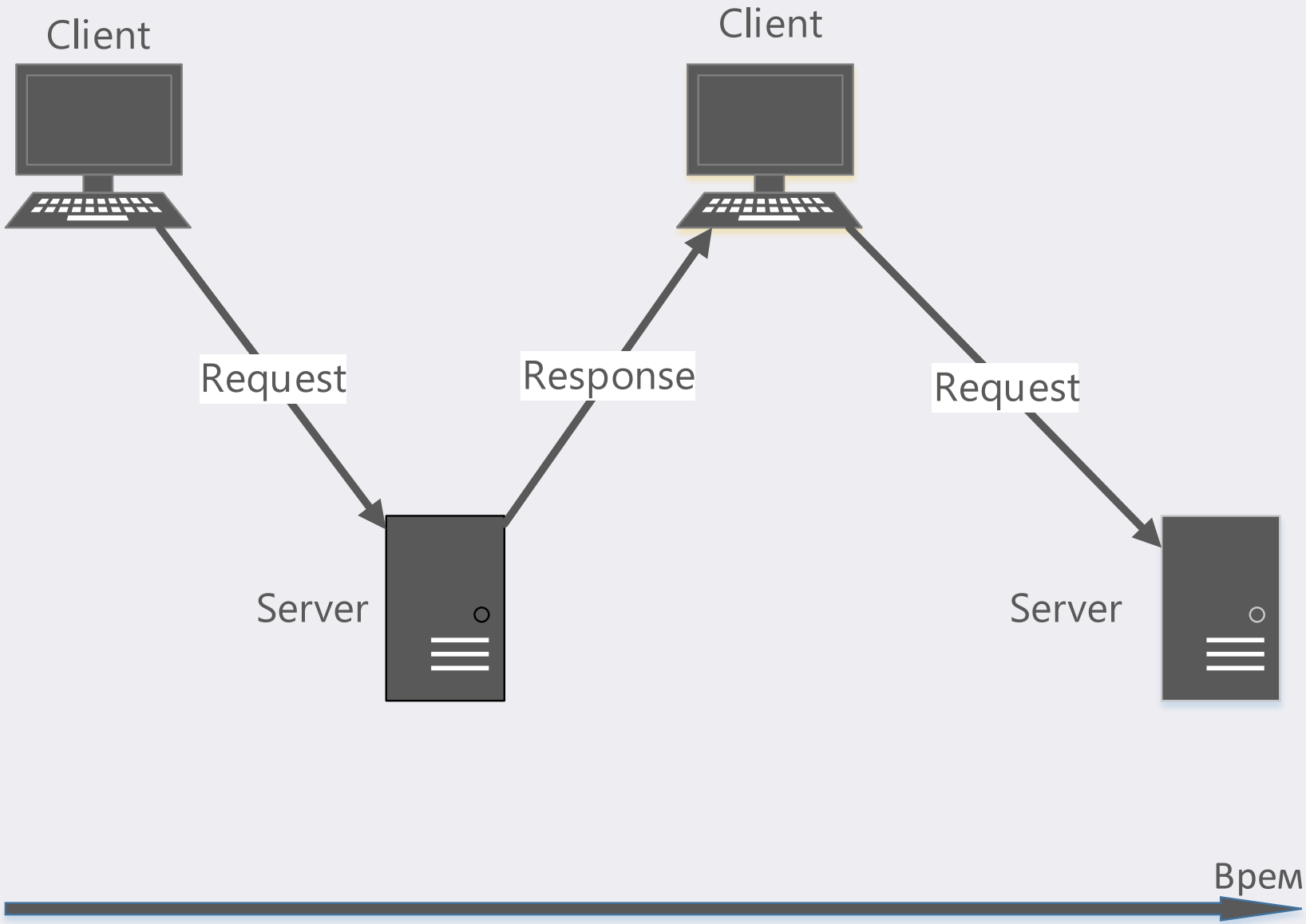
Клиент отправляет запрос вида Get/Post с большим или отсутствующим таймаутом

Сервер сохраняет HttpContext и оставляет соединение открытым

После наступления ожидаемого события – ответ клиенту

Клиент после принятия ответа повторяет запрос

Long Polling



Long Polling

Плюсы

- Те же что и pooling
- При «редких» событиях хорошо применим

Минусы

- При частом возникновении события – становится pooling
- Открытые висящие соединения

