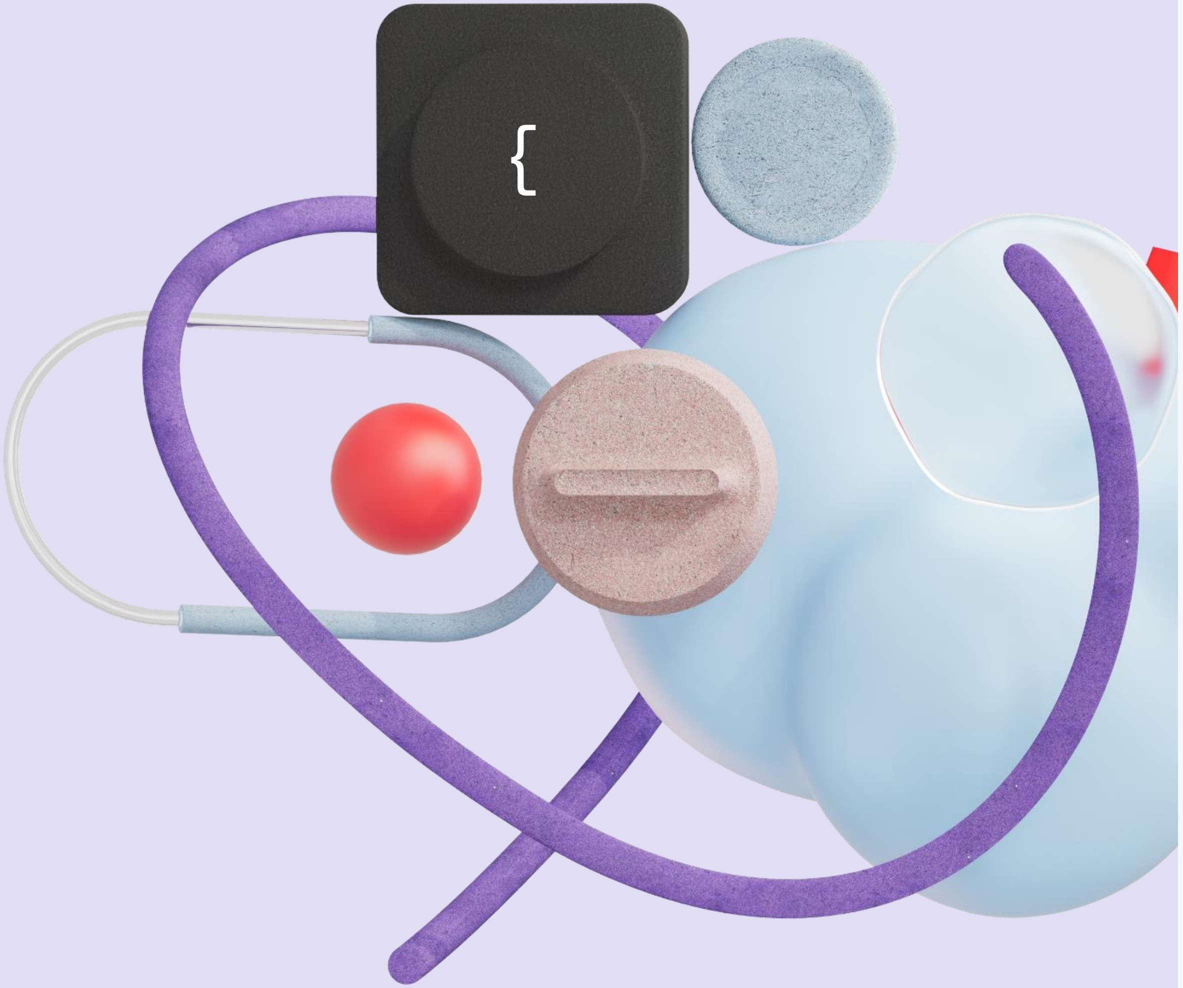
МТС True Tech +

Метрики с Micrometer

Сурен Калайчьян

Backend developer, МТС Digital



> WHOAMI

- 5 лет как backend java dev
- Преподаватель в ведущем московском техническом вузе
- Амбассадор МТС

Кубанский государственный университет 1920 Россия

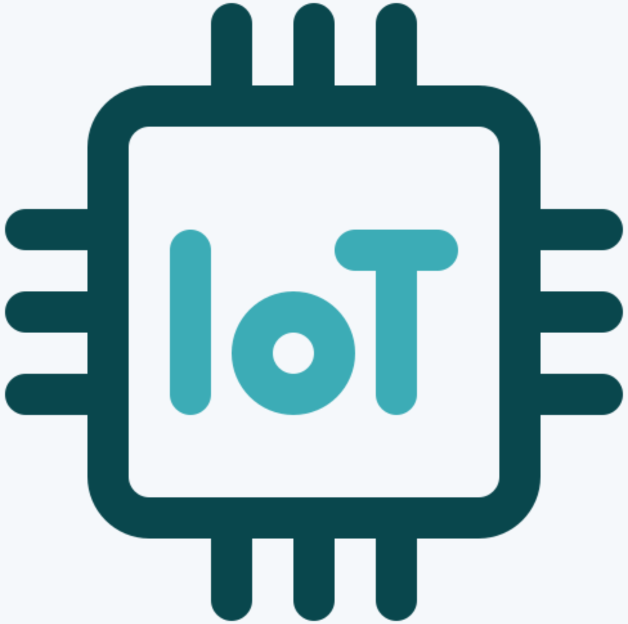
Ростелеком

МТС

WALIOT

IOT PLATFORM

- Поддерживаемые протоколы: MQTT, COAP, HTTP, TCP
- 30 микросервисов
- Выдерживаемая нагрузка до 10к RPS

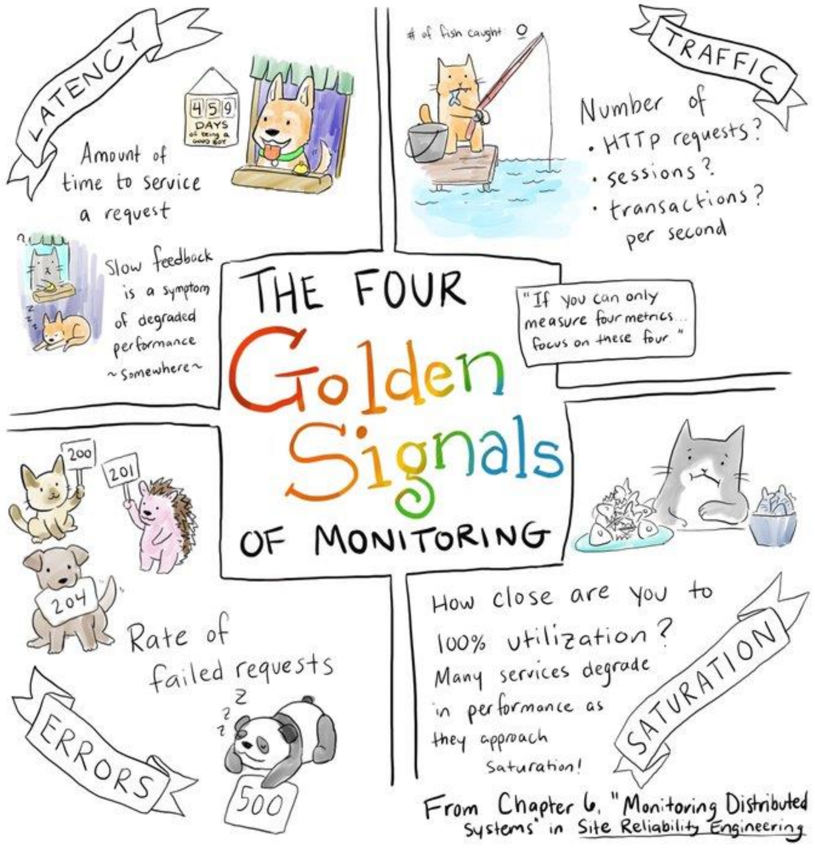


ЧТО БУДЕТ

1. Пройдемся по базовым вещам метрик
2. Пройдемся от и до по внедрению метрик в проекте
3. Параллельно разберем нюансы

ЦЕЛЬ

1. Обеспечить мониторинг системы (Покрыть 4-мя золотыми сигналами)
2. Добавить детальные метрики по каждой технологии


LATENCY
Amount of time to service a request
459 DAYS
Slow feedback is a symptom of degraded performance ~Somewhere~
# of fish caught 0
TRAFFIC
Number of
• HTTP requests?
• sessions?
• transactions? per second
THE FOUR Golden Signals OF MONITORING
"If you can only measure four metrics focus on these four"
200
201
204
Rate of failed requests
ERRORS
500
How close are you to 100% utilization? Many services degrade in performance as they approach saturation!
SATURATION
From Chapter 6, "Monitoring Distributed Systems" in Site Reliability Engineering

MICROMETER

- SLF4J, but for observability ©
- Allowing you to instrument your JVM-based application ©
- Vendor-neutral

API

- Counter
- Gauge
- Summary
- Histogram

API

- Counter
- Gauge
- DistributionSummary
- Timer

SPRING BOOT ACTUATOR

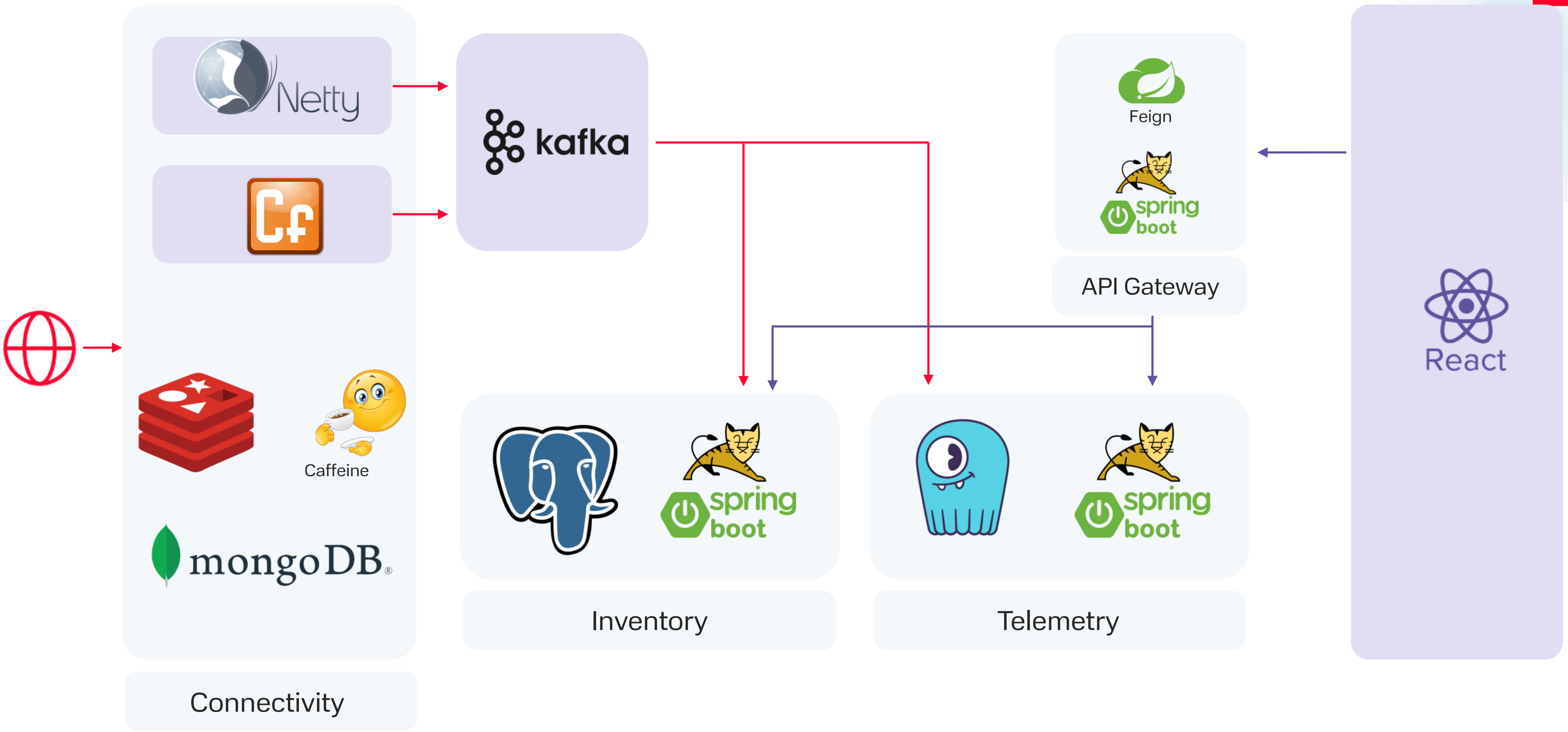
- Основной инструмент для мониторинга вашего приложения и сбора метрик с него
- Имеет готовый набор production-ready метрик
- Довольно сильная завязка на Micrometer

