Единая точка входа или точка отказа. Путь к non-blocking API Gateway.

Никита Летов

Росбанк

@nikita_letov

Technixc.n.l@gmail.com

JPoint

росбанк

Single point of entry or point of failure. Non-blocking API Gateway Path.

Nikita Letov

Rosbank

JPoint

росбанк

Nikita Letov

TechniXC

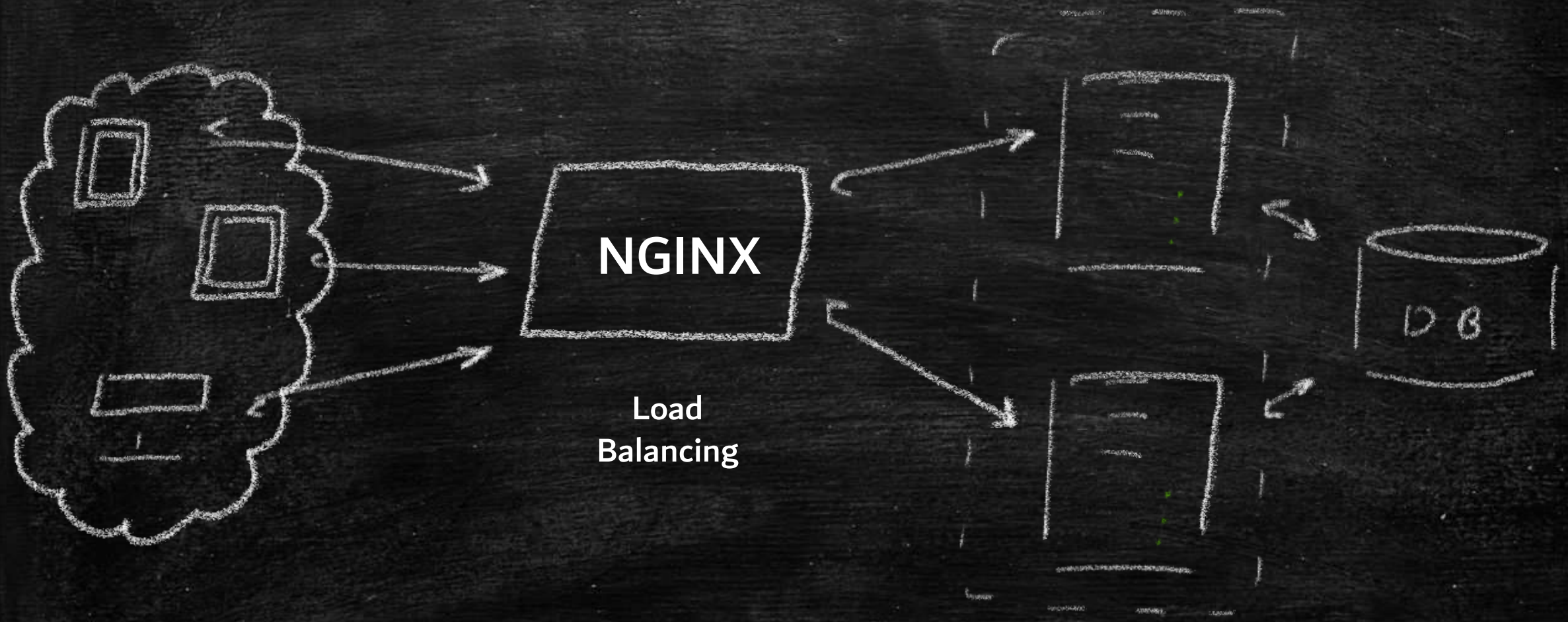
Bio

- Старший Java разработчик и Тимлид бекенд разработки платформы ДБО в Росбанке
- Инфобез в прошлом