Long Polling C#

```
while (true)
{
    if (toker.IsCancellationRequested) break;

    string currentTemp = await "http://localhost:9700/temp/GetTemp".GetStringAsync();

    Console.WriteLine(currentTemp);
}
```

Long Polling JS

```
<script type="text/javascript">
    function RequestData()
    {
        fetch('http://localhost:9700/temp/gettemp')
            .then(function(response) {
                                    return response.text();
            })
            .then(function(val) {
                document.getElementById("tempHolder").innerText = val;
                return true;
            })
            .then(function (value) {
                RequestData();
            });
    }
    RequestData();
</script>
```

Push OWIN, или Немного нестандартно

Application as Service

Desktop-приложение предоставляет канал связи в виде OWIN Self Host Rest сервиса с уникальным адресом

Rest реализует контракт push-уведомлений

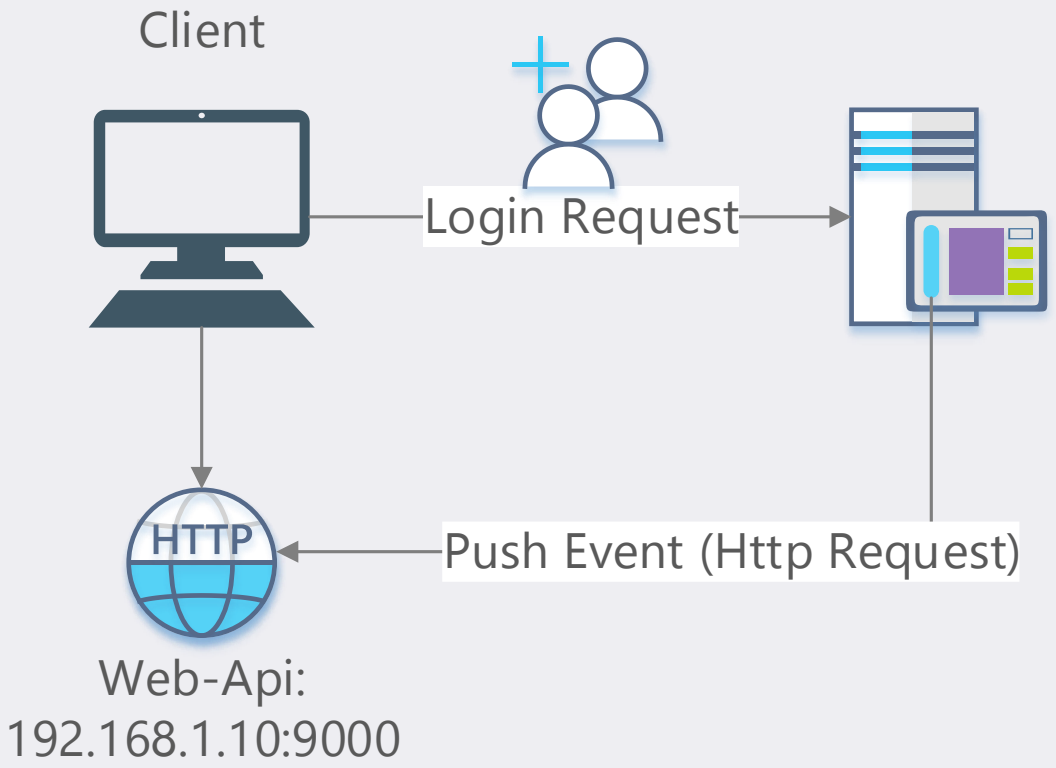
Rest принимает push-сообщения и прокидывает дальше в приложение (Pub/Sub, Event Bus, Rx)

Server as Request Client

При регистрации/логине клиента серверная часть «запоминает» клиента

При возникновении события сервер посылает Rest-запрос по «адресу» клиента

Push OWIN


Client
Login Request
HTTP
Push Event (Http Request)
Web-Api:
192.168.1.10:9000

Push OWIN

Owin Self Host

Owin совместо с Self-Host позволяют Windows-приложению реализовывать веб-сервис внутри запускаемого файла