ВЫБОР ИНСТРУМЕНТА

ВЫБОР ИНСТРУМЕНТА

OpenTelemetry

ВЫБОР ИНСТРУМЕНТА



Caffeine

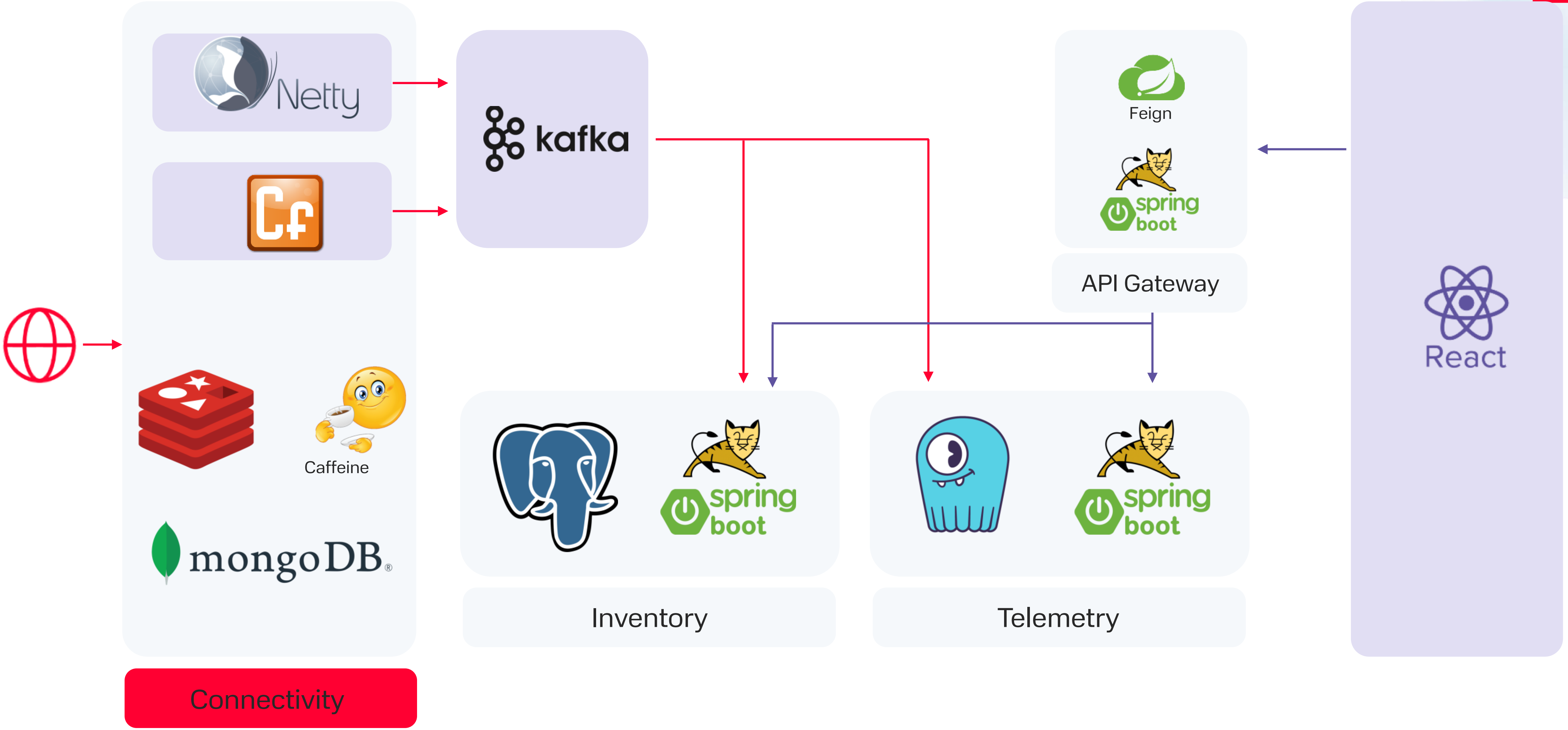
Feign

API Gateway

Inventory

Telemetry

Connectivity



Feign

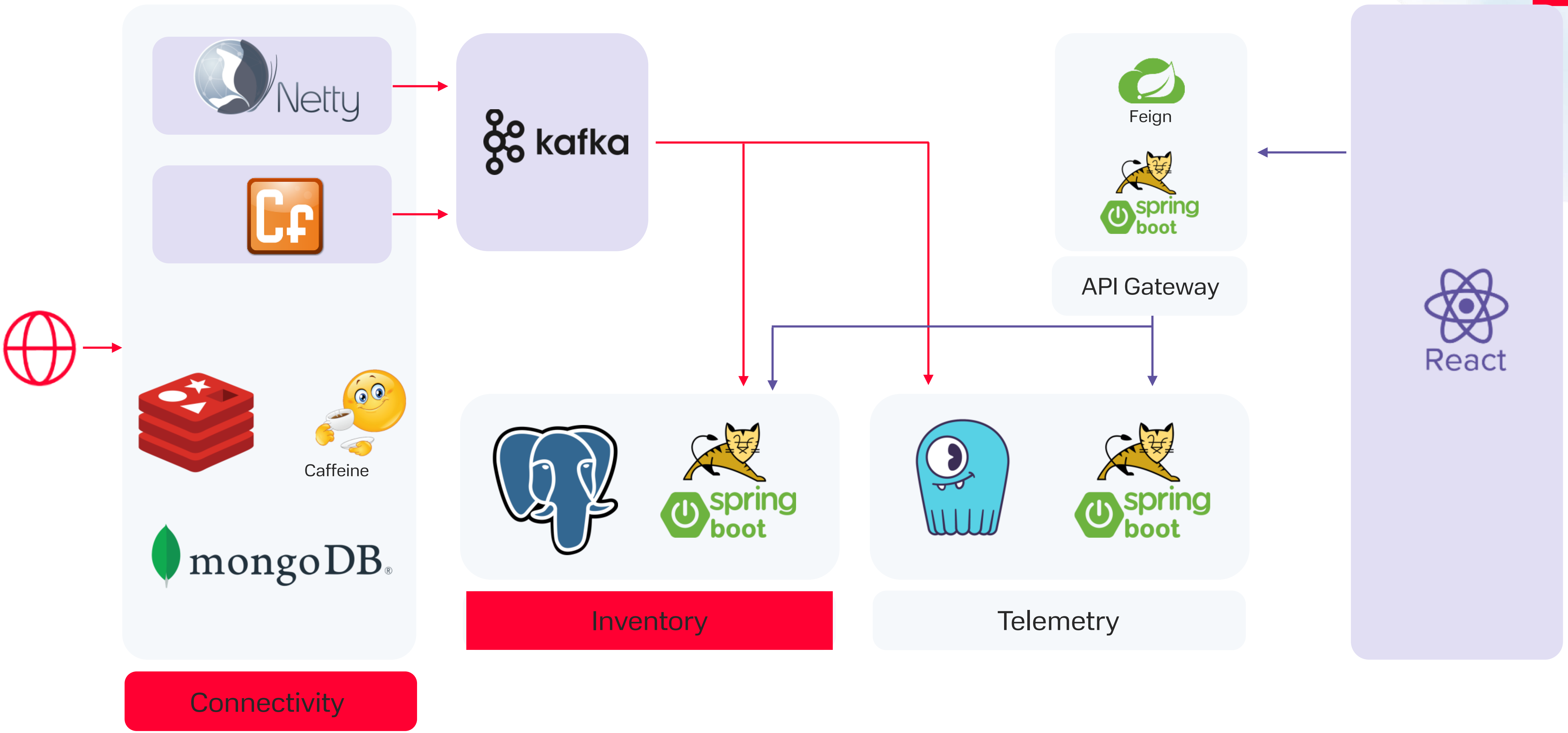
API Gateway

Caffeine

Inventory

Telemetry

Connectivity



Feign

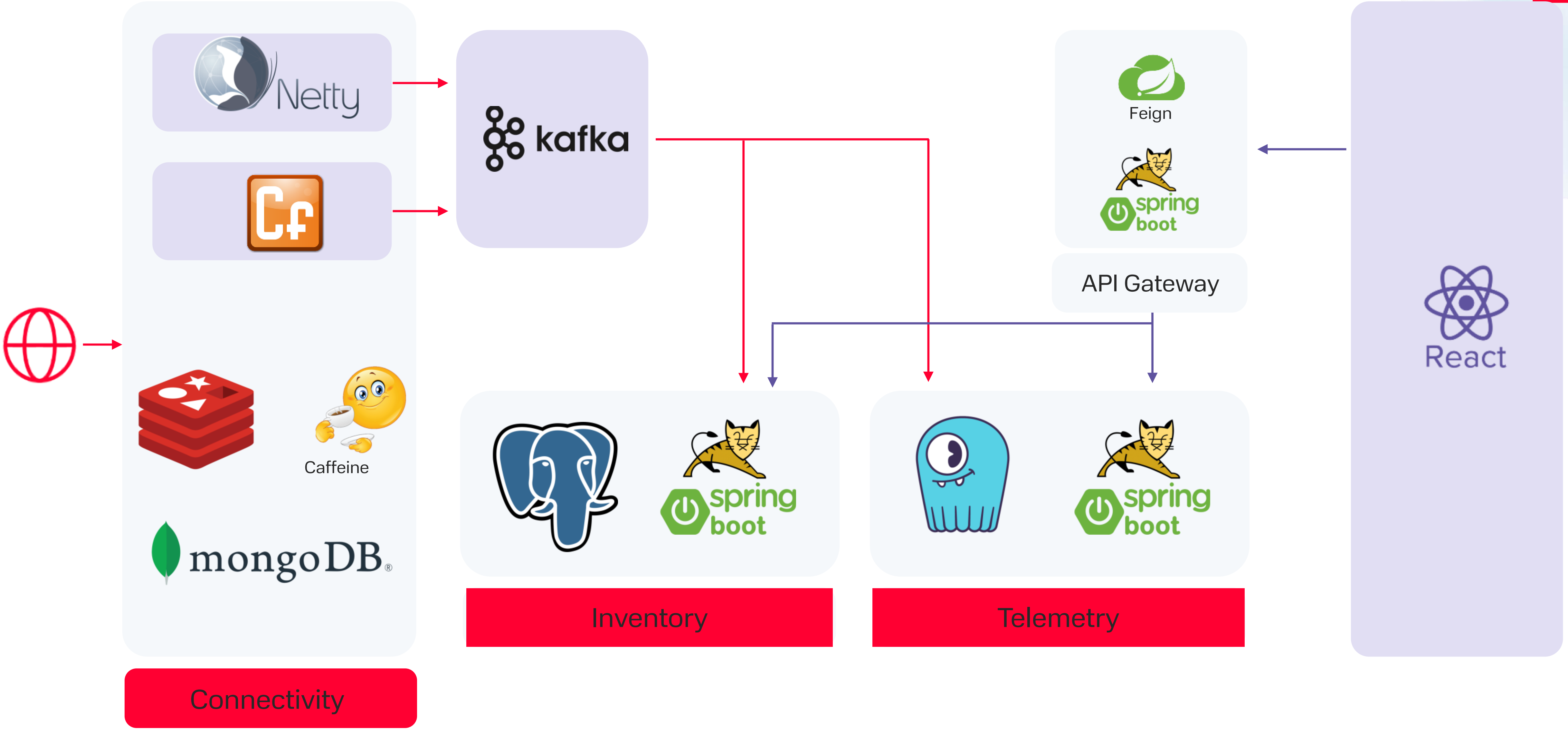
API Gateway

Caffeine

Inventory

Telemetry

Connectivity



Feign

API Gateway

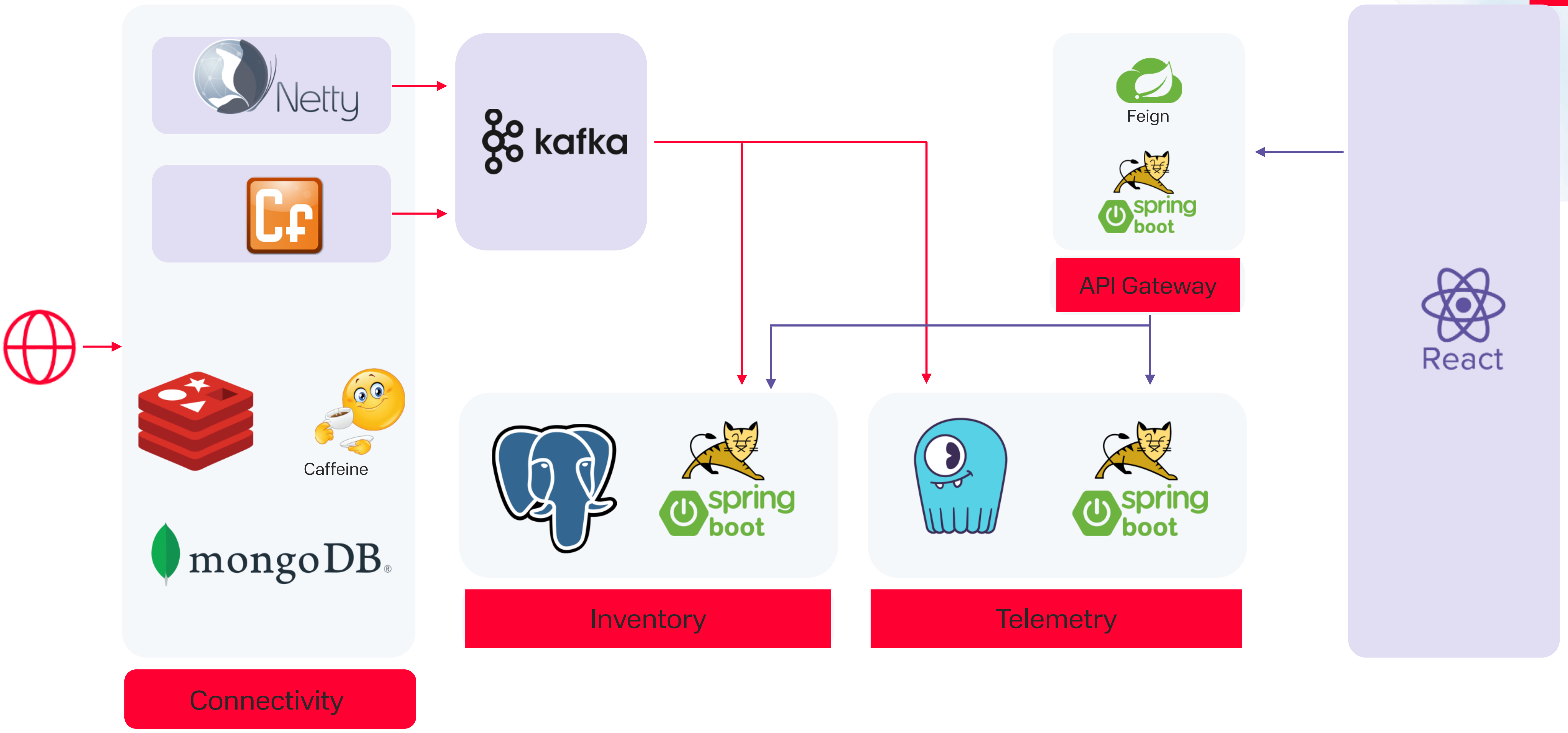
React

Caffeine

Connectivity

Inventory

Telemetry



Feign

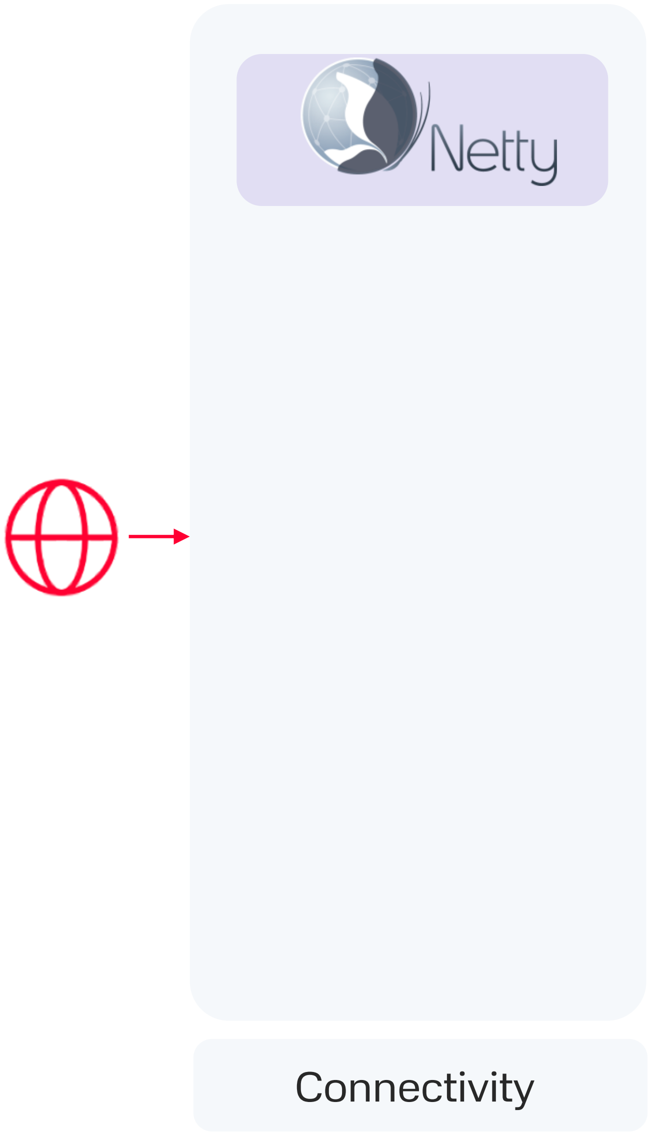
API Gateway

Caffeine

Connectivity

Inventory

Telemetry



Connectivity

Netty

- Основной фреймворк для IoT проектов (MQTT, TCP, HTTP)
- Необходимо замерять RPS и Latency

API

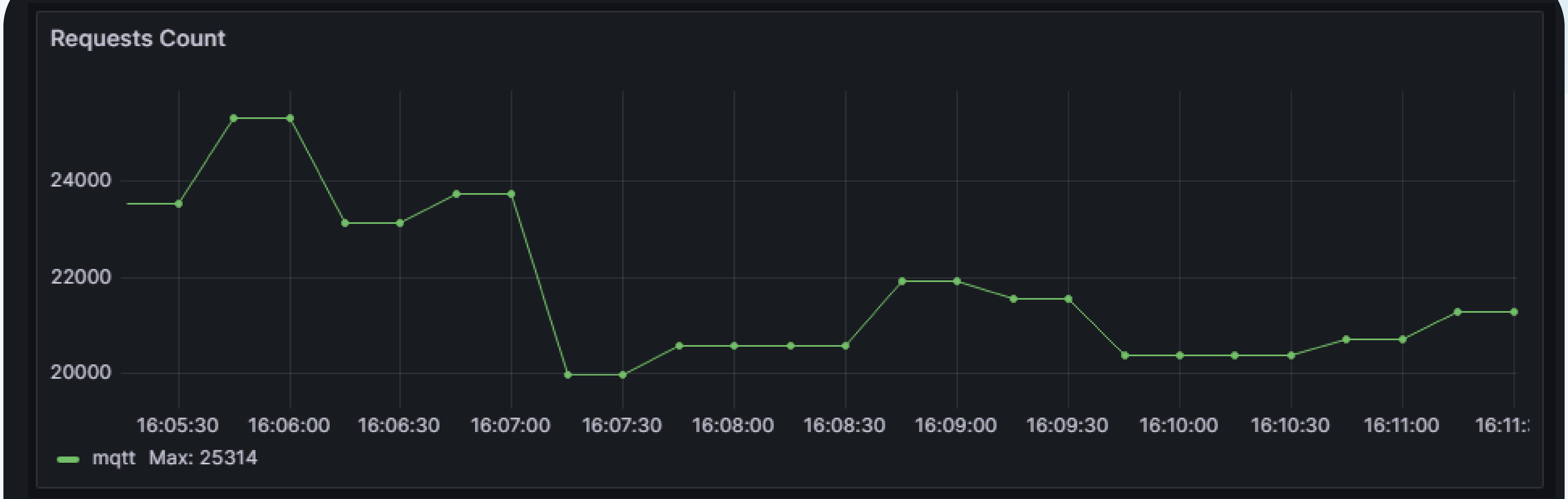
- ~~Counter~~
- ~~Gauge~~
- DistributionSummary
- ~~Timer~~

DISTRIBUTIONSUMMARY

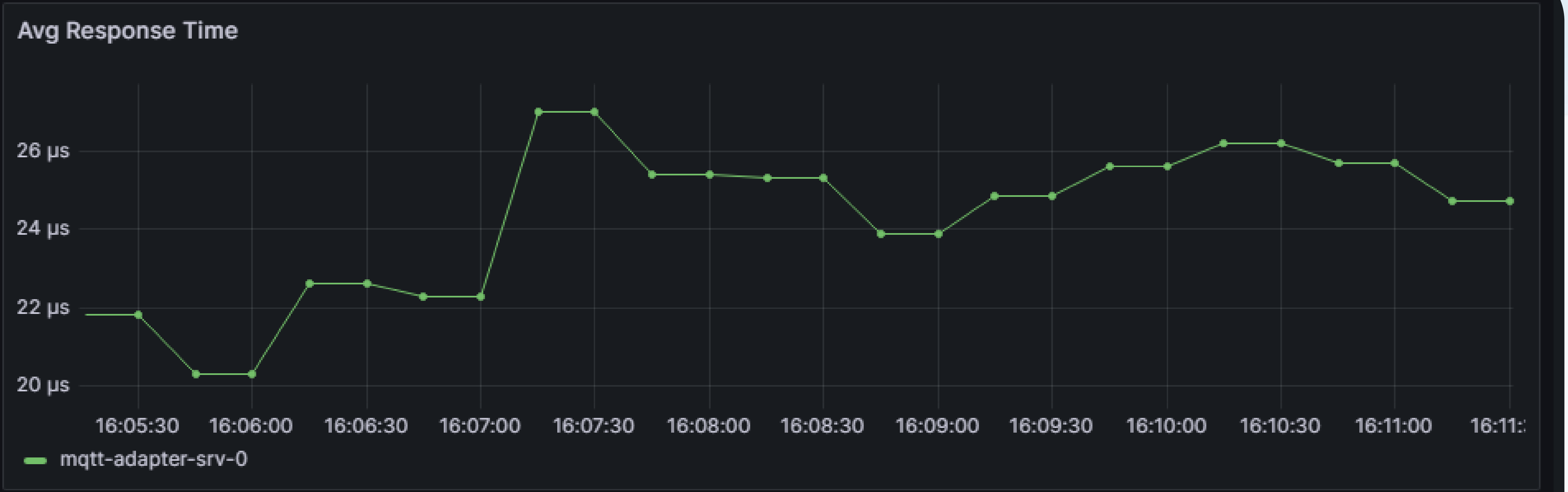
- Count
- Sum
- Max
- Quantiles (При добавлении параметров)
- Buckets (При добавлении параметров)