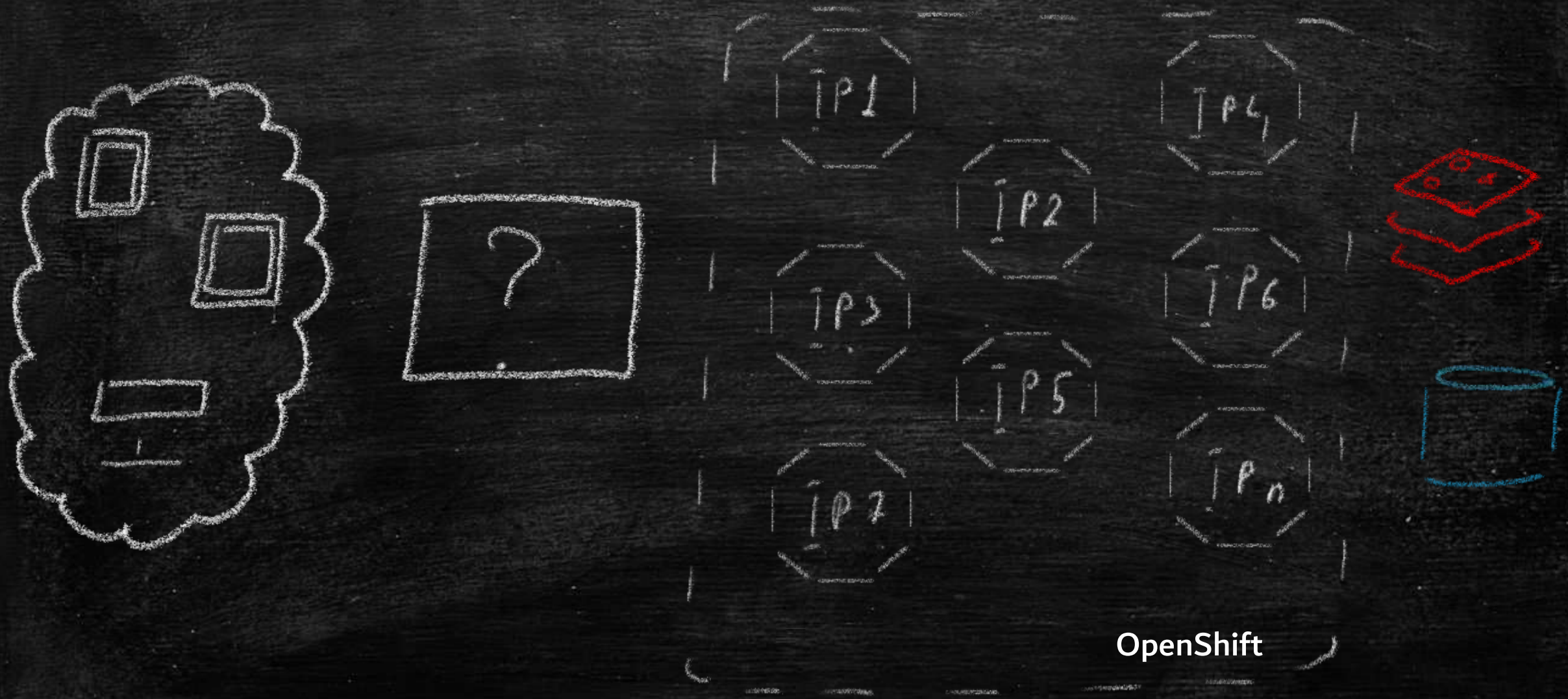
О чем поговорим?

- Для чего нужная входная точка в приложение?
- Когда появляется необходимость в единой точка входа
- Задачи решаемые API Gateway
- Рассмотрим классический блокирующий подход на примере Zuul 1.X
- Разберем проблемы, с которыми можно столкнуться при разработке и эксплуатации блокирующего Gateway
- Рассмотрим реактивный, Non-blocking Spring Cloud Gateway и возможные сложности перехода на него
- Сравним оба подхода, оценим профиты

Как все обычно начинается…



Когда появляется необходимость во входной точке



Монолит разбили, а как теперь все собрать? (Мем)

- Будем продолжать использовать Nginx или какое-то сетевое решение типа NetScaler / F5. (С кучей конфигураций и постоянной ругани с сетевым отделом)
- Каждому микро-сервису свой поддомен и выставляем наружу. xD
- Используем Edge (пограничный) сервис для решения этой задачи.

– Привет API Gateway!