Asp.Net Web API как вариант реализации контракта

Можно использовать любые расширения – Cors, SignalR and etc

Swagger

Для упрощения, документирования и тестов – SwashBuckle

Unit-Testing

Microsoft.Owin.Testing – юнит-тесты для вашего API

Поддерживаемые платформы

Console, WinForms, WPF

.Net Core - Console

Since .Net Core 3 – WPF, WinForms

Push OWIN

Плюсы

- Рест везде
- Просто и быстро
- Хорошо смотрится в распределенных системах
- Не зависит от бэкэнда (Java, Python, NodeJs)
- Все плюсы веб-сервисов
- Уменьшение нагрузки сервера + возможно создавать гибридные приложения

Минусы

- Только настольные приложения
- Проблемы с доступом ~~к джойказино~~ и правами (netsh http add urlacl)
- Обмен контрактов (кто главнее) и/или двойной код запросов
- Сложно синхронизировать

Push OWIN

```
public void Configuration(IAppBuilder appBuilder)
{
    var config = new HttpConfiguration();

    config.MapHttpAttributeRoutes();


    config.Routes.MapHttpRoute("Home", "{controller}/{action}");

    config
        .EnableSwagger(c => c.SingleApiVersion("v1", "PushOwinSwagger"))
        .EnableSwaggerUi();


    appBuilder.UseCors(CorsOptions.AllowAll);

    appBuilder.UseWebApi(config);

    config.EnsureInitialized();
}
```

Push OWIN

```
using (WebApp.Start<StartUp>(hostAddress))
{
    "http://localhost:9800/reg/subscribe".PostJsonAsync(new PushRegistration()
    {
        IpAddress = hostAddress,
        ClientId = Guid.NewGuid()
    });


    Console.WriteLine("Press to exit");
    Console.ReadKey();
}
```

Push OWIN

```
[HttpPost]
public IHttpActionResult Current(TempChange currentTemp)
{
    Console.WriteLine($"Current temp is {currentTemp.TemperatureInC}C ({currentTemp.TemperatureInF} F)");

    return Ok();
}
```

```
public void NotifyClients(int temp)
{
    foreach (var sub in subscriberHolder.GetSubscribers)
        $"{sub}/temp/current".PostJsonAsync(new TempChange(temp));
}
```