API

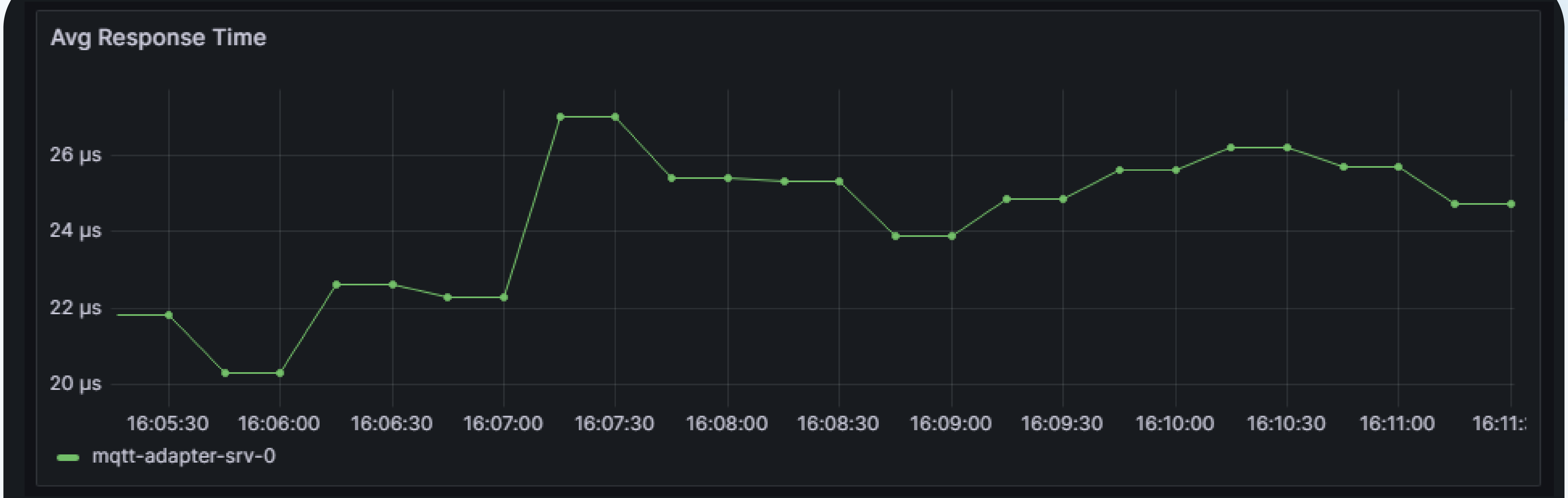
```
private final MeterRegistry registry;

public void registerTimeMetrics(double requestTime, String exceptionName) {
  DistributionSummary
      .builder("netty_custom_server_requests")
      .baseUnit("seconds")
      .description("Summary of netty server requests")
      .tag("exception", exceptionName)
      .register(registry)
      .record(requestTime);
}
```



```
rate(netty_custom_server_requests_seconds_count{namespace="$namespace", job="$service", pod=~"$pod"})
```

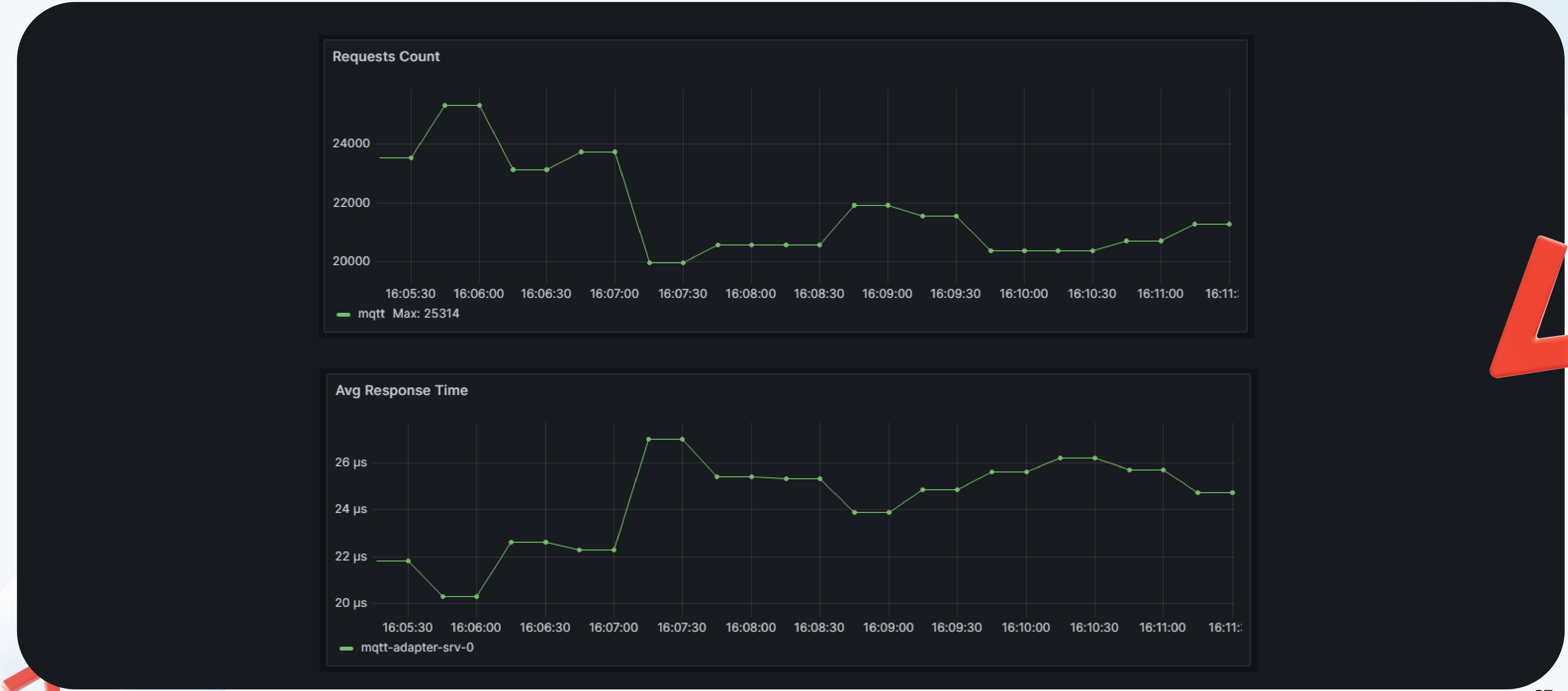


```
increase(netty_custom_server_requests_seconds_sum{namespace="$namespace", job="$service", pod=~"$pod"}[5m])
/ increase(netty_custom_server_requests_seconds_count{namespace="$namespace", job="$service", pod=~"$pod"}[5m])
```



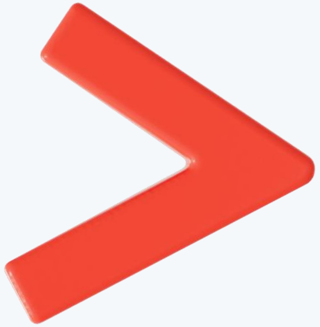
```
increase(netty_custom_server_requests_seconds_sum{namespace="$namespace", job="$service", pod=~"$pod"}[5m])
/ increase(netty_custom_server_requests_seconds_count{namespace="$namespace", job="$service", pod=~"$pod"}[5m])
```

GRAFANA


Requests Count
24000
22000
20000
16:05:30
16:06:00
16:06:30
16:07:00
16:07:30
16:08:00
16:08:30
16:09:00
16:09:30
16:10:00
16:10:30
16:11:00
16:11:
mqtt Max: 25314
Avg Response Time
26 µs
24 µs
22 µs
20 µs
16:05:30
16:06:00
16:06:30
16:07:00
16:07:30
16:08:00
16:08:30
16:09:00
16:09:30
16:10:00
16:10:30
16:11:00
16:11:
mqtt-adapter-srv-0

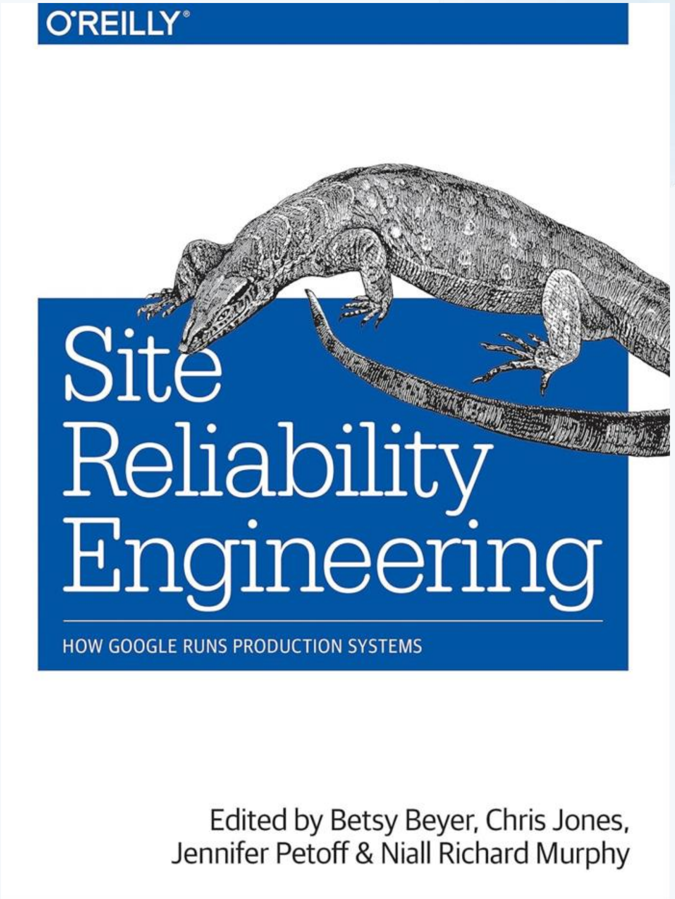
SLA/SLO/SLI

- На прохождение end-to-end запроса установлен SLA в 1 секунду
- Так же присутствует SLA на отправку ответа устройству (зависит от типа)



SLA/SLO/SLI

- На прохождение end-to-end запроса установлен SLA в 1 секунду
- Так же присутствует SLA на отправку ответа устройству (зависит от типа)



O'REILLY

Site Reliability Engineering

HOW GOOGLE RUNS PRODUCTION SYSTEMS

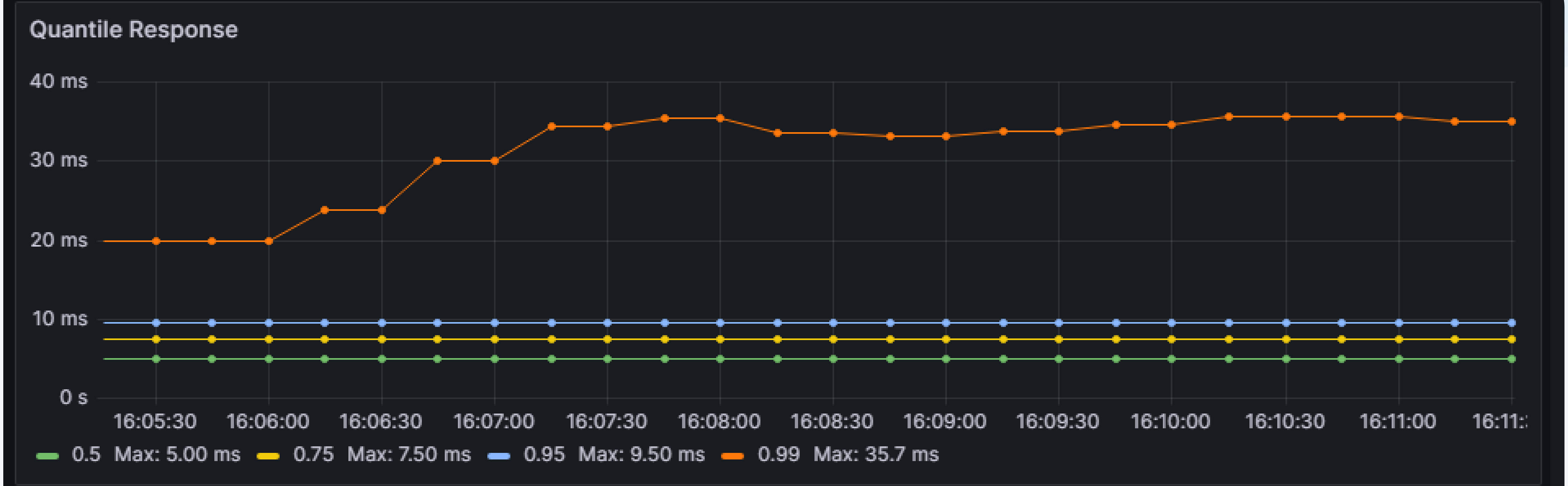
Edited by Betsy Beyer, Chris Jones,
Jennifer Petoff & Niall Richard Murphy

API

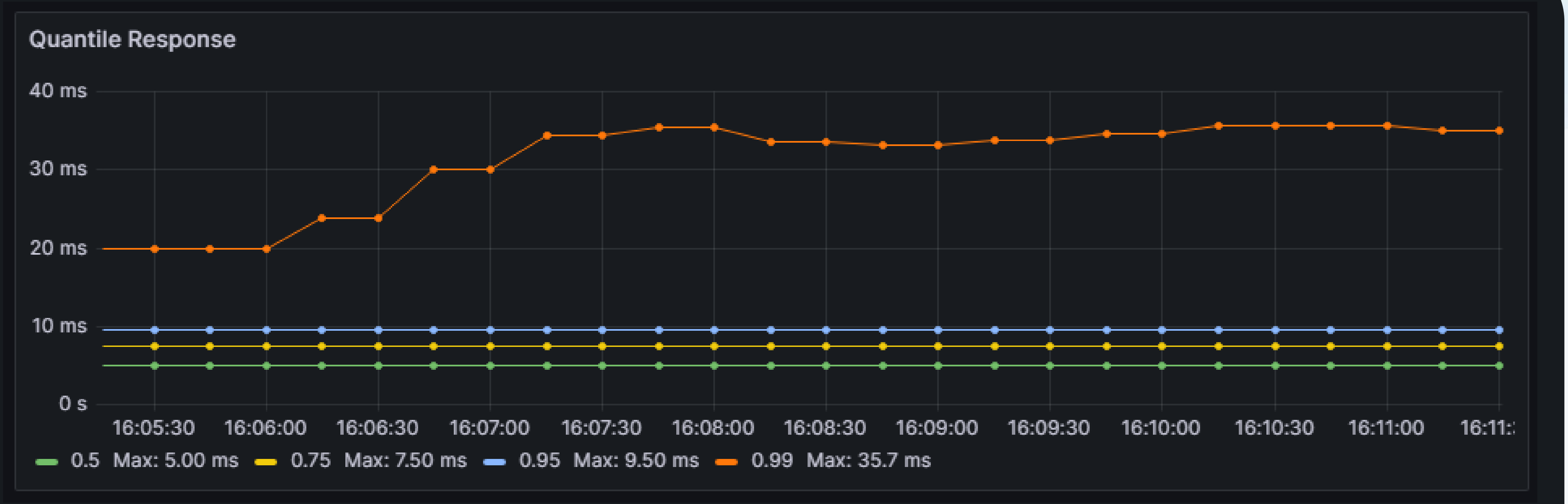
```
private final MeterRegistry registry;

public void registerTimeMetrics(double requestTime, String exceptionName) {
  DistributionSummary
      .builder("netty_custom_server_requests")
      .baseUnit("seconds")
      .description("Summary of netty server requests")
      .tag("exception", exceptionName)
      .register(registry)
      .record(requestTime);
}
```

API

```
private final MeterRegistry registry;

public void registerTimeMetrics(double requestTime, String exceptionName) {
  DistributionSummary
      .builder("netty_custom_server_requests")
      .baseUnit("seconds")
      .description("Summary of netty server requests")
      .tag("exception", exceptionName)
      .publishPercentiles(0.5, 0.75, 0.95, 0.99)
      .register(registry)
      .record(requestTime);
}
```



```
netty_custom_server_requests_seconds{namespace="$namespace", job="$service", pod="$pod", quantile="0.5"}
```