Что вообще из себя представляет API Gateway

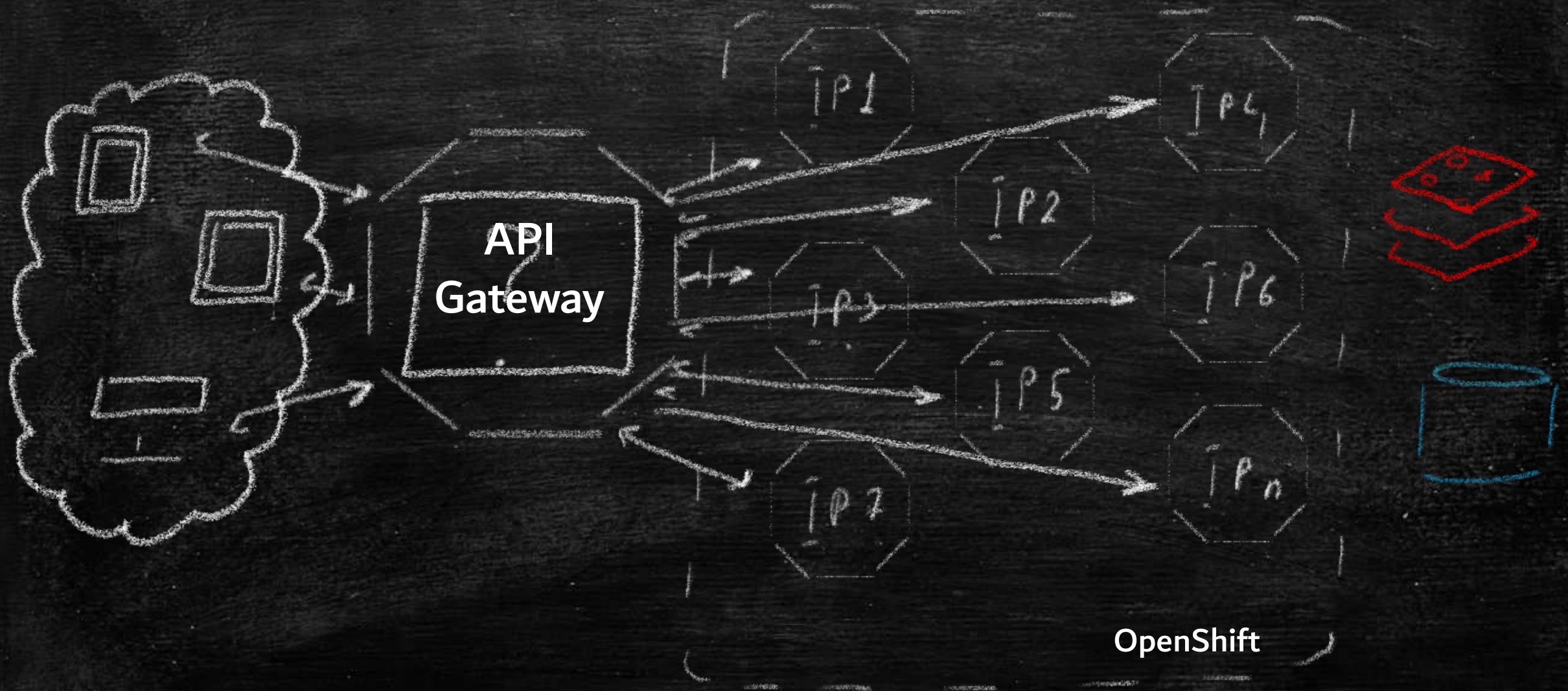
Если бы HTTP запросы летали самолетами

Задачи решаемые Gateway

- Унификация – один интерфейс для всех сервисов
- Маршрутизация – перенаправление запросов на целевые сервисы
- Валидация – проверка наличия обязательных параметров
- Авторизация и Аутентификация
- Трассировка запросов / ответов
- Стандартизация – обрезка лишнего
- Обогащение запросов / ответов – добавление Headers или cookie

- Метрики (Observability)
- Shaping / Throttling
- Отказоустойчивость

Когда появляется необходимость во входной точке



Задачи поставленные перед Gateway

- Маршрутизация клиентских запросов в сервисы – Gateway должен маршрутизировать запросы в сервисы разных версий в зависимости от версии клиентского приложения.
- Gateway должен проверять авторизацию клиентов, выступать в роли Oauth2.0 (OpenID) Resource Server, а в API сервисы уже отдавать информацию о пользователе
- Трассировка запросов/ответов, gateway должен иметь возможность вешать на каждый запрос/ответ “ярлык” с уникальным идентификатором.
- Необходимо логирование параметров запроса и ответа для того, что бы можно было качественно разбирать возможные проблемы.
- Необходимо, что бы Gateway отдавал метрики для того, чтобы оценивать его загрузку и работоспособность
- Необходим функционал для костюмного обогащения запросов в виде добавления cookie, headers или же подмена значений в body запросов – ответов

Выбор технологии (Вернемся в прошлое)

Spring Cloud Netflix Zuul

- Актуальное решение на стеке Spring + Java
- Интеграция с Spring Boot и Cloud
- Обширное Community
- Классический подход к разработке
- Используется многими крупными компаниями в том числе самим Netflix!

NETFLIX
ZUUL

Spring Cloud Netflix. Zuul.

- Spring Cloud Netflix Zuul является частью проекта Spring Cloud Netflix, представляет собой API Gateway, который обеспечивает динамическое маршрутизирование, мониторинг, безопасность и другие кросс-функциональные возможности для микросервисных приложений.

- Zuul поддерживает фильтры на разных этапах обработки запросов, таких как пред-фильтры (pre), пост-фильтры (post) и фильтры обработки исключений (error).

- Основан на стеке сервелетов, использует Apache Tomcat, и в своей работе полностью полагается на парадигму – one-request-per-thread, для обмена информацией между фильтрами используется RequestContext.

Servlet Context

- Контекст сервлетов предоставляет среду, в которой сервлеты выполняются внутри контейнера сервлетов - в том же Apache Tomcat, а так же доступ к общим ресурсам

- При разработке Spring MVC приложения, мы не используем ServletContext напрямую, а полагаемся на абстракции предоставляемые Spring, такие как контроллеры и аннотации.