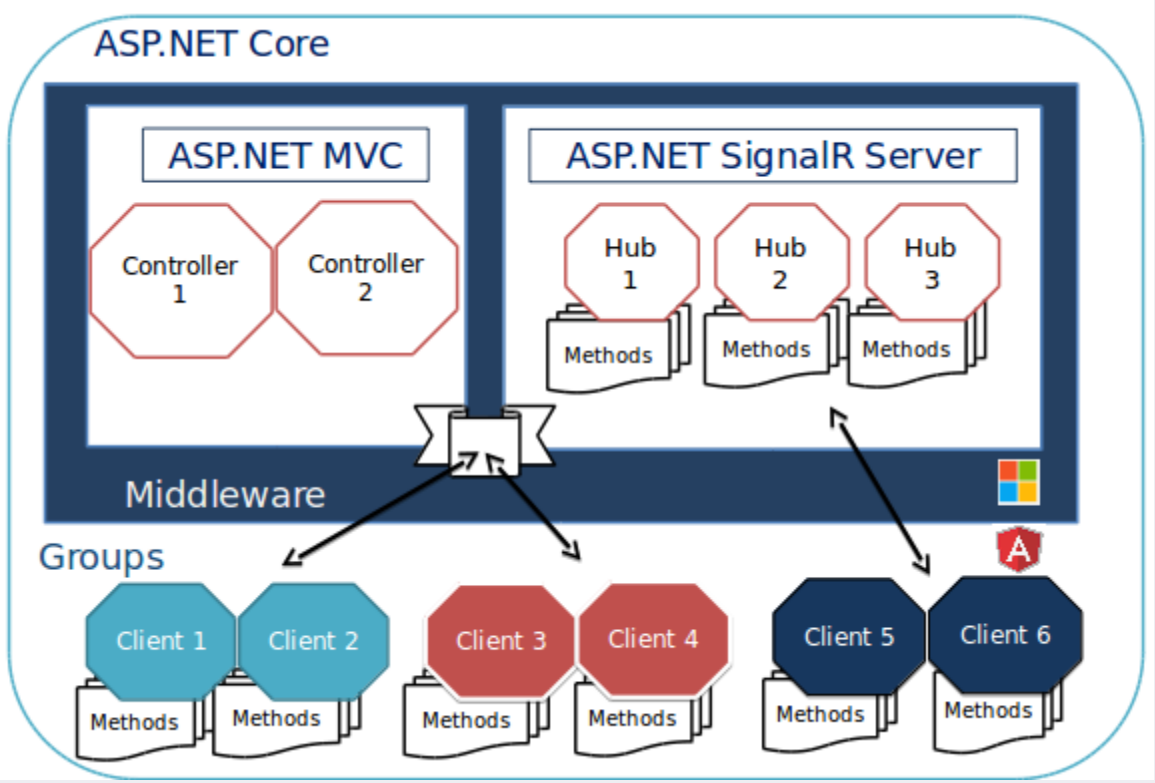
Signal R

SignalR – расширение функциональности ASP.net для поддержки механизма нотификации клиента