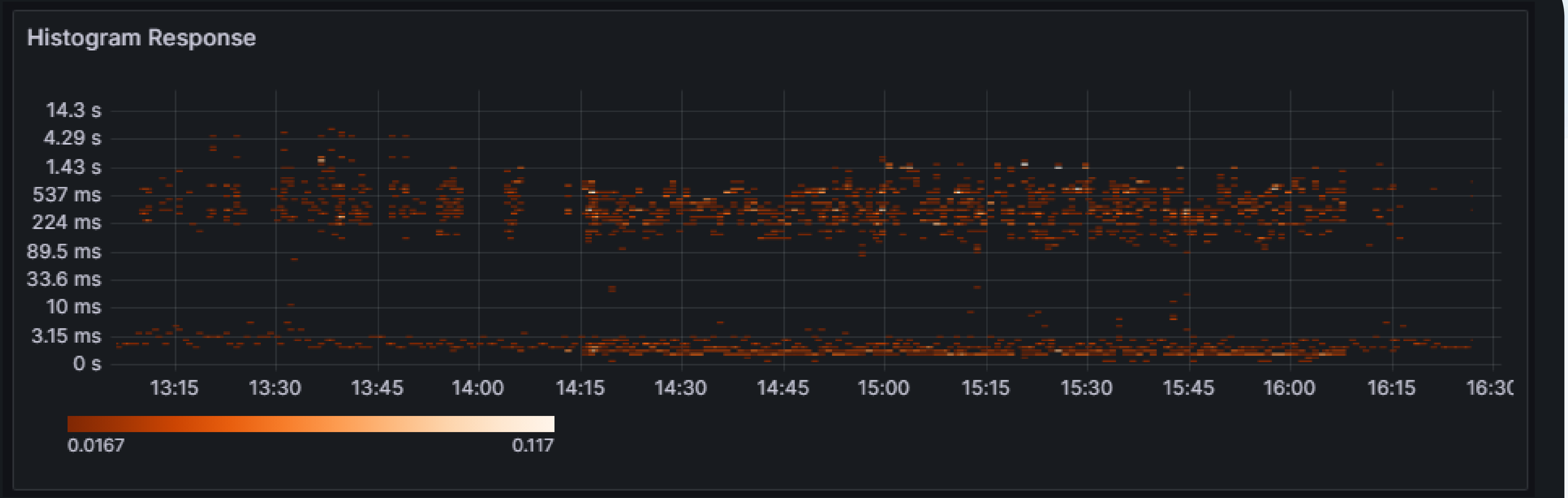
```
netty_custom_server_requests_seconds{namespace="$namespace", job="$service", pod="$pod", quantile="0.5"}
netty_custom_server_requests_seconds{namespace="$namespace", job="$service", pod="$pod", quantile="0.75"}
netty_custom_server_requests_seconds{namespace="$namespace", job="$service", pod="$pod", quantile="0.95"}
netty_custom_server_requests_seconds{namespace="$namespace", job="$service", pod="$pod", quantile="0.99"}
```

Разрежённость запросов

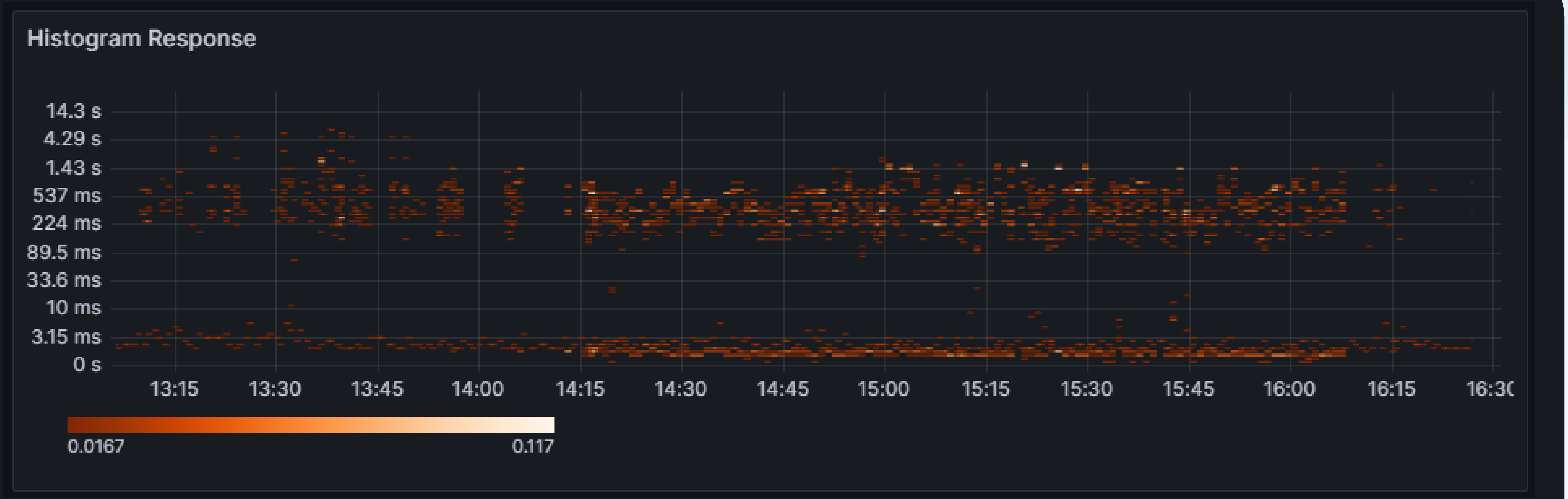
- Есть потребность иметь более детальный график о времени выполнения запросов

API

```
private final MeterRegistry registry;

public void registerTimeMetrics(double requestTime, String exceptionName) {
  DistributionSummary
      .builder("netty_custom_server_requests")
      .baseUnit("seconds")
      .description("Summary of netty server requests")
      .tag("exception", exceptionName)
      .publishPercentiles(0.5, 0.75, 0.95, 0.99)
      .publishPercentileHistogram()
      .register(registry)
      .record(requestTime);
}
```



```
sum(
     rate(netty_custom_server_requests_seconds_bucket{namespace="$namespace", job="$service",pod=~"$pod"}
     [$__rate_interval]))
by  (le))
```

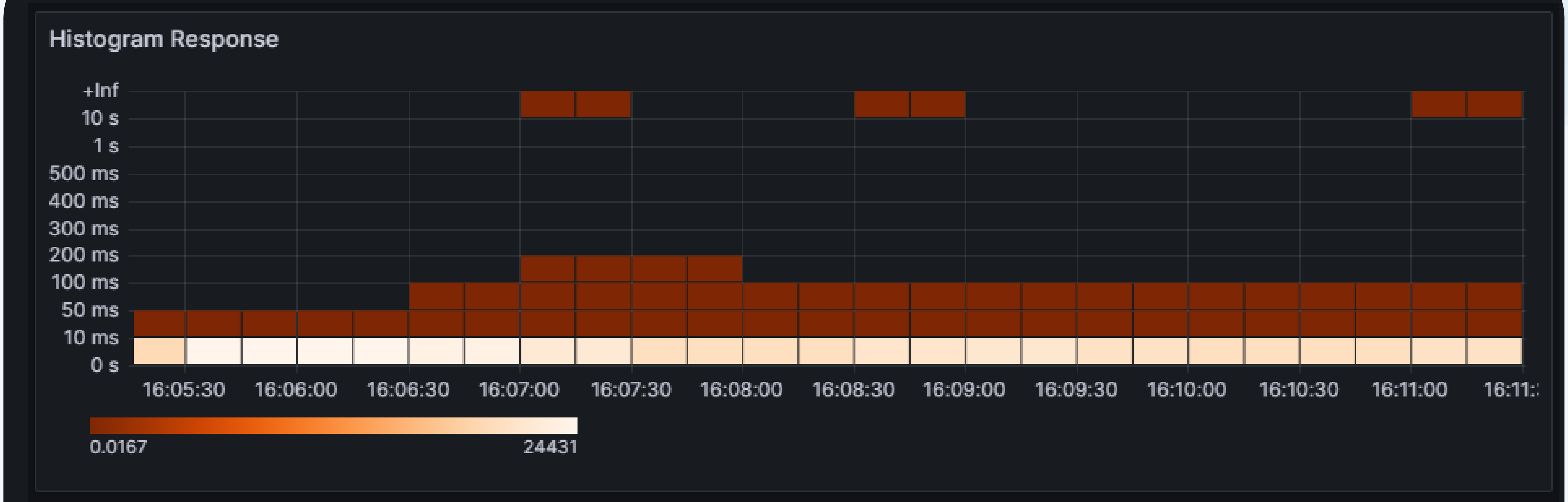


Histogram Response

```
sum(
      rate(netty_custom_server_requests_seconds_bucket{namespace="$namespace", job="$service",pod=~"$pod"}
      [$__rate_interval]))
by (le))
```

API

```
private final MeterRegistry registry;

public void registerTimeMetrics(double requestTime, String exceptionName) {
  DistributionSummary
     .builder("netty_custom_server_requests")
     .baseUnit("seconds")
     .description("Summary of netty server requests")
     .tag("exception", exceptionName)
     .publishPercentiles(0.5, 0.75, 0.95, 0.99)
     .serviceLevelObjectives(10, 50, 100, 200, ... 10000)
     .register(registry)
     .record(requestTime);
}
```



```
sum(
    rate(netty_custom_server_requests_seconds_bucket{namespace="$namespace", job="$service",pod=~"$pod"}
    [$__rate_interval]))
by (le))
```

Реплики

Инстансов адаптеров обычно больше чем 1.

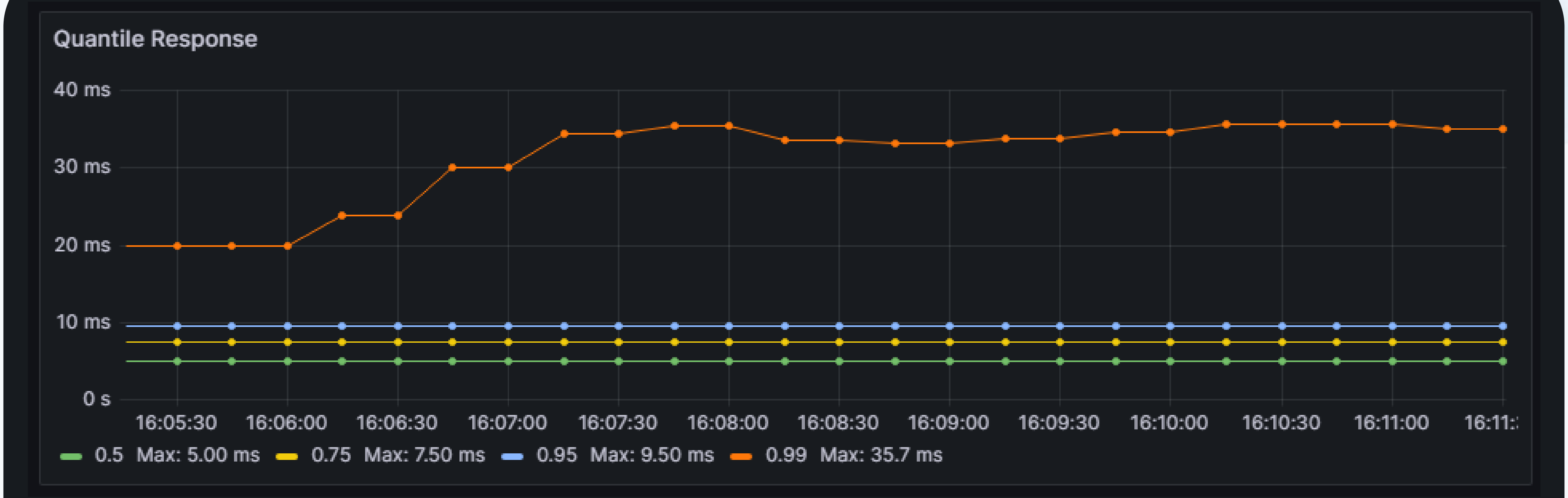
Имеет ли это влияние на метрики?

Реплики

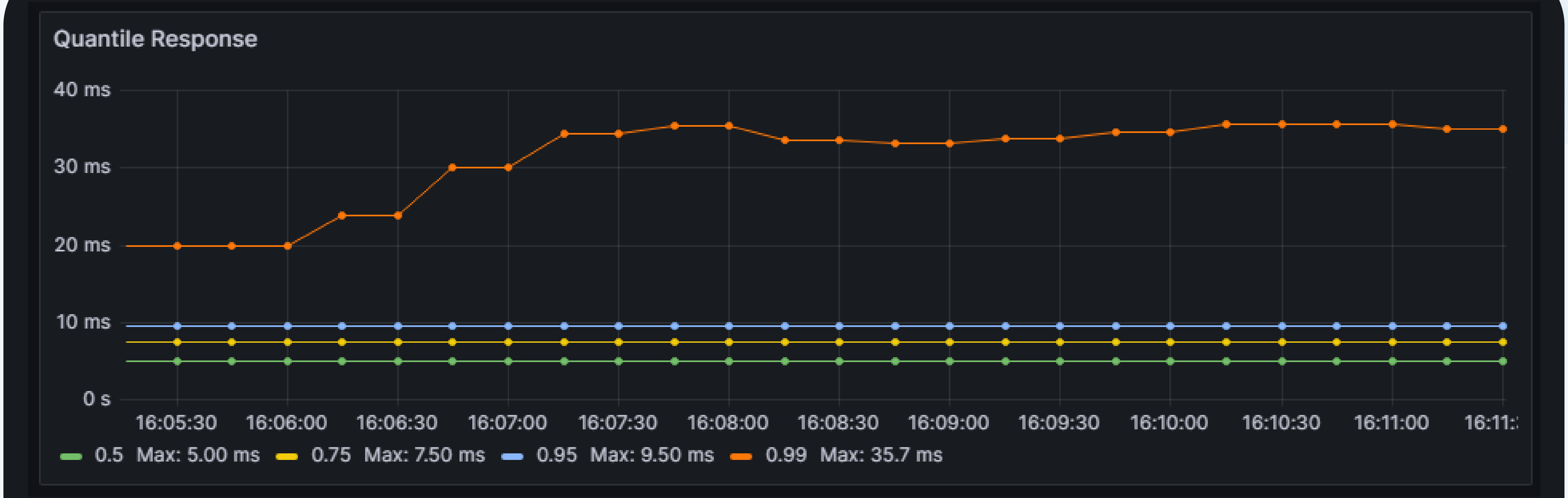
- Квантили (Summary) высчитываются на стороне сервиса
- Бакеты (Histogram) высчитываются на стороне Prometheus
- Можно использовать бакеты так же и для расчета квантилей (для графиков SLA)

API

```
private final MeterRegistry registry;

public void registerTimeMetrics(double requestTime, String exceptionName) {
  DistributionSummary
      .builder("netty_custom_server_requests")
      .baseUnit("seconds")
      .description("Summary of netty server requests")
      .tag("exception", exceptionName)
      ~~.publishPercentiles(0.5, 0.75, 0.95, 0.99)~~
      .serviceLevelObjectives(10, 50, 100, 200, ... 10000)
      .register(registry)
      .record(requestTime);
}
```