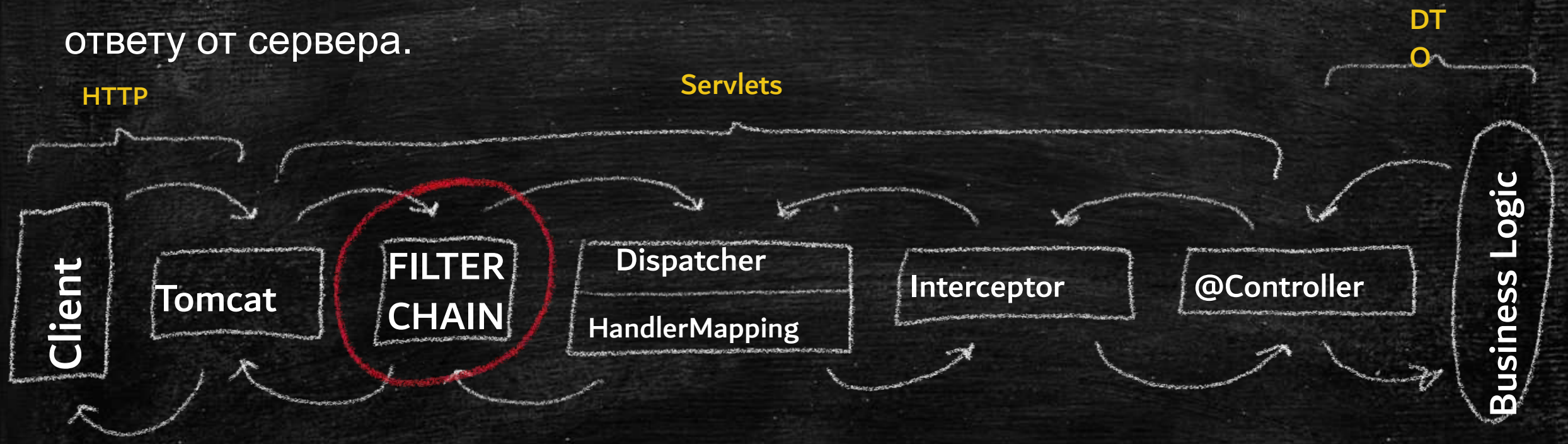
Абстракции Spring

```
@RestController

public class MyController {


    @GetMapping("/hello")

    public String hello() {

        return "Hello,
World!";

    }

}
```

```
@GetMapping("/greet")

public String greet(@RequestParam String name)
{

    return "Hello, " + name + "!";

}
```

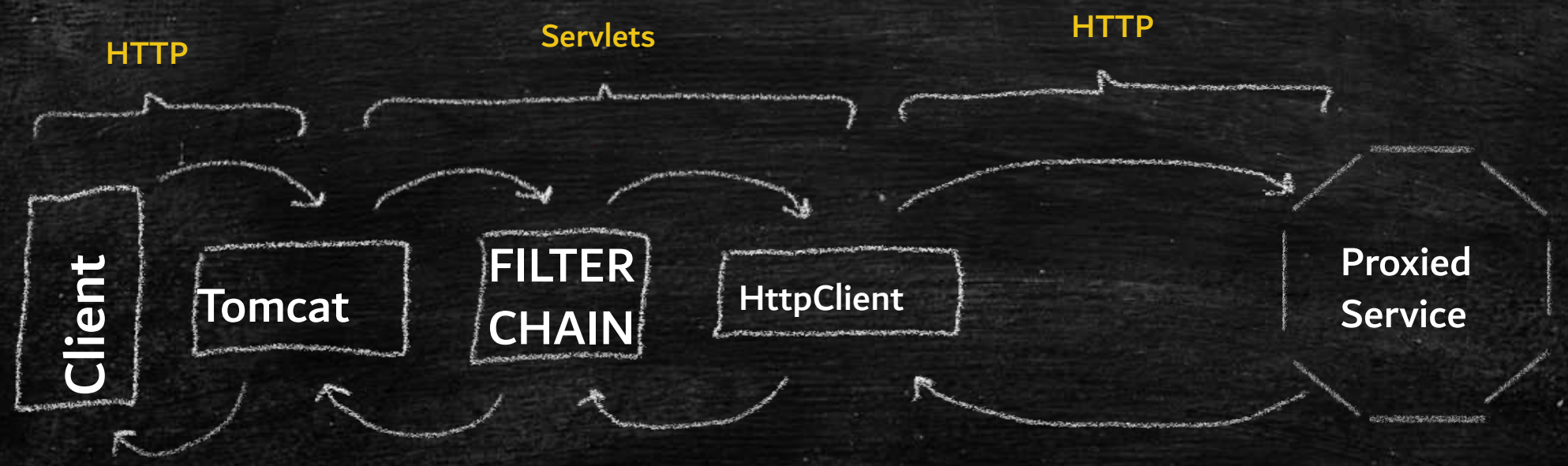
Servlet Container. Путь HTTP запроса

- HttpServletRequest / HttpServletResponse это объекты, которые для нас создает контейнер сервлетов – Apache Tomcat. Они соответствуют HTTP запросу и ответу от сервера.



Servlet Container. Севрлеты.

- Так как API Gateway не имеет своих контроллеров, после прохождения цепочки фильтров происходит маршрутизация в проксируемый сервис.



Как Zuul работает с сервлетами.