Реализует Push Model

Позволяет строить Real Time Application

Поддержка Server-To-Client RPC



Signal R

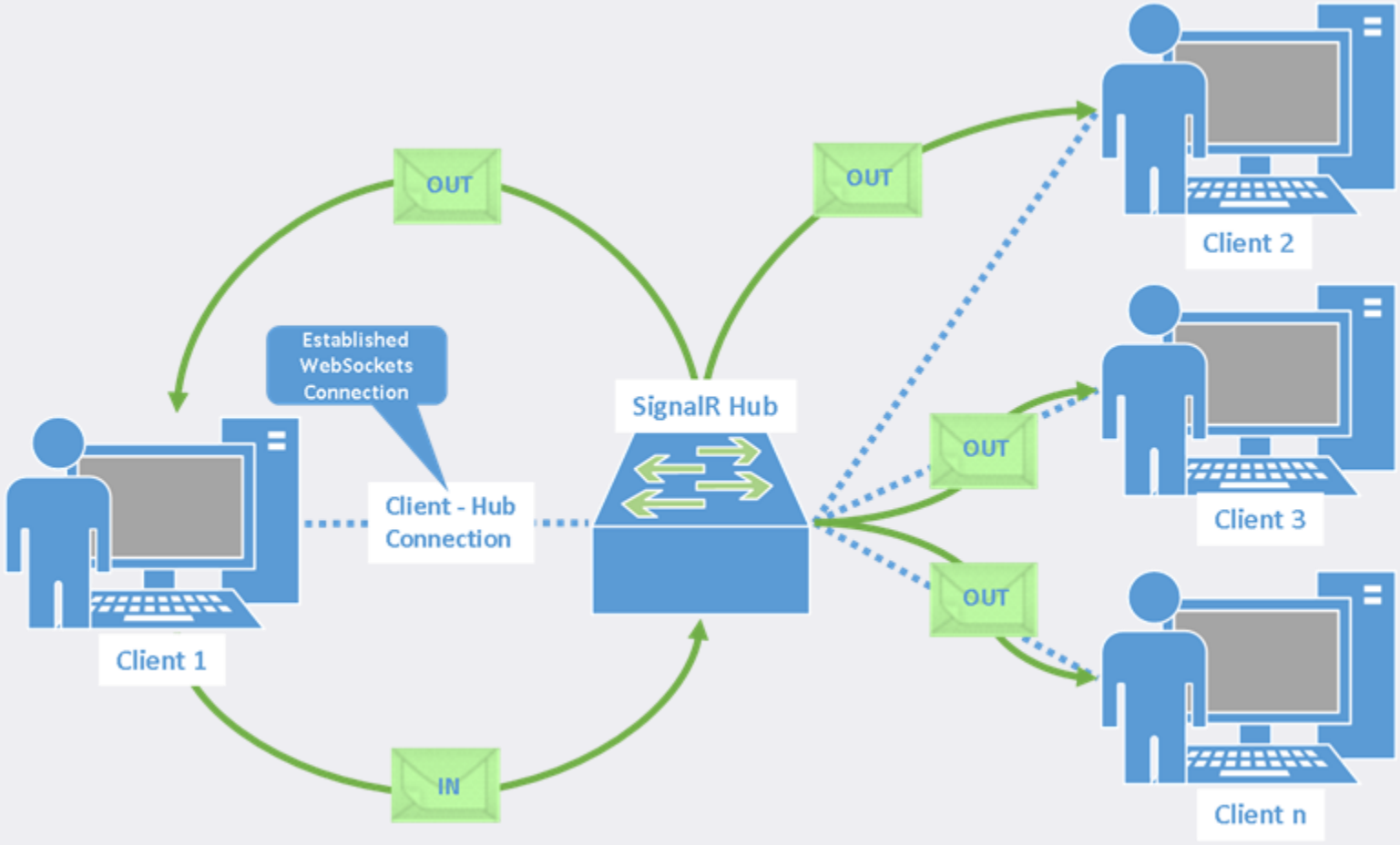
Сервер

Встроен в Asp.net и Asp.Net Core
Все плюшки Asp.Net (OWIN, CORS…)
Масштабируемость (SQL, Redis, Azure Service Bus)
Нативная адресация клиентов (модели Peer to peer, BroadCast, All except me)
Один канал для общения с клиентом
Возможность деплоя в AZURE

Клиент

Поддержка разных платформ – от браузеров до десктопов (Windows Desktop, A lot of browsers, Java Client)
Готовые библиотеки для использования
Вариативность транспорта (WebSockets, Server Sent Events, Forever Frame, long polling)
Мониторинг состояния подключения

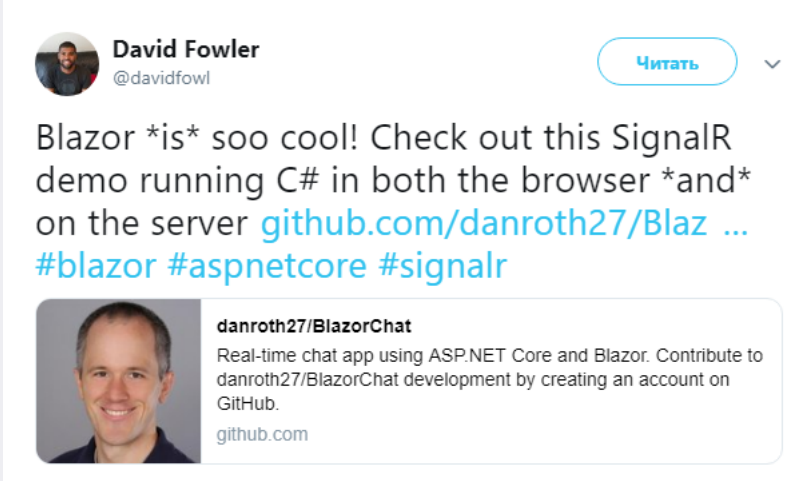
Signal R


OUT
OUT
Established
WebSockets
Connection
SignalR Hub
Client 2
OUT
Client 3
Client - Hub
Connection
OUT
Client 1
IN
Client n

Signal R

Плюсы:
Открытый код и развитие технологии
Много фич и плюшек (Loading Percent, Streams)
Тонна документации и примеров
Нативно и .Net
Поддержка современных технологий

Минусы:
Нативно и .Net
Разрастается при масштабировании



Signal R

```
var hubConnection = new HubConnection("http://localhost:9200");

IHubProxy stockTickerHubProxy = hubConnection.CreateHubProxy("TempHub");

stockTickerHubProxy.On<int>("CurrentTemp", temp => Console.WriteLine($"Temp is {temp}"));

hubConnection.Start().Wait();

Console.WriteLine("Signal R Client started");
```

Signal R

```
public void Configuration(IAppBuilder appBuilder)
{
    var config = new HttpConfiguration();

    config.MapHttpAttributeRoutes();

    config.Routes.MapHttpRoute("Home", "{controller}/{action}");

    appBuilder.UseCors(CorsOptions.AllowAll);
    appBuilder.MapSignalR();

    appBuilder.UseWebApi(config);

    config.EnsureInitialized();
}
```

Signal R

```
public class TempHub : Hub<IClient>
{
    private static int temp = 25;

    public void Up()
    {
        temp++;
        Clients.All.CurrentTemp(temp);
    }

    public void Down()
    {
        temp--;
        Clients.All.CurrentTemp(temp);
    }
}
```