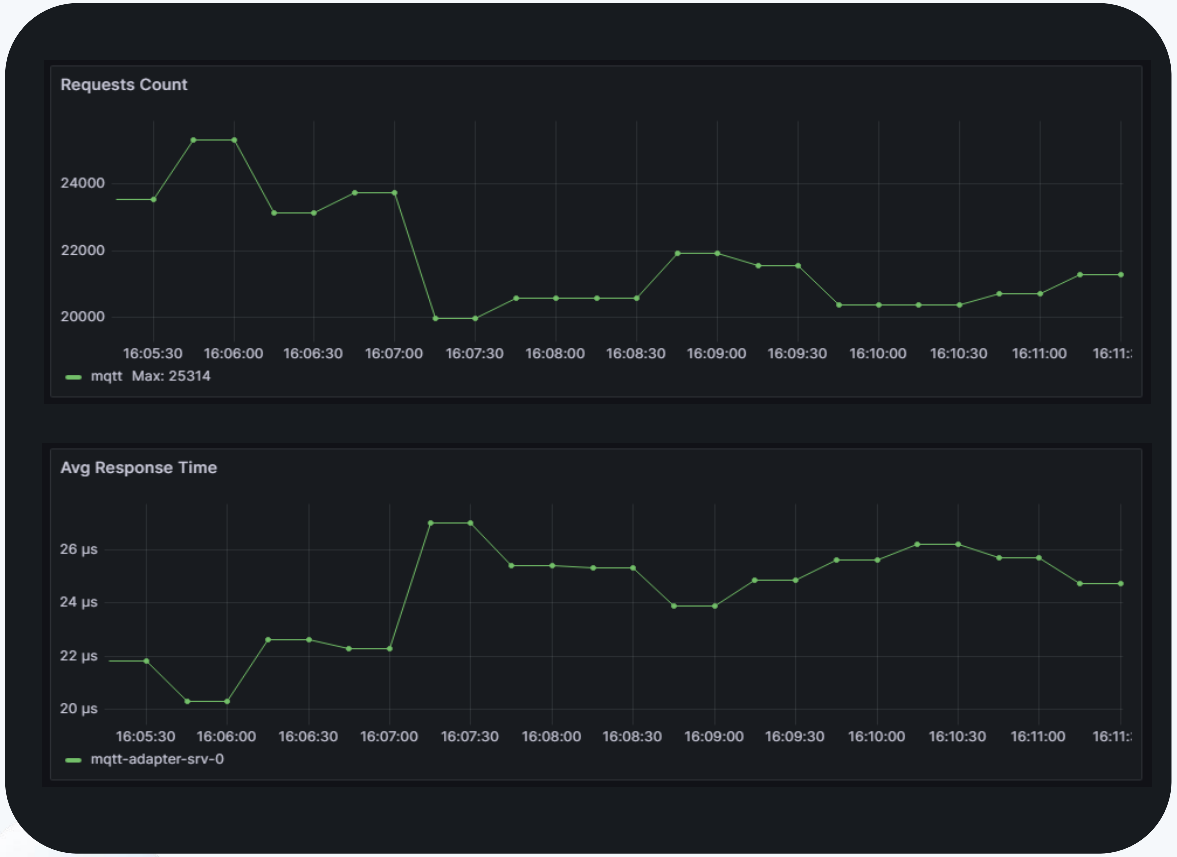


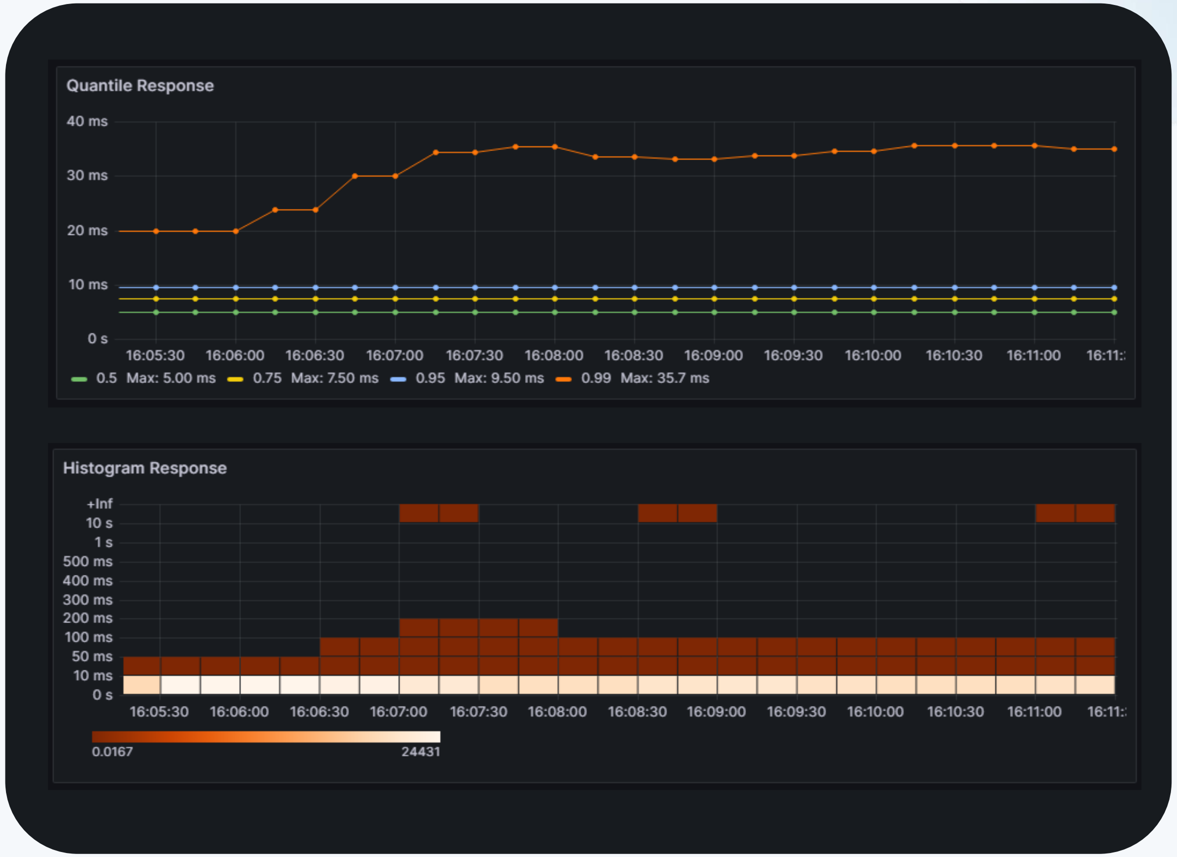
```
histogram_quantile(0.5,
     rate(netty_custom_server_requests_seconds_bucket{namespace="$namespace", job="$service",pod=~"$pod"}
     [$__rate_interval]))
by (le))
```

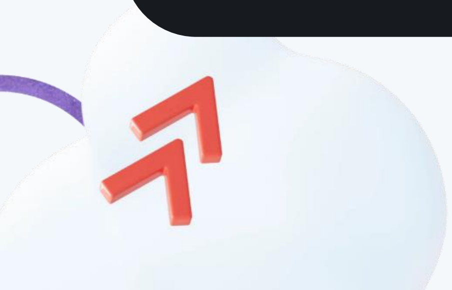


```
histogram_quantile(0.5,
     rate(netty_custom_server_requests_seconds_bucket{namespace="$namespace", job="$service",pod=~"$pod"}
     [$__rate_interval]))
by (le))
```

GRAFANA


Requests Count
24000
22000
20000
16:05:30
16:06:00
16:06:30
16:07:00
16:07:30
16:08:00
16:08:30
16:09:00
16:09:30
16:10:00
16:10:30
16:11:00
16:11:
mqtt Max: 25314
Avg Response Time
26 µs
24 µs
22 µs
20 µs
16:05:30
16:06:00
16:06:30
16:07:00
16:07:30
16:08:00
16:08:30
16:09:00
16:09:30
16:10:00
16:10:30
16:11:00
16:11:
mqtt-adapter-srv-0

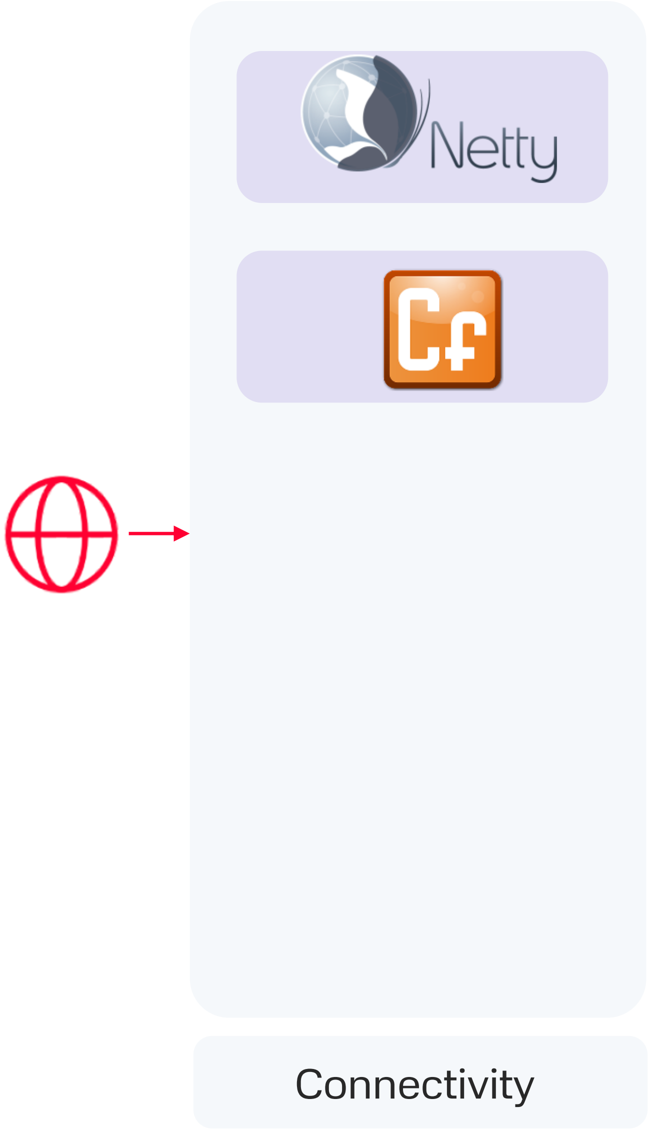

Quantile Response
40 ms
30 ms
20 ms
10 ms
0 s
16:05:30
16:06:00
16:06:30
16:07:00
16:07:30
16:08:00
16:08:30
16:09:00
16:09:30
16:10:00
16:10:30
16:11:00
16:11:
0.5 Max: 5.00 ms
0.75 Max: 7.50 ms
0.95 Max: 9.50 ms
0.99 Max: 35.7 ms
Histogram Response
+Inf
10 s
1 s
500 ms
400 ms
300 ms
200 ms
100 ms
50 ms
10 ms
0 s
16:05:30
16:06:00
16:06:30
16:07:00
16:07:30
16:08:00
16:08:30
16:09:00
16:09:30
16:10:00
16:10:30
16:11:00
16:11:
0.0167
24431



NETTY

netty_allocator_memory_pinned
netty_allocator_memory_used
netty_allocator_pooled_arenas
netty_allocator_pooled_cache_size
netty_allocator_pooled_chunk_size
netty_eventexecutor_tasks_pending
netty_allocator_pooled_threadlocal_caches

docs.micrometer.io/micrometer/reference/reference/netty.html



Connectivity

CALIFORNIUM (COAP PROTOCOL)

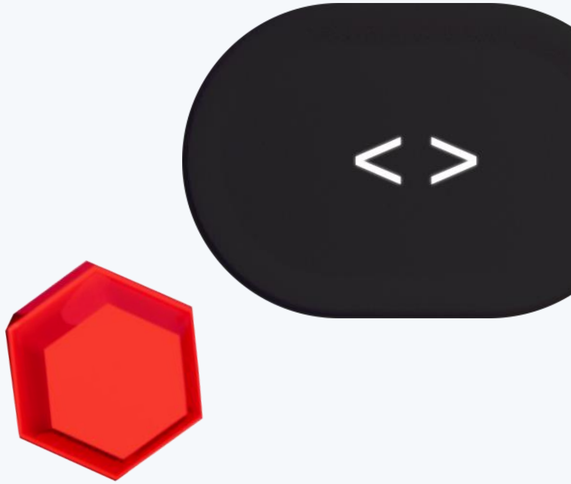
- Необходим тот же набор метрик
- Есть требование в виде low latency

OVERHEAD METRICS

- До ~10 мкс - с тегами
- До ~6 мкс - без тегов

METRICS OVERHEAD

- New Builder();
- New Immutabletag();
- New Meter.Id();



SHORTCUT (V. 1.12)

```
private final MeterRegistry registry;
private final Meter.MeterProvider<DistributionSummary> nettyTimer;

      ...
  nettyTimer = DistributionSummary
     .builder("netty_custom_server_requests")
     .baseUnit("seconds")
     .description("Summary of netty server requests")
     .serviceLevelObjectives(10, ...)
     .withRegistry(registry);

public void registerTimeMetrics(double requestTime, String exceptionName) {
  nettyTimer.withTag("Exception", ex).record(requestTime);
}
```

METRICS OVERHEAD

- New Builder();
- New Immutabletag();
- New Meter.Id();

METRICS OVERHEAD

- New Immutabletag();
- Tag.Of();
- New Tag();
- New Tags();
- New [] Tag;

API

```java
public Tags and(Tag... tags) {

  Tag[] newTags = new Tag[last + tags.length];
  System.arraycopy(this.tags, 0, newTags, 0, last);
  System.arraycopy(tags, 0, newTags, last, tags.length);
  return new Tags(newTags);
}
```

ОПТИМИЗАЦИЯ - ПЕРЕМЕННАЯ

```
private final MeterRegistry registry;
private final DistributionSummary nettyTimer;

      ...
  nettyTimer = Timer
     .builder("netty_custom_server_requests")
     .baseUnit("seconds")
     .description("Summary of netty server requests")
     .serviceLevelObjectives(10, ...)
     .register(registry);

public void registerTimeMetrics(double requestTime, String exceptionName) {
  nettyTimer.record(requestTime);
}
```

ОПТИМИЗАЦИЯ - MAP

```
private final Map<String, DistributionSummary> nettyTimers = new ConcurrentHashMap<>();

private void buildSummary(String exceptionName) {
  return DistributionSummary
      .builder("netty_custom_server_requests")
      .baseUnit("seconds")
      .description("Summary of netty server requests")
      .tag("exception", exceptionName)
      .serviceLevelObjectives(10, ...)
      .register(registry);

public void registerTimeMetrics(String metricFullName, double requestTime, String exceptionName) {

      DistributionSummary summary = nettyTimers.compute(metricFullName, (k, v) -> v == null
      ? buildSummary(metric, exceptionName)
      : v);

     summary.record(requestTime);
}
```

ОПТИМИЗАЦИЯ - MAP

```
private final Map<String, DistributionSummary> nettyTimers = new ConcurrentHashMap<>();

private void buildSummary(String exceptionName) {
  return DistributionSummary
     .builder("netty_custom_server_requests")
     .baseUnit("seconds")
     .description("Summary of netty server requests")
     .tag("exception", exceptionName)
     .serviceLevelObjectives(10, ...)
     .register(registry);

public void registerTimeMetrics(String metricFullName, double requestTime, String exceptionName) {

     DistributionSummary summary = nettyTimers.compute(metricFullName, (k, v) -> v == null
     ? buildSummary(metric, exceptionName)
     : v);

    summary.record(requestTime);
}
```

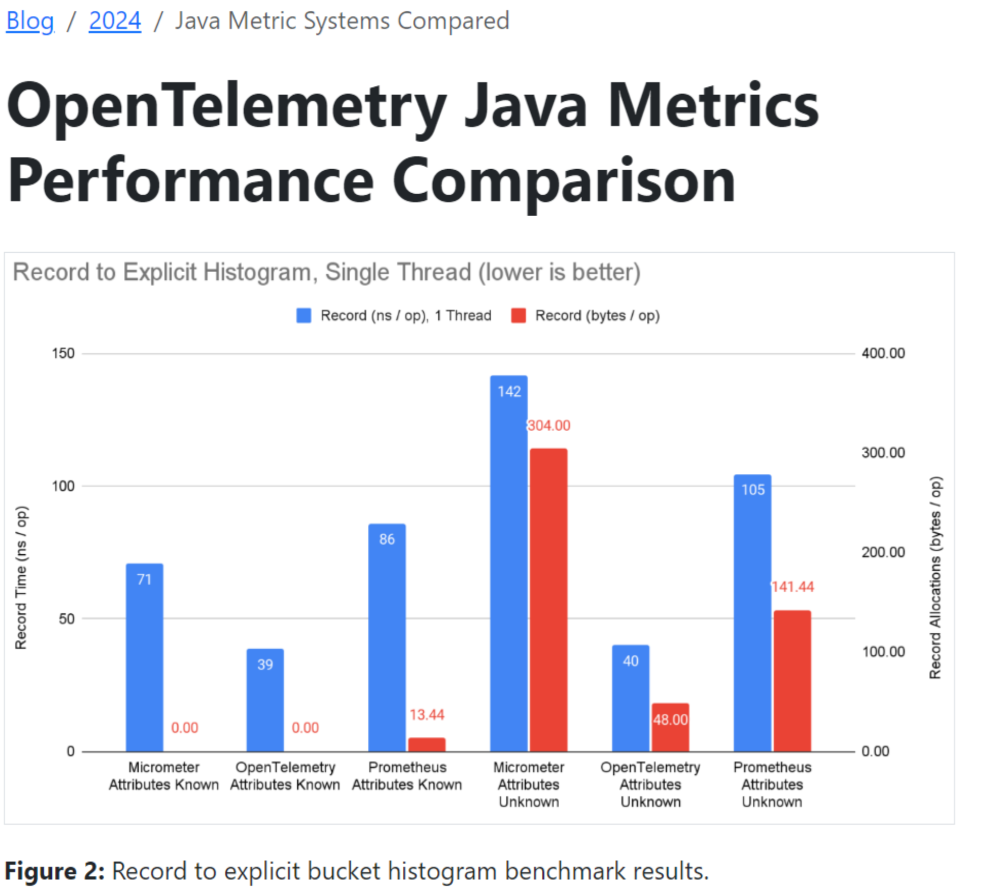
ОПТИМИЗАЦИЯ - MAP

```
private final Map<String, DistributionSummary> nettyTimers = new ConcurrentHashMap<>();

private void buildSummary(String exceptionName) {
  return DistributionSummary
      .builder("netty_custom_server_requests")
      .baseUnit("seconds")
      .description("Summary of netty server requests")
      .tag("exception", exceptionName)
      .serviceLevelObjectives(10, ...)
      .register(registry);

public void registerTimeMetrics(String metricFullName, double requestTime, String exceptionName) {

      DistributionSummary summary = nettyTimers.compute(metricFullName, (k, v) -> v == null
      ? buildSummary(metric, exceptionName)
      : v);

     summary.record(requestTime);
}
```