- Zuul не работает с сервлетами напрямую
- Zuul для обмена информацией между фильтрами использует потокобезопасный RequestContext.
- RequestContext заполняется данными из HttpServletRequest, HttpServletResponse и ServletContext

Фильтры

- Фильтр - это компонент, который может быть использован для обработки входящих и исходящих HTTP-запросов и ответов перед тем, как они достигнут конечных точек контроллеров сервисов или вернутся клиенту.
- Zuul по факту весь написан на ZuulFilter
- Так как Zuul интегрирован в Spring Boot –для обработки запросов мы можем использовать javax.servlet.Filter, а так же фильтры из пакета o.s.web.filter такие как GenericFilterBean или OncePerRequestFilter

ZuulFilter

```
public class SampleZuulFilter extends ZuulFilter {
    @Override
    public String filterType() {
        return FilterConstants.PRE_TYPE;
    }

    @Override
    public int filterOrder() {
        return PRE_DECORATION_FILTER_ORDER + 1;
    }

    @Override
    public boolean shouldFilter() {
        return true;
    }

    @Override
    public Object run() {
        log.info("Logic executed here");
        return null;
    }
}
```

Тип (PRE/POST/ERROR)

Порядок

Флаг включения фильтра

Логика фильтра.

игнорируется!

Filter / GenericFilterBean

```
public class SampleFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
            throws IOException, ServletException {
        try {

            log.info("Pre. filter logic executed!");
            chain.doFilter(request, response);
            log.info("Post. filter logic executed!";

        } catch (Exception ex) {
            log.warn("Filter will be skipped due the exception", ex);
            chain.doFilter(request, response)
        }

    }
}
```

Информация из ServletContext

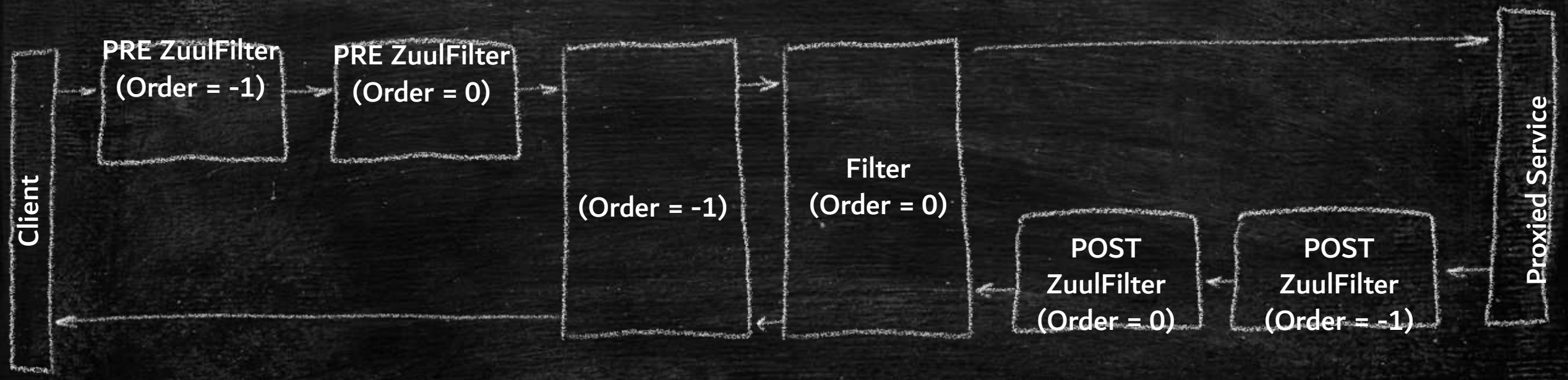
Логика PRE

Логика POST

Передача запроса дальше по цепочке

Order. Порядок выполнения фильтров.

- При написании фильтров необходимо учитывать порядок их выполнения.



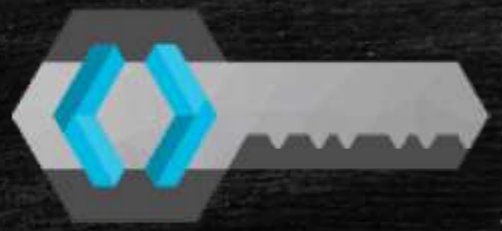
В случае существования двух фильтров одного типа с одинаковым порядком – очередность выполнения становится непредсказуемой!!

Service Discovery

- В качестве Service Discovery можно использовать пупулярный Netflix Eureka
- В нашем случае будем использолвать Kube-DNS – это сервис Kubernetes отвечающий за сопоставление DeplymentConfig Name и IP-адреса поды внутри данного DC. При этом при обращении по имени DC используется балансировщик нагрузки между подами. (Round Robin by default)