Message Queue

При наличии инфраструктуры и возможности надлежащего развертывания

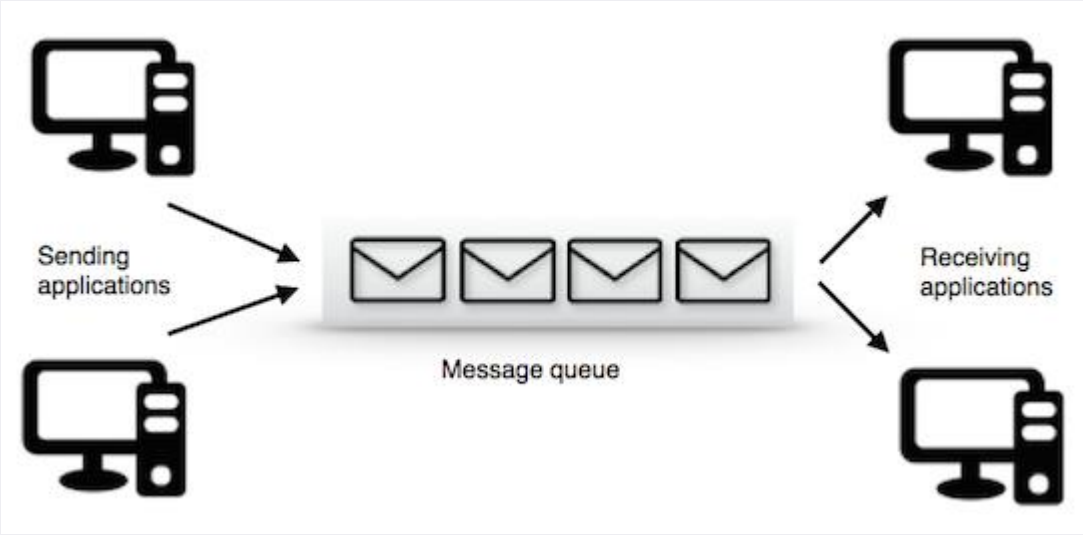
Message Queue как канал обмена между клиентом и сервером

Producer/Consumer – отличный паттерн для общения между клиентом и сервером

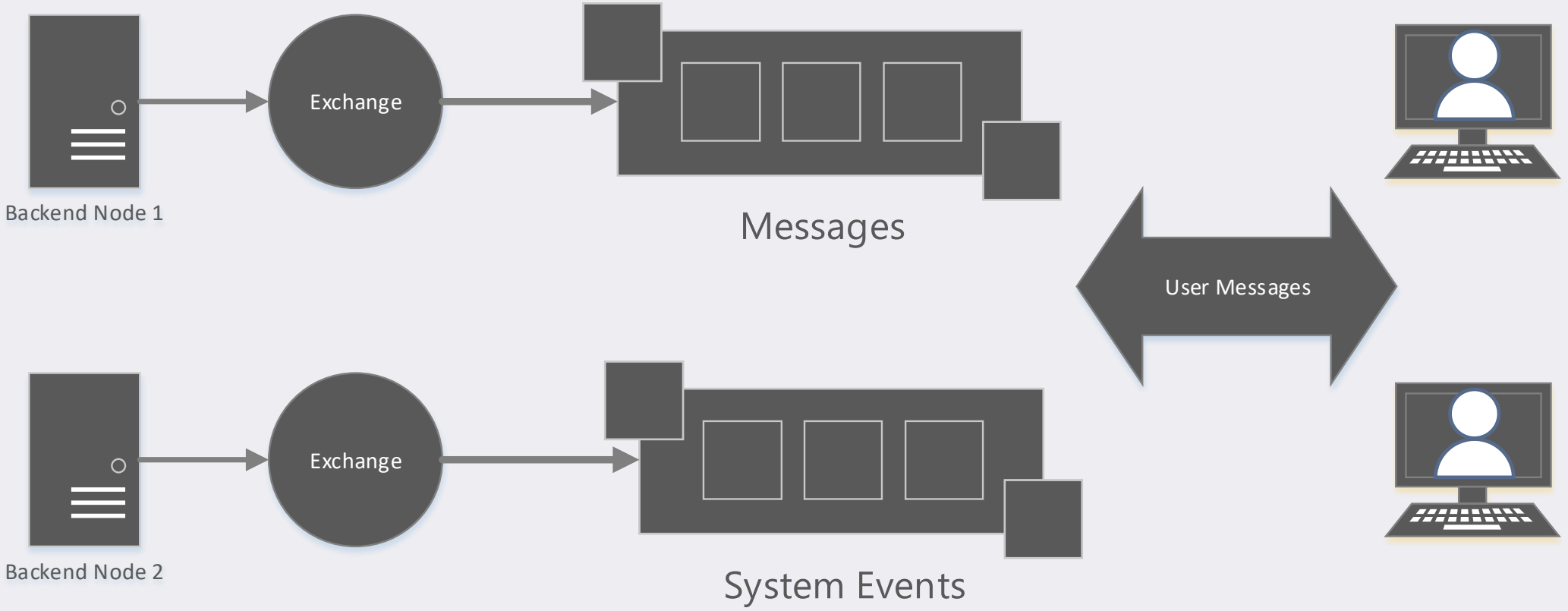
Сервер (Producer) генерирует сообщения на основе бизнес-логики