ОПТИМИЗАЦИЯ

- ~~До ~10 мкс~~ с тегами до 2 мкс
- ~~До ~6 мкс~~ без тегов до 600 нс

ОПТИМИЗАЦИЯ



Blog / 2024 / Java Metric Systems Compared

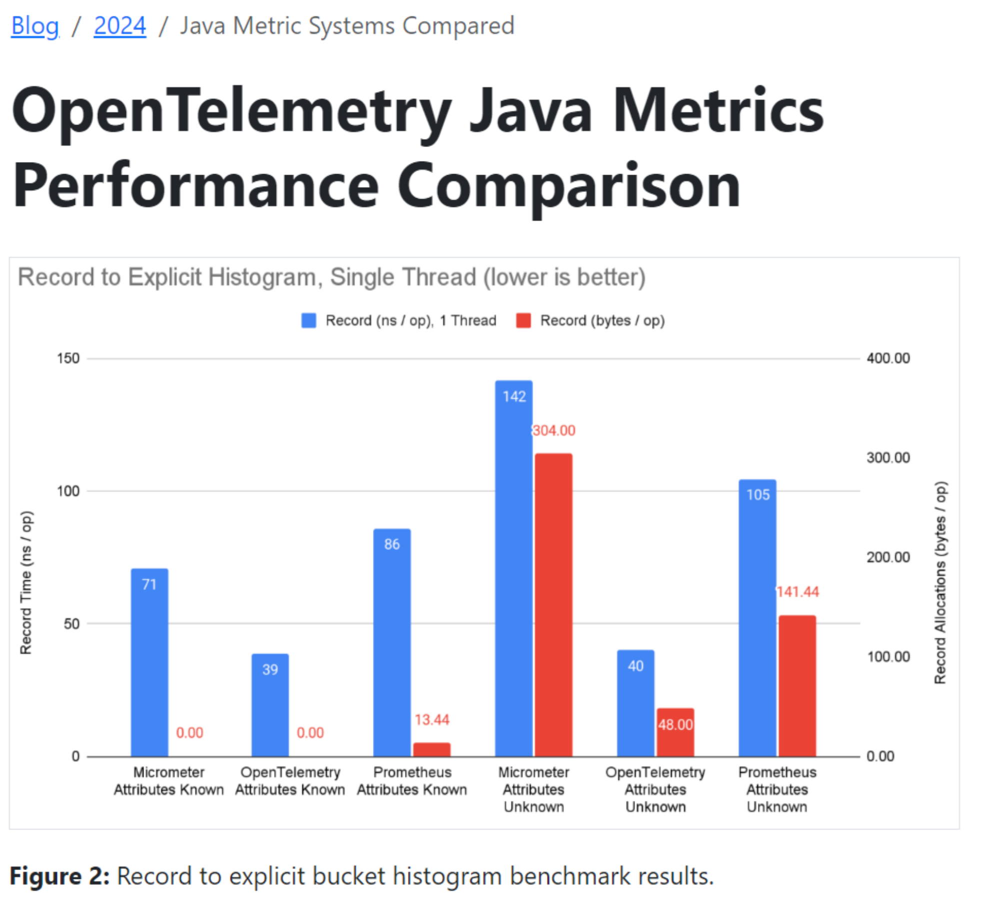
OpenTelemetry Java Metrics Performance Comparison

Record to Explicit Histogram, Single Thread (lower is better)
Record (ns / op), 1 Thread
Record (bytes / op)
Record Time (ns / op)
Record Allocations (bytes / op)
150
100
50
0
400.00
300.00
200.00
100.00
0.00
71
0.00
39
0.00
86
13.44
142
304.00
40
48.00
105
141.44
Micrometer Attributes Known
OpenTelemetry Attributes Known
Prometheus Attributes Known
Micrometer Attributes Unknown
OpenTelemetry Attributes Unknown
Prometheus Attributes Unknown

Figure 2: Record to explicit bucket histogram benchmark results.

opentelemetry.io/blog/2024/java-metric-systems-compared

ОПТИМИЗАЦИЯ



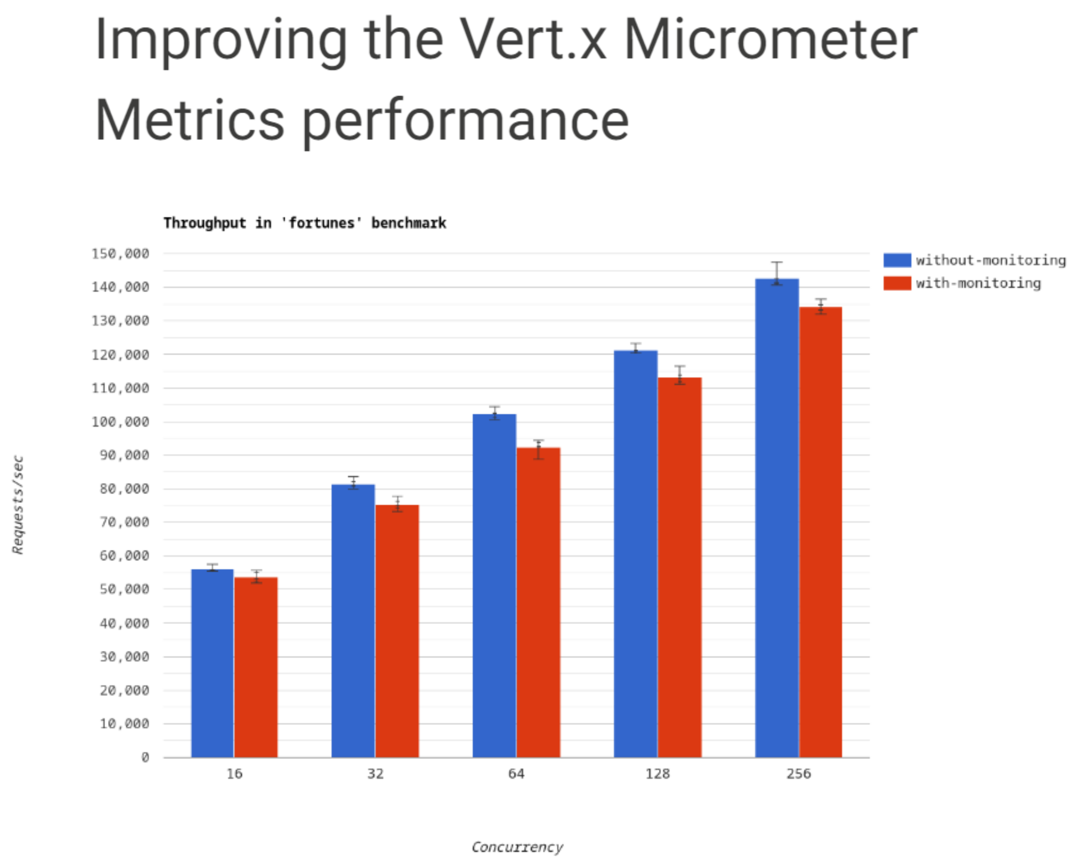
Blog / 2024 / Java Metric Systems Compared

OpenTelemetry Java Metrics Performance Comparison

Record to Explicit Histogram, Single Thread (lower is better)
Record (ns / op), 1 Thread
Record (bytes / op)
Record Time (ns / op)
Record Allocations (bytes / op)
150
100
50
0
400.00
300.00
200.00
100.00
0.00
71
0.00
39
0.00
86
13.44
142
304.00
40
48.00
105
141.44
Micrometer Attributes Known
OpenTelemetry Attributes Known
Prometheus Attributes Known
Micrometer Attributes Unknown
OpenTelemetry Attributes Unknown
Prometheus Attributes Unknown

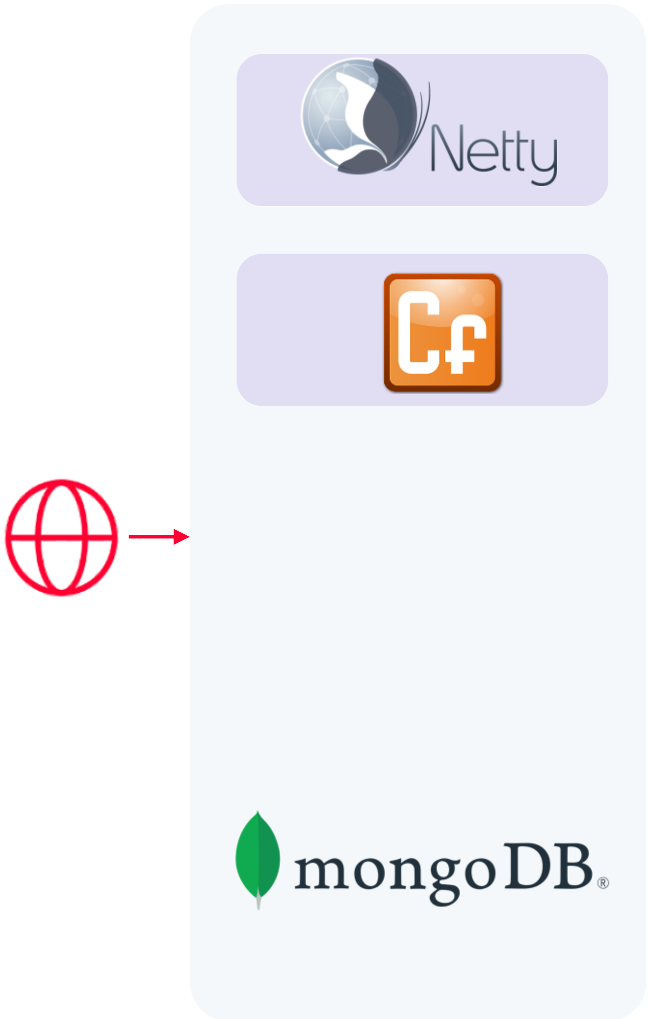
Figure 2: Record to explicit bucket histogram benchmark results.

opentelemetry.io/blog/2024/java-metric-systems-compared



Improving the Vert.x Micrometer Metrics performance

vertx.io/blog/micrometer-metrics-performance



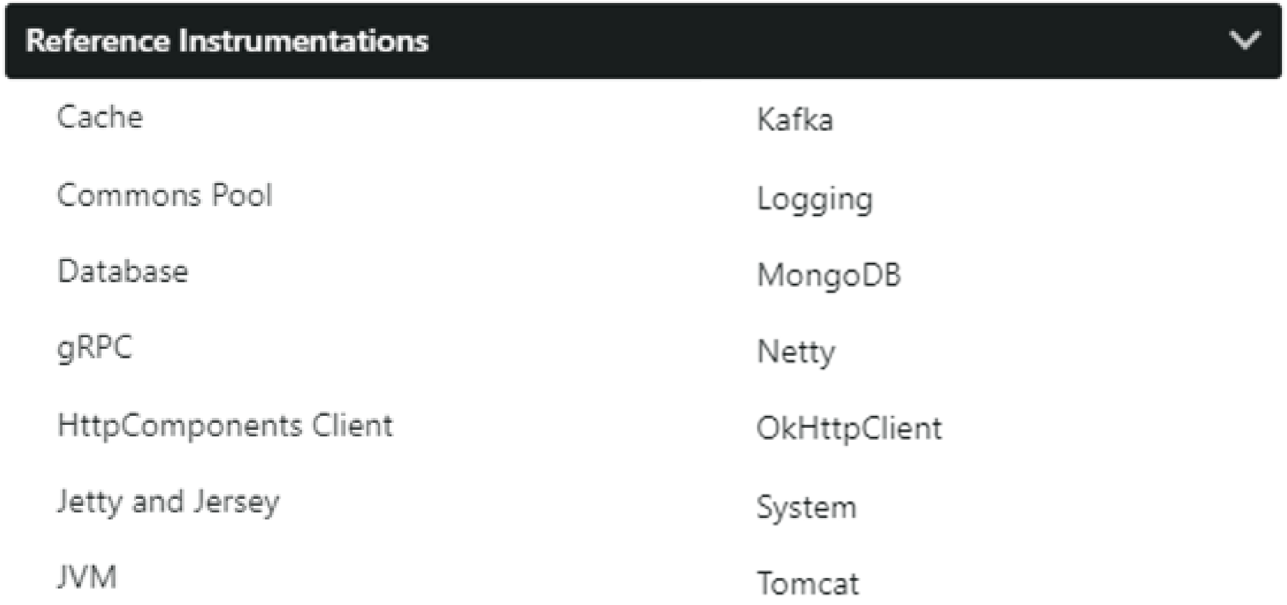
Connectivity

MONGODB

По дефолту поддерживается и настроено

Название - mongodb_driver_commands_seconds

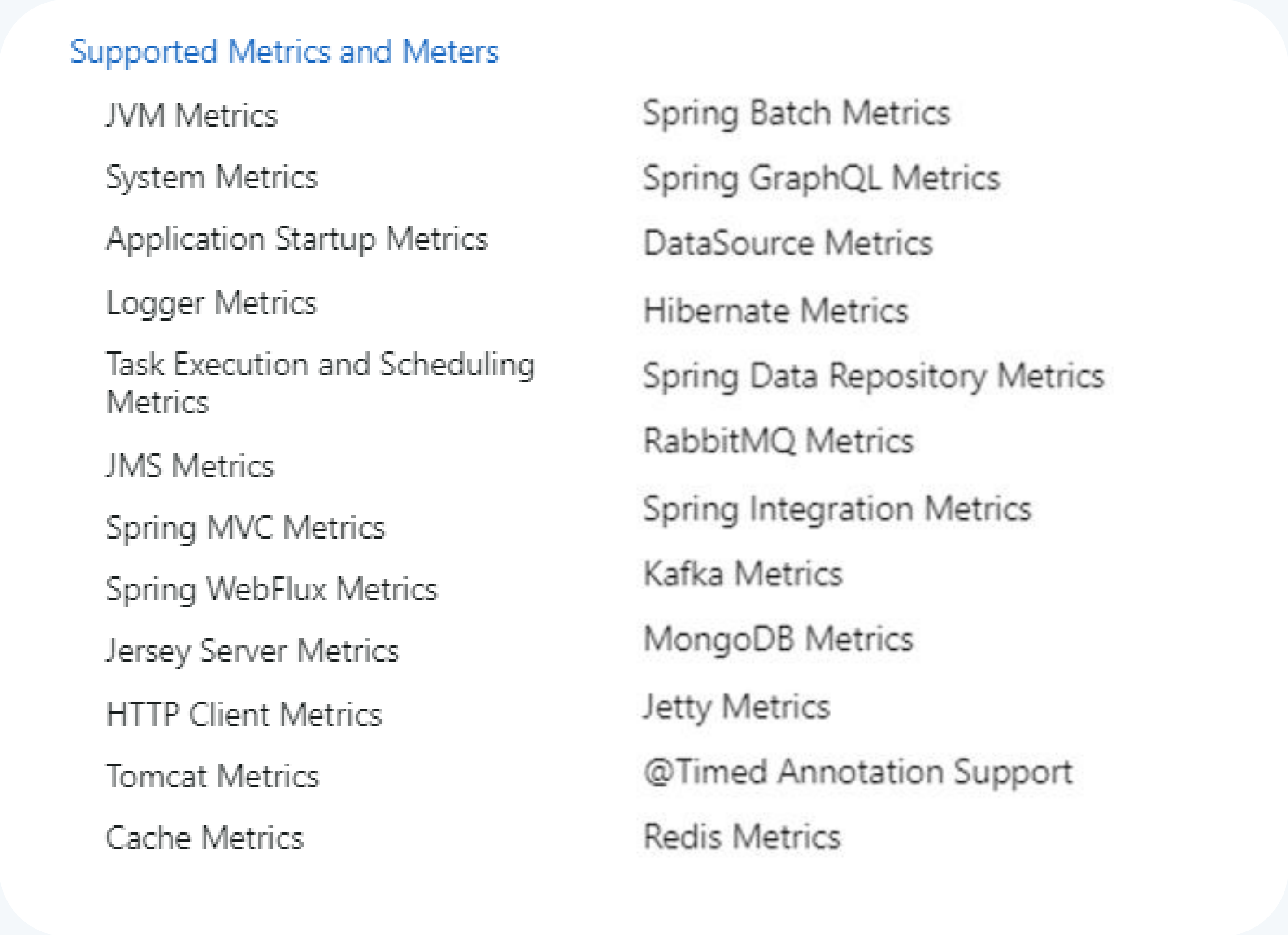
MICROMETER



Reference Instrumentations

- Cache
- Commons Pool
- Database
- gRPC
- HttpComponents Client
- Jetty and Jersey
- JVM
- Kafka
- Logging
- MongoDB
- Netty
- OkHttpClient
- System
- Tomcat

ACTUATOR

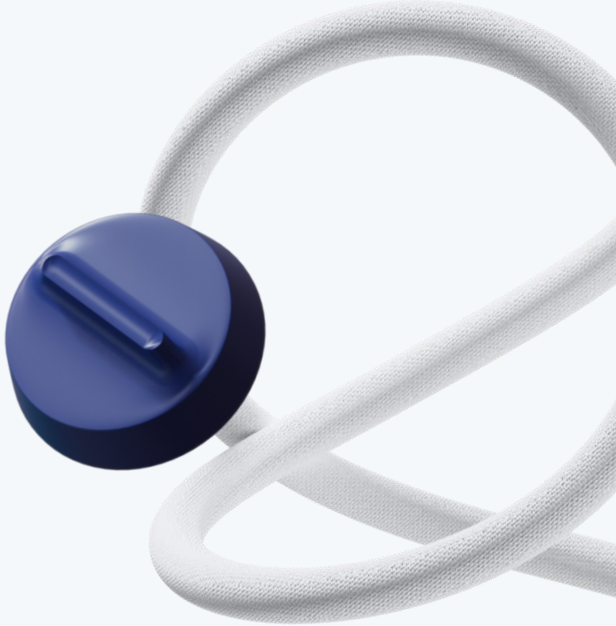


Supported Metrics and Meters

- JVM Metrics
- System Metrics
- Application Startup Metrics
- Logger Metrics
- Task Execution and Scheduling Metrics
- JMS Metrics
- Spring MVC Metrics
- Spring WebFlux Metrics
- Jersey Server Metrics
- HTTP Client Metrics
- Tomcat Metrics
- Cache Metrics
- Spring Batch Metrics
- Spring GraphQL Metrics
- DataSource Metrics
- Hibernate Metrics
- Spring Data Repository Metrics
- RabbitMQ Metrics
- Spring Integration Metrics
- Kafka Metrics
- MongoDB Metrics
- Jetty Metrics
- @Timed Annotation Support
- Redis Metrics