Описание стенда

gateway
http://localhost:8081



keycloack-server
http://localhost:8443

fast-rest-service-v0
http://localhost:8180

fast-rest-service-v1
http://localhost:8181

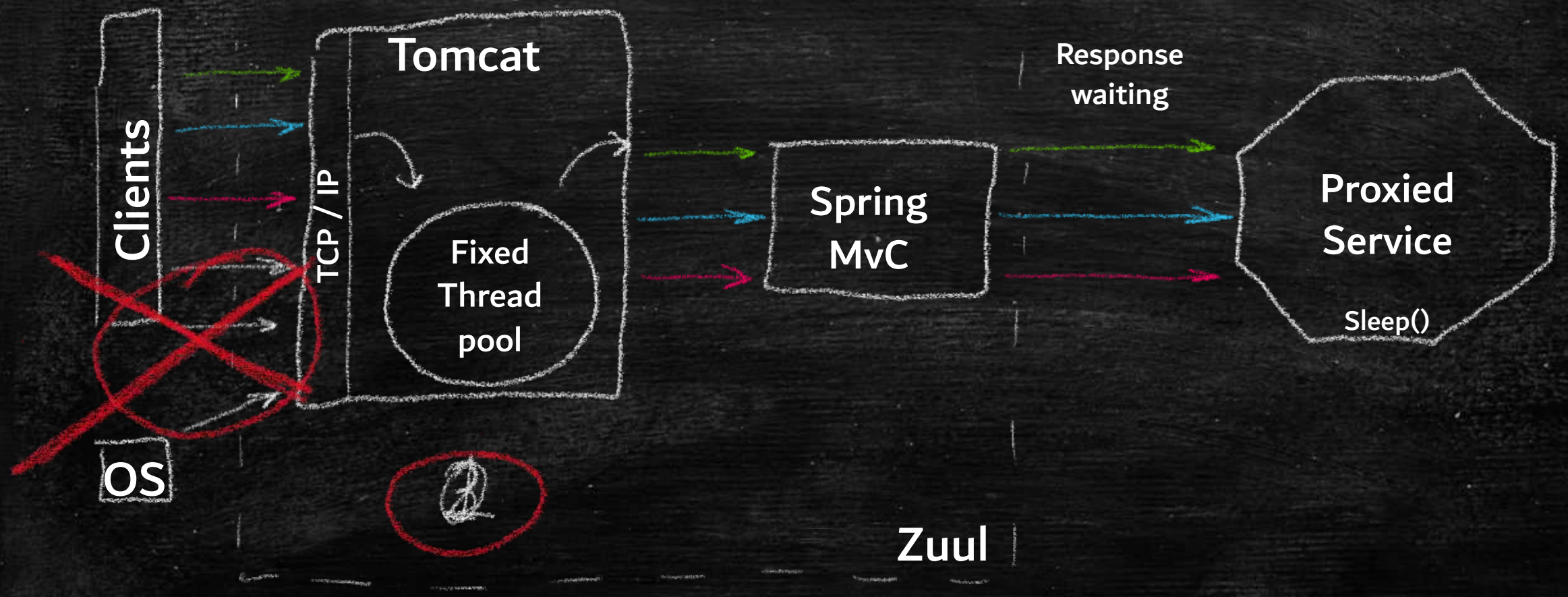
slow-rest-service-v0
http://localhost:828

slow-rest-service-v1
http://localhost:8281

Neflix Zuul Gateway

Демо

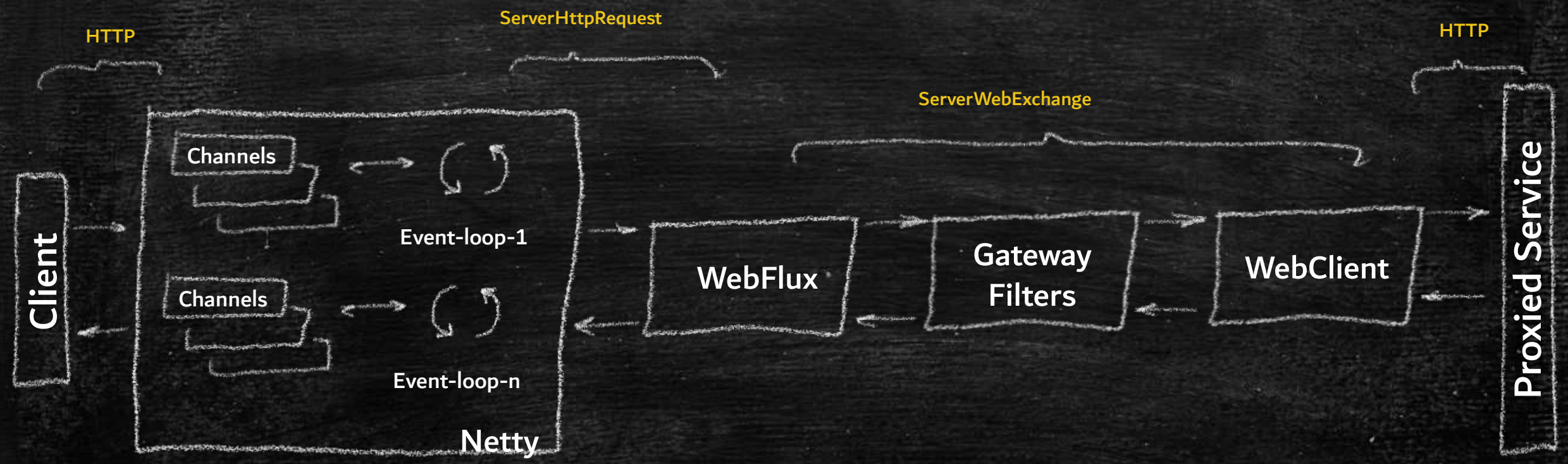
Почему возникла проблема?