Message Queue – транспорт и доставка сообщений

Клиент (Consumer) получает сообщения (лично или broadcast) и обрабатывает на UI



Message Queue



Message Queue

Плюсы:

Множество моделей обмена сообщений
Гибкость подхода и расширения
Гарантированная доставка
Дополнительные возможности для тестирования и отслеживания процессов
Не зависим от платформы клиента и сервера (есть варианты даже для JS)
Возможность совмещения с инфраструктурой
Подходит для высоконагружженых систем
Отказоустойчивость

Минусы:

Требования к контуру работы
Дополнительные узлы инфраструктуры при развертывании

Message Queue

```
using (var channel = connection.CreateModel())
{
    channel.QueueDeclare("tempQueue",
        false,
        false,
        false,
        null);

    var consumer = new EventingBasicConsumer(channel);

    consumer.Received += (model, ea) =>
    {
        var body = ea.Body;
        var message = Encoding.UTF8.GetString(body);
        Console.WriteLine("Current Temp is {0}", message);
    };

    channel.BasicConsume("tempQueue",
        true,
        consumer);
}
```

Message Queue

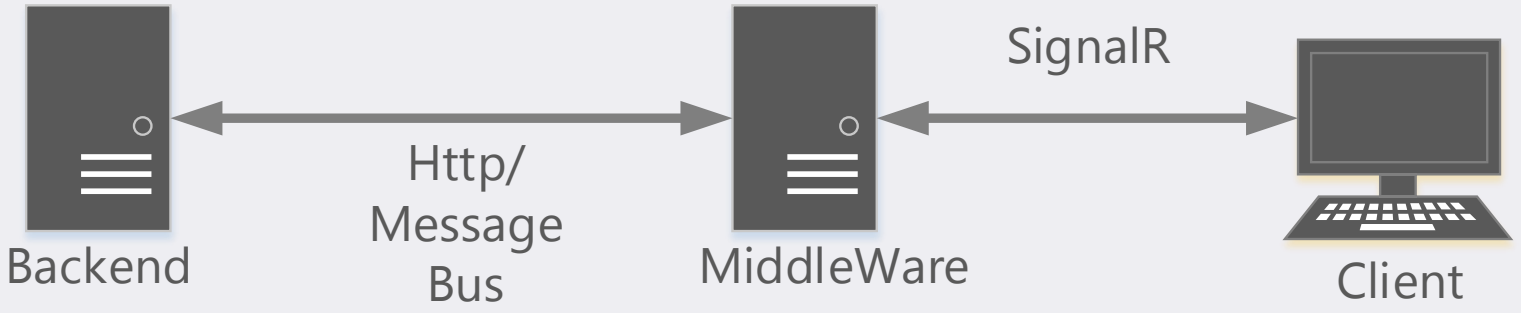
```
var factory = new ConnectionFactory { HostName = "localhost" };

using (IConnection connection = factory.CreateConnection())
{
    using (IModel channel = connection.CreateModel())
    {
        channel.QueueDeclare("tempQueue", false, false, false, null);

        byte[] body = Encoding.UTF8.GetBytes(message);

        channel.BasicPublish("", "tempQueue", null, body);

        Console.WriteLine(" [x] Sent {0}", message);
    }
}
```

Комбинируйте

Единого подхода не существует, есть лишь Best Practices

Все зависит от окружения и задач

Интеграции никто не отменял



На этом все

Владимир Зарубин

Luxoft – Omsk – Glonass

Skype: vzarubinluxoft@gmail.com
Telegram: @faust69
E-mail: Vzarubin@luxoft.com
69Voland69@gmail.com
Github: VAZarubin