MONGODB

- Status (success)
- Command (find)
- Collection (script)



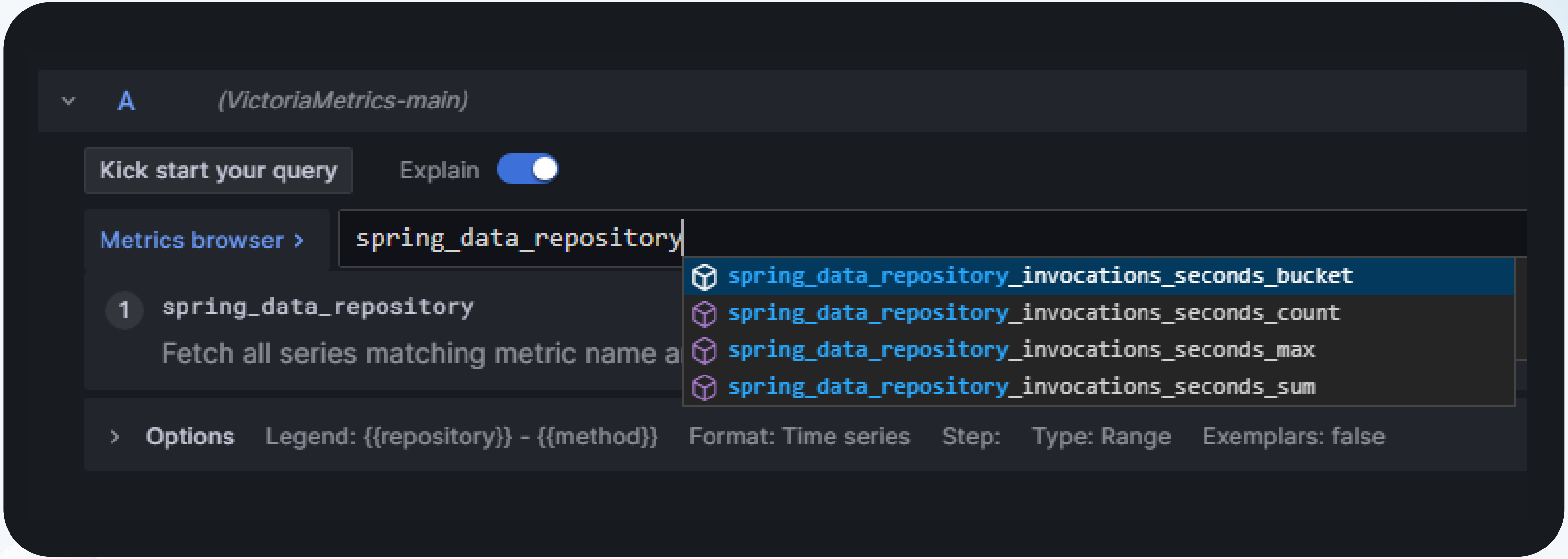
MONGODB

- Mongodb_driver_pool_checkedout
- Mongodb_driver_pool_size
- Mongodb_driver_pool_waitqueuesize

MONGODB

```
 @Bean
 public MongoClientSettingsBuilderCustomizer mongoCustomizer(MeterRegistry registry) {
    return builder -> builder

      .addCommandListener(new MongoMetricsCommandListener(registry))
      .applyToConnectionPoolSettings(b ->
          b.addConnectionPoolListener(new MongoMetricsConnectionPoolListener(registry)));
}
```

SPRING REPOSITORY



```
spring_data_repository
```

- spring_data_repository_invocations_seconds_bucket
- spring_data_repository_invocations_seconds_count
- spring_data_repository_invocations_seconds_max
- spring_data_repository_invocations_seconds_sum

1 spring_data_repository

Fetch all series matching metric name a

Legend: {{repository}} - {{method}} Format: Time series Step: Type: Range Exemplars: false

SPRING REPOSITORY

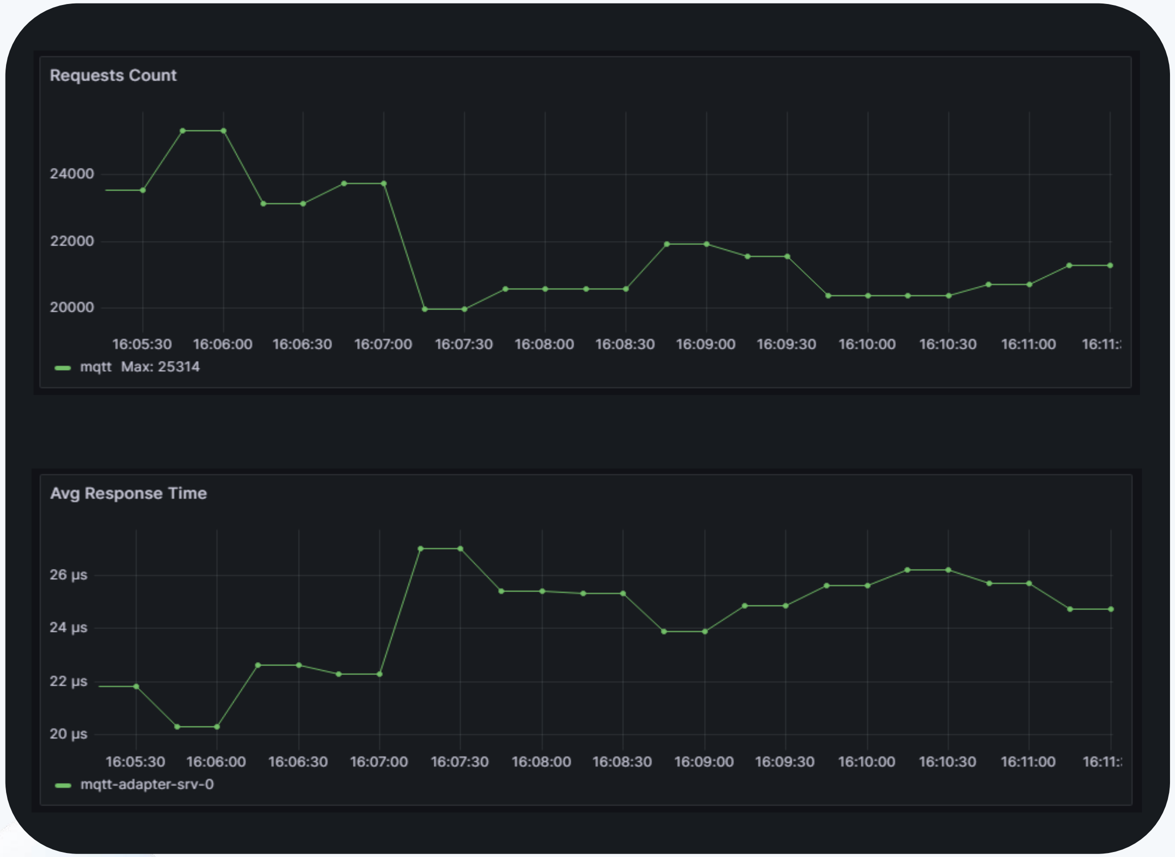
- Documented (Mongo)
- RDBMS (PostgreSQL)
- Wide-column (Scylla)

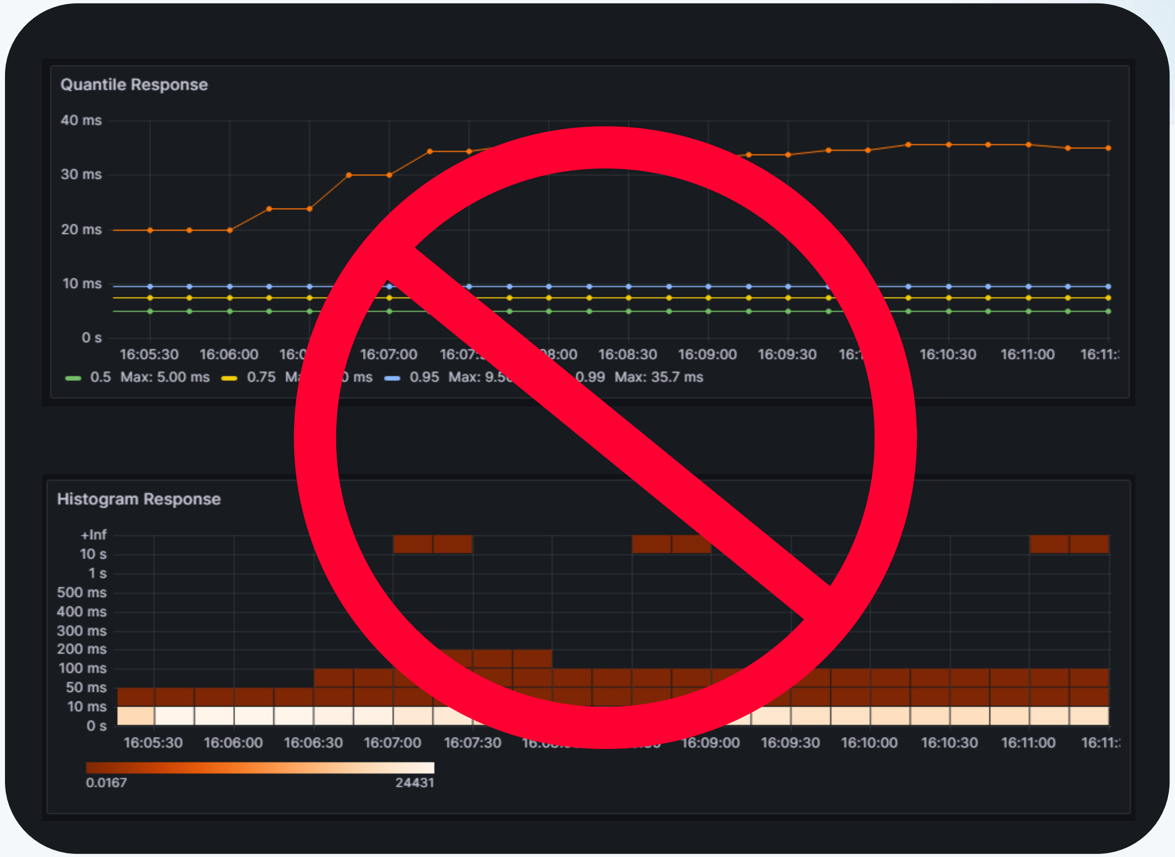
SPRING REPOSITORY

Название - spring_data_repository_invocations

- State (Success)
- Exception (TimeoutException)
- Repository (ExecutionRepository)
- Method (Save)

GRAFANA


Requests Count
24000
22000
20000
16:05:30
16:06:00
16:06:30
16:07:00
16:07:30
16:08:00
16:08:30
16:09:00
16:09:30
16:10:00
16:10:30
16:11:00
16:11:
mqtt Max: 25314
Avg Response Time
26 µs
24 µs
22 µs
20 µs
16:05:30
16:06:00
16:06:30
16:07:00
16:07:30
16:08:00
16:08:30
16:09:00
16:09:30
16:10:00
16:10:30
16:11:00
16:11:
mqtt-adapter-srv-0


Quantile Response
40 ms
30 ms
20 ms
10 ms
0 s
16:05:30
16:06:00
16:0
16:07:00
16:07:
8:00
16:08:30
16:09:00
16:09:30
16:
16:10:30
16:11:00
16:11:
0.5 Max: 5.00 ms
0.75 M
0 ms
0.95 Max: 9.5
0.99 Max: 35.7 ms
Histogram Response
+Inf
10 s
1 s
500 ms
400 ms
300 ms
200 ms
100 ms
50 ms
10 ms
0 s
16:05:30
16:06:00
16:06:30
16:07:00
16:07:30
16:0
16:09:00
16:09:30
16:10:00
16:10:30
16:11:00
16:11:
0.0167
24431

FILTER METRICS

```
public class FeignMetricFilter implements MeterFilter {

  @Override
  public DistributionStatisticConfig configure(Meter.Id id, DistributionStatisticConfig config) {
    if (id.getName().equals("spring.data.repository.invocations")) {
      return DistributionStatisticConfig.builder()
        .serviceLevelObjectives(10, 50, 100, 200, ... 10000)
        .build()
        .merge(config);
    }
    return config;
  }
}
```

PRE-METER CUSTOMIZATION

- management.metrics.enable
- management.metrics.distribution.percentiles-histogram
- management.metrics.distribution.percentiles
- management.metrics.distribution.slo

Netty

Caffeine

mongoDB

Connectivity

CAFFEINE (CACHE)

По дефолту поддерживается, но изначально не настроено

CACHE

Spring doc

Auto-configuration enables the instrumentation of all available Caches on startup with metrics prefixed cache.

CACHE

```
@Bean
public Cache<Object, Object> registerCache() {
  var cb = Caffeine.newBuilder()
      .recordStats()
      .build();
}
```

CACHE

Метрики:

- cache.gets.total
- cache.puts.total
- cache.evictions.total
- cache.size (count)

CACHE

Теги:

- cache (devices)
- cacheManager (caffeine)

CACHE

```
@Bean
public Cache<Object, Object> registerCache() {
  var caffeineStatsCounter = new CaffeineStatsCounter(registry, "cache");

  var cb = Caffeine.newBuilder()
      .recordStats(() -> caffeineStatsCounter)
      .build();
}
```

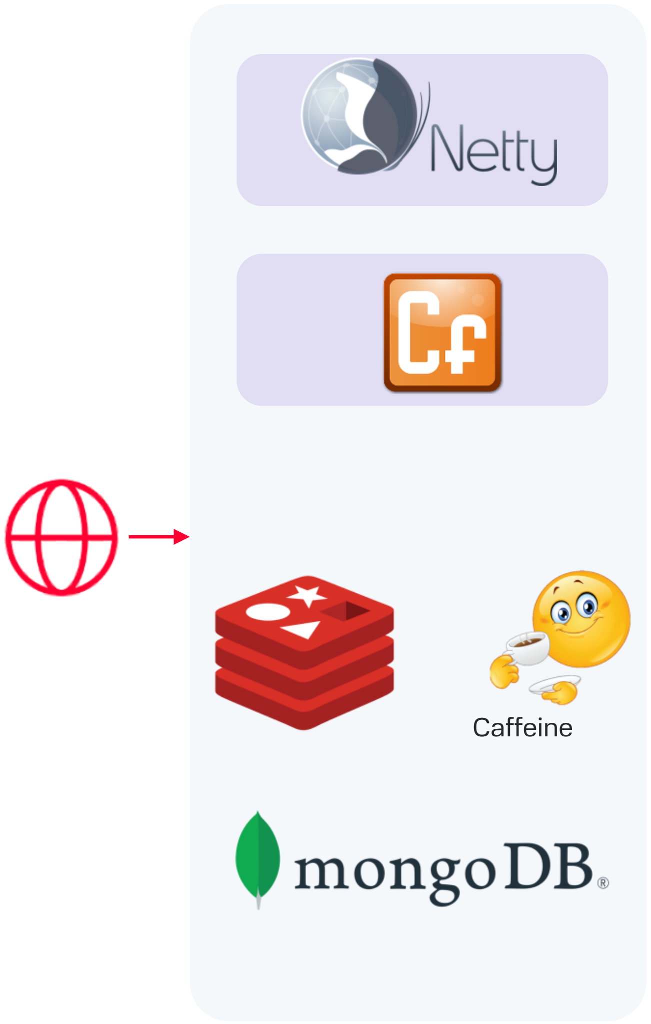
CACHE

```
java.lang.IllegalArgumentException: Failed to register Collector of type
MicrometerCollector: cache_evictions_created is already in use by another
Collector of type MicrometerCollector*
```

CACHE

```
java.lang.IllegalArgumentException: Failed to register Collector of type
MicrometerCollector: cache_evictions_created is already in use by another
Collector of type MicrometerCollector*
```

'Collector already registered that provides name: cache_removals' error at Spring Boot application startup

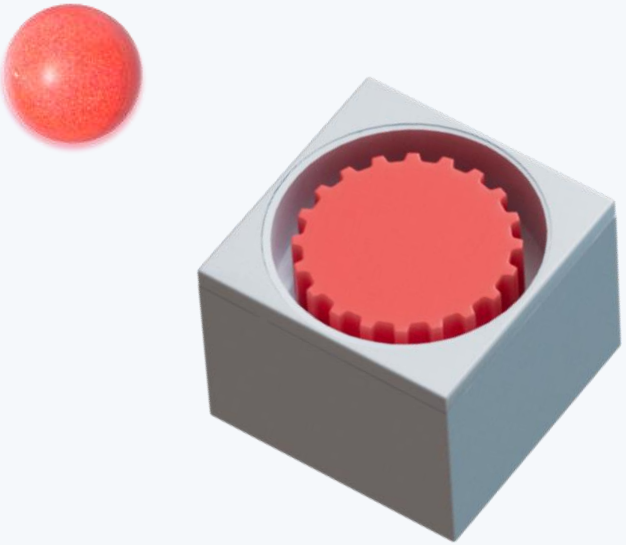


Caffeine

Connectivity

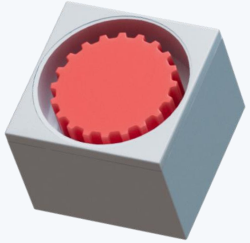
REDIS

По дефолту поддерживается, но не настроено



REDIS

```
@Bean
public RedisCacheManager redisCacheManager(RedisConnectionFactory rcf) {
  return RedisCacheManager.builder(rcf)
      .enableStatistics()
      .build();
}
```