Пути решения проблемы недоступности

- Увеличение количества инстансов Gateway (Кстати именно так делал Netflix до Zuul 2)
- Оптимизация и уменьшение таймаутов подключения/ответа
- Использование Deprecated Circuit Breaker Hystrix.
- ....
- Переход на Non-Blocking API Gateway

Как работатет Non-Blocking API ?



Мем Sound Like a plan

Реактивный Gateway + Blocking-API endpoints.

- Данный фактор не является блокирующим так как:
  - Spring Cloud Gateway использует WebFlux **WebClient**, что обеспечивает асинхронное, **неблокирующее** и эффективное **ожидание** ответов с поддержкой таймаутов.
  - При отправке запроса WebClient создает **реактивный поток**, в котором ожидает ответа от проксируемого сервиса.

Выбор технологии. Еще раз.

NETFLIX ZUUL

Kong

krakenD

Spring Cloud Gateway

- Spring Cloud Gateway – это проект в рамках экосистемы Spring Cloud, предоставляющий API Gateway решение для микросервисной архитектуры.
- Spring Cloud Gateway использует реактивный стек на основе Project Reactor и Netty, что обеспечивает высокую производительность, асинхронную и неблокирующую обработку запросов.
- Поддерживает глобальные и маршрутные фильтры
- Из “коробки” содержит в себе огромное количество встроенных фильтров, которые могут покрыть большинство бизнес требований к Gateway, активировав которые можно без написания нового кода.

Переход на Spring Cloud Gateway

- Подход к разработке отличается от классического, практически всю логику фильтров придется переписать.
- Большинство используемых зависимостей из Cloud времен Netflix Zuul уже устарело, и необходимо будет обновляться.

Например:

o.s.cloud:spring-cloud-starter-sluth -> io.micrometer:micrometer-tracing

o.springframework.security.oauth:* -> o.s.boot:spring-boot-starter-security

Переход d Gateway

от классиче

переписать.

используемых зав уже

ю, и необходимо буд

На

o.s.cloud:spring-cloud-starter-sluth - ometer:micrometer-tracing

o.springframework.security.oauth:* -> s.boot:spring-boot-starter-security

А что там с сервлетами? Их нет ☺

- На замену сервлетам приходит WebFlux, построенный на основе реактивного стека, на базе Project Reactor и Netty.
- Вместо, работы с RequestContext или ServletRequest / ServletResponse мы теперь обращаемся к интерфейсу WebFlux – ServerWebExchange, который содержит в себе:
  - HttpServerRequest
  - HttpServerResponse
  - Аттрибуты
  - Сессия
  - и т.д.

Как теперь писать фильтры?

Global Filter

- Глобальный фильтр применяется ко всем маршрутам

Route Filter

- Маршрутный фильтр применяется к определенному маршруту или ко всем через application.yml либо через Java config
- SCG имеет 36 встроенных фильтров для решения наших задач

- Оба типа фильтров имееют переопределяемый порядок Order

GlobalFilter

```
@Component
public class ExampleGlobalFilter implements GlobalFilter, Ordered {

    @Override
    public Mono<Void> filter(ServerWebExchange exchange,
                             GatewayFilterChain chain) {
        log.info("PRE Logic executed!");
        return chain.filter(exchange)
                .then(Mono.fromRunnable(() -> {
                    log.info("POST logic executed!");
                }));
    }

    @Override
    public int getOrder() {
        return -1;
    }
}
```

Инфо о запросе из WebFlux

Логика PRE

логика POST

Порядок

Передача слайд по цепочке фильтрам

GatewayFilter Factory

Имя фильтра

```
@Component
public class ExampleLoggerGatewayFilterFactory extends
AbstractGatewayFilterFactory<Object> {

    @Override
    public GatewayFilter apply(Object config) {

        return (exchange, chain) -> {
            log.info("PRE Logic executed");
            return chain.filter(exchange)
                    .then(Mono.fromRunnable(() -> {
                        log.info("POST Logic executed");
                    }));
            };
    }
}
```

PRE Logic

отправка дальше

POST Logic

GatewayFilter Factory (Ordered with Config)

Наименование фильтра

```
@Component
public class ExampleLoggerGatewayFilterFactory extends
AbstractGatewayFilterFactory<ExampleLoggerGatewayFilterFactory.Config> {

    @Override
    public GatewayFilter apply(Config config) {
        return new OrderedGatewayFilter((exchange, chain) -> {
            log.info("PRE Logic executed with {}", config.getParam());
            return chain.filter(exchange).then(Mono.fromRunnable(() -> {
                log.info("POST Logic executed with {}",  config.getParam());
                }));
        }, -1);

    }

    @Setter
    @Getter
    public static class Config {
        private String param;
    }
}
```