REDIS

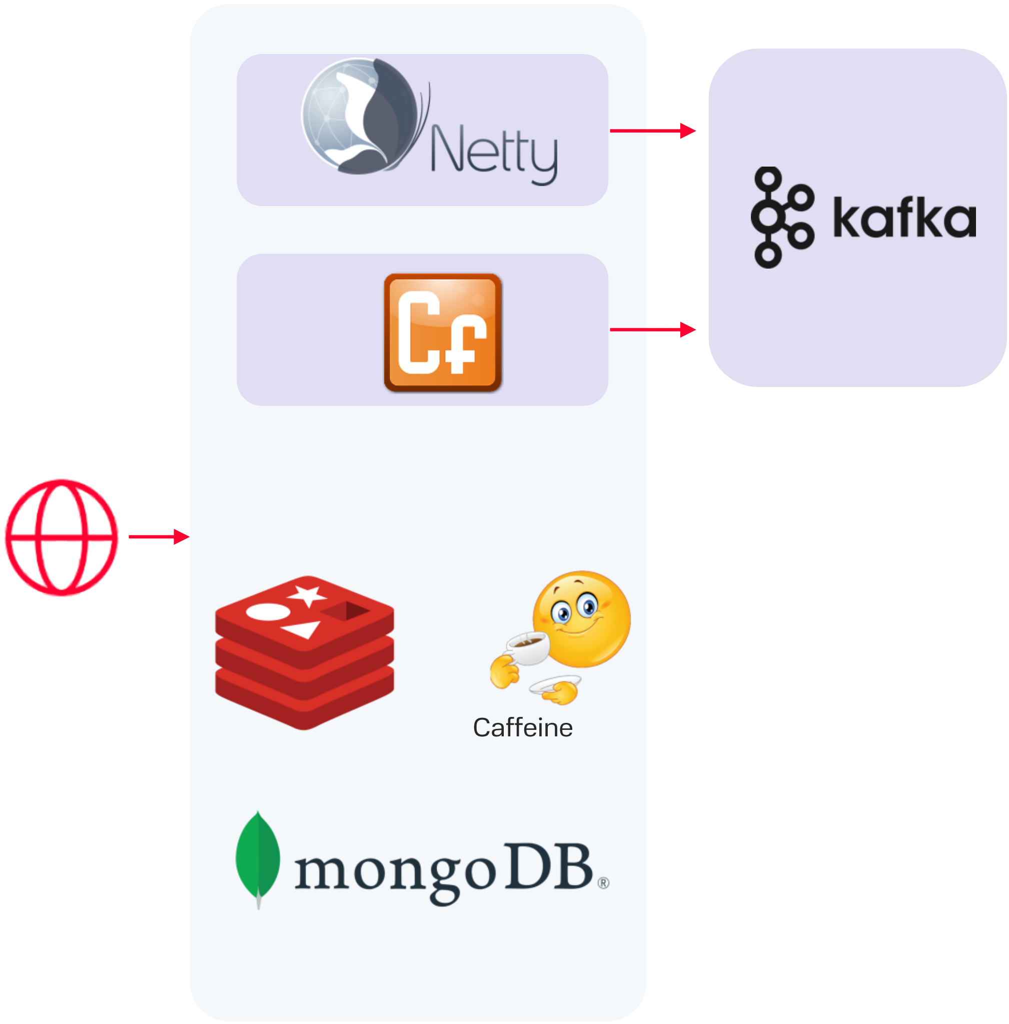
Метрики:

- lettuce.command.completion.seconds
- lettuce.command.firstresponse.seconds

REDIS

Теги:

- command (GET, SET …)
- local (localhost || ANY)
- remote (redis-master)



Connectivity

KAFKA PRODUCER

Название - spring.kafka.template

- result (success)
- exception / error (TimeoutException)
- name (defaultKafkaTemplate)

KAFKA PRODUCER

Инструменты для тегов:

- static tags
- dynamic tags (v. 2.9.8 & 3.0.6)

KAFKA PRODUCER

```
var kafkaTemplate = new KafkaTemplate<>(producerFactory());
kafkaTemplate.setMicrometerTagsProvider(r -> {
            Map<String, String> tags = new HashMap<>(1);
            tags.put("topic", r.topic());
            return tags;
});
```

KAFKA PRODUCER

Название - spring.kafka.template

- result (success)
- exception / error (TimeoutException)
- name (defaultKafkaTemplate)

KAFKA PRODUCER

Название - spring.kafka.template

- result (success)
- exception / error (TimeoutException)
- name (defaultKafkaTemplate)

Spring Boot with Kafka Template with supplied tags provider breaks spring_kafka_template metric

KAFKA PRODUCER ASYNC

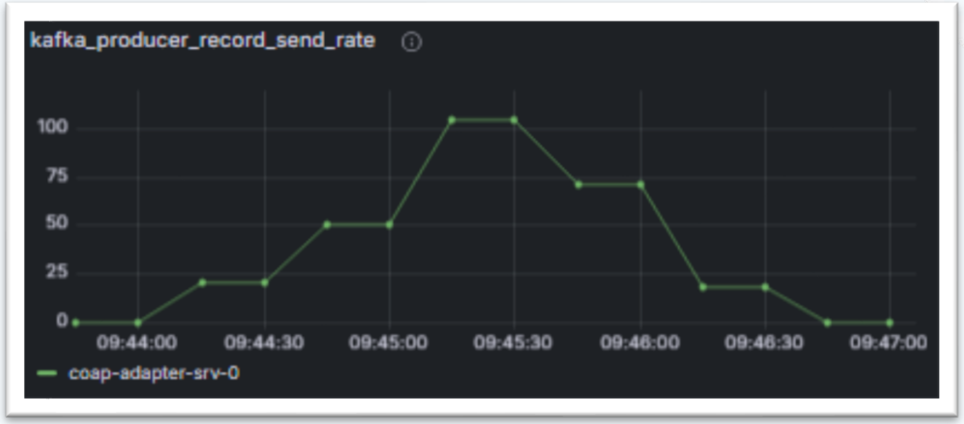
Нет метрик на callback продьюсинга

KAFKA PRODUCER

```
public void sendTelemetry(TelemetryDto dto) {
    long startProcess = System.nanoTime();
    CompletableFuture<SendResult<Object, Object>> future = kafkaTemplate.send(topicName, dto);
            future.whenComplete((result, ex) -> {
                if (e == null) {
                    registerMetric(startProcess, topicName, e);
                } else {
                    registerMetric(startProcess, topicName);
                });
            }
}
```

KAFKA PRODUCER

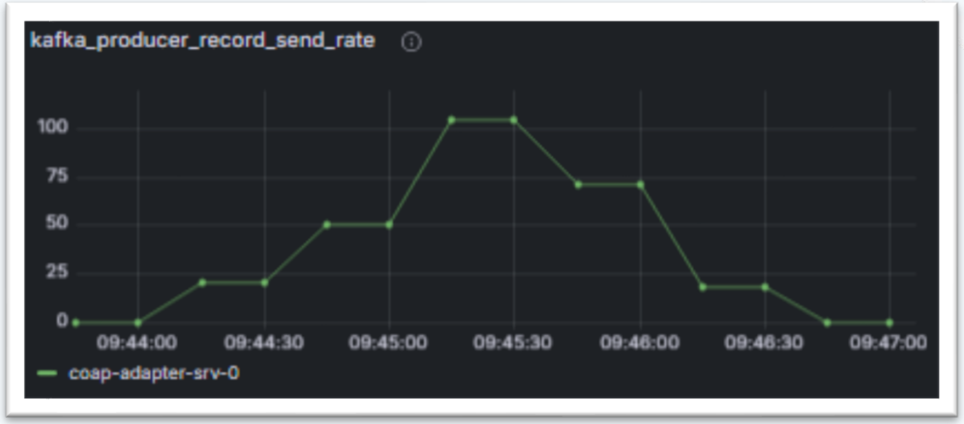
Native metrics



- Их около 60
- Позволяет получить информацию по количеству запросов, по количеству bytes (i/o), latency и т.д

docs.confluent.io/platform/current/kafka/monitoring.html#producer-metrics

KAFKA PRODUCER



Native metrics

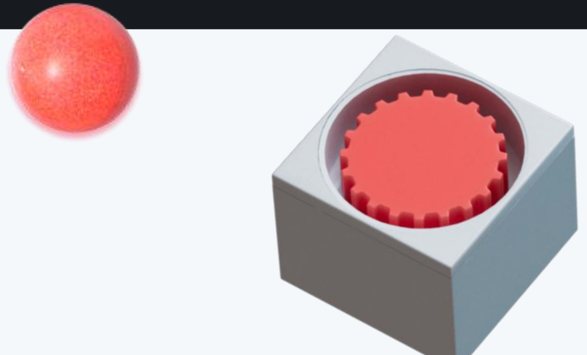
- kafka_producer_record_send_rate
- kafka_producer_request_size_avg
- kafka_producer_compression_rate_avg
- kafka_producer_buffer_available_bytes
- kafka_producer_request_latency_avg
- kafka_producer_requests_in_flight

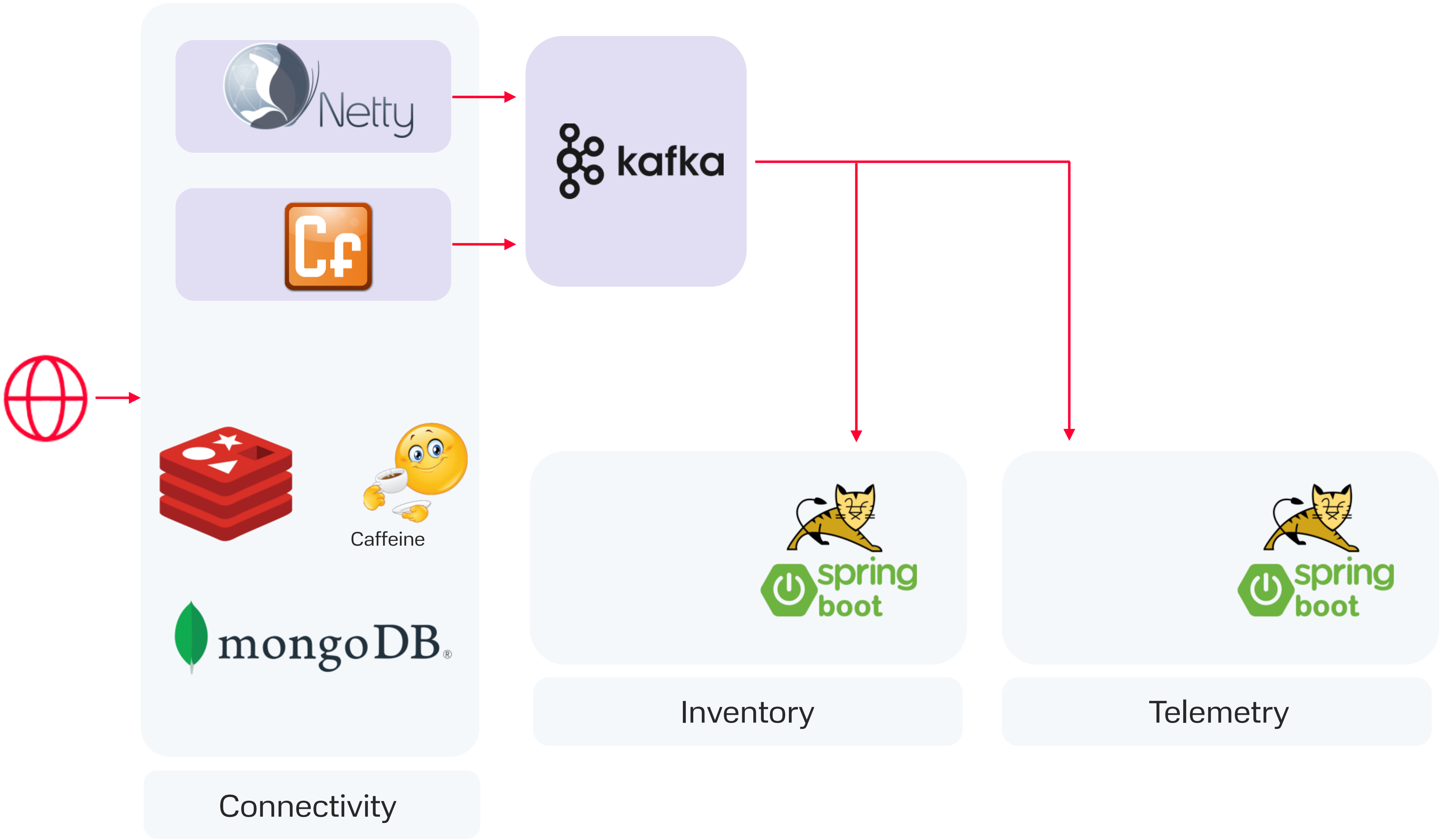
KAFKA PRODUCER by Micrometer

```
@Bean
public ProducerFactory<String, Object> producerFactory(
                       ObjectProvider<DefaultKafkaProducerFactoryCustomizer> customizers) {

   Map<String, Object> props = new HashMap<>();
    ...
   DefaultKafkaProducerFactory<String, Object> factory = new DefaultKafkaProducerFactory<>(props);
   customizers.orderedStream().forEach(c -> c.customize(factory));
   return factory;
}
```

Где будут созданы:

- *MicrometerProducerListener*
- *KafkaProducerMetrics*





Inventory

Telemetry

Connectivity

KAFKA CONSUMER

Название - spring.kafka.listener

- result (success)
- exception / error (ListenerExecutionFailedException)
- name (e62a4464-6fd7-466a-bdce-2b48e1434f69)

KAFKA CONSUMER

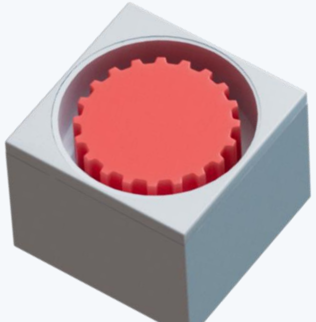
```
@KafkaListener(id = "#{'${spring.kafka.incoming.topic}'}",
  topics = "#{'${spring.kafka.incoming.topic}'}",
  groupId = "#{'${spring.kafka.consumer.group-id}'}",
  idIsGroup = false)
public void listen(List<ConsumerRecord<String, Object>> messages) {
  for (var consumerRecord : messages) {
            ...
  }
}
```

KAFKA CONSUMER

```
@KafkaListener(id = "#{'${spring.kafka.incoming.topic}'}",
  topics = "#{'${spring.kafka.incoming.topic}'}",
  groupId = "#{'${spring.kafka.consumer.group-id}'}",
  idIsGroup = false)
public void listenAction(List<ConsumerRecord<String, Object>> messages) {
  for (var consumerRecord : messages) {
            ...
  }
}
```

KAFKA CONSUMER

```
@KafkaListener(id = "#{'${spring.kafka.incoming.topic}'}",
  topics = "#{'${spring.kafka.incoming.topic}'}",
  groupId = "#{'${spring.kafka.consumer.group-id}'}",
  idIsGroup = false)
public void listenAction(List<ConsumerRecord<String, Object>> messages) {
  for (var consumerRecord : messages) {
              ...
  }
}
```



KAFKA CONSUMER

```
@KafkaListener(id = "#{'${spring.kafka.incoming.topic}'}",
  topics = "#{'${spring.kafka.incoming.topic}'}",
  groupId = "#{'${spring.kafka.consumer.group-id}'}",
  idIsGroup = false)
public void listenAction(List<ConsumerRecord<String, Object>> messages) {
  for (var consumerRecord : messages) {
            ...
  }
}
```

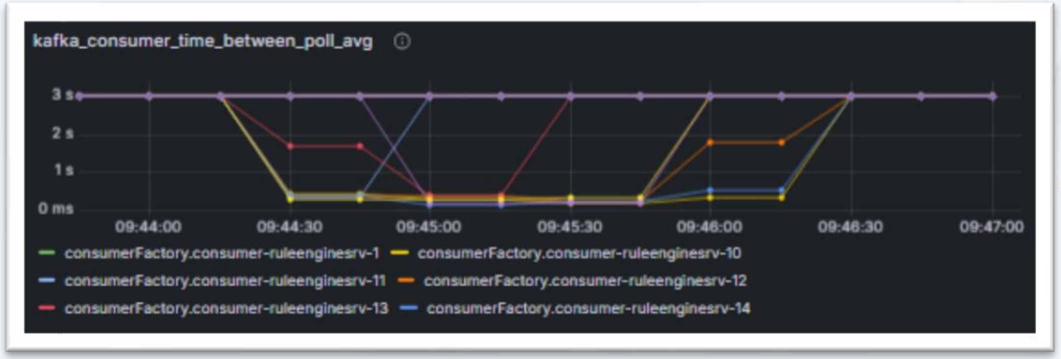
KAFKA CONSUMER

Название - spring.kafka.listener

- result (success)
- exception / error (ListenerExecutionFailedException)
- name (action-proc-srv.topic-0)

KAFKA CONSUMER

Native metrics