PRE

POST

Порядок

Отправка запроса в цепочку

Конфиг настройки

Включение фильтров в application.yml

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: example_route
          uri: https://example.org
          predicates:
            - Host: examplehost.org
          filters:
            - ExampleLogger=honors
```

Output:

```
PRE Logic executed with honors
POST Logic executed with honors
```

Встроенные в SCG фильтры

```yaml
spring:
 cloud:
   gateway:
     routes:
       - id: example_route
         uri: https://example.org
         predicates:
           - Host: examplehost.org
         filters:
           - ExampleLogger=honors
           - name: RequestRateLimiter
             args:
               redis-rate-limiter:
                 replenishRate: 10
                 burstCapacity: 20
                 requestedTokens: 1
```

Пример реализации Rate-Limiter с Zuul

```
apublic class ThrottlingFilter extends ZuulFilter {

  private final int limit;
  private final boolean enabled;
  private final long refreshTime;
  private final List<String> throttlingKey;
  private final AtomicInteger requestCounter;
  private final RequestLimitService requestLimitService;

  @Autowired
  public ThrottlingFilter(
      @Value("${gateway.throttling.enabled}") boolean enabled,
      @Value("${gateway.throttling.key}") String throttlingKey,
      @Value("${gateway.throttling.request-limit}") int limit,
      @Value("${gateway.throttling.refresh-rate}") long refreshTime,
      RequestLimitService requestLimitService
  ) {
    this.enabled = enabled;
    this.limit = limit;
    this.refreshTime = refreshTime;
    this.throttlingKey = List.of(throttlingKey);
    this.requestLimitService = requestLimitService;
    this.requestCounter = new AtomicInteger(0);
  }

  @Scheduled(fixedDelayString = "${gateway.throttling.refresh-rate}")
  public void counterReset() {
    requestCounter.set(0);
  }

  @Override
  public String filterType() {
    return PRE_TYPE;
  }

  @Override
  public int filterOrder() {
    return ORDER_THROTTLING_FILTER;
  }

  @Override
  public boolean shouldFilter() {
    return enabled;
  }

  @Override
  public Object run() {
    try {
      boolean isAllowed = requestLimitService.isRequestAllowed(throttlingKey, limit, refreshTime);
      if (!isAllowed) {
        throwThrottlingException();
      }
    } catch (Exception ex) {
      if (ex instanceof GatewayException) {
        throw ex;
      }
      log.error("Something went wrong when trying to get information from Redis", ex);
      if (requestCounter.get() >= limit) {
        throwThrottlingException();
      } else {
        requestCounter.incrementAndGet();
      }
    }
    return null;
  }
  private void throwThrottlingException() {
    throw new GatewayException(ExceptionType.LimitExceededException);
  }
}
```

Spring Cloud Gateway

Демонстрация кода и работы Spring Cloud Gateway.

Spring Boot 3 - Native

- Spring Boot 3 имеет поддержку компиляции в native образ средствами AOT Compiler.
- Пожертвовав аннотациями @ConditionalOnProperty и @Profile мы можем получить скорость запуска сервиса до 100 раз быстрее чем запуск Jar!

А что в нашем деле главное?

Быстро откатиться! ☺

Преимущество и недостатки Zuul

Преимущества

- Легко писать фильтры, так как у нас всегда под рукой контекст и там всегда лежит запрос
- При помощи фильтров можно реализовать абсолютно любую логику преобразования запроса - ответа

Недостатки

- Количество RPS к приложению напрямую привязано к доступным потокам, соответственно единственный способ расширения – это добавление потоков и скейлинг под вверх.
- Ограничения использования Hystrix

Преимущество и недостатки Spring Cloud Gateway

Преимущества

- Реактивный стек, запросы не привязаны к потокам.
- Большая гибкость настройки, глобальные и частные фильтры. Очень много фильтров из коробки.
- Поддержка resilince4j
- Отличная интеграция с Spring Boot / Spring Cloud

Недостатки

- Запросы не привязаны к потоку и контексту, нельзя в любом месте обратиться к параметрам запроса
- Несовместимость с прошлым решением (Spring Cloud Zuul) надо переписывать заново весь функционал
- ?? Нужно изучить что то новое ☺ (Преимущество)

Затраты на переход на SCG

- Zuul и Cloud Gateway не совместимы и придется переписать все фильтры
- Если вы используете spring-security-oauth2 – необходимо будет мигрировать на spring-security
- Очень много фильтров идет из коробки у Spring Cloud Gateway и возможно большую часть кода вы замените на пару строк в application.yml
- Некоторые вещи в SCG реализуются проще.

Что же получили на выходе:

- Современный Non-Blocking API Gateway на актуальном Spring Boot 3
- Надежную защиту от веерного отказа, когда один или несколько сервисов роняют все ваше приложение.
- Поддержку всех технологий типа: WebSocket и SSE
- Возможность постепенной реализации Flux от клиента и до бекенд сервисов

- Бонус в виде перехода в Native из коробки

Q & A

TechniXC.N.L@gmail.com

@letov_nikita

@TechniXC

https://github.com/TechniXC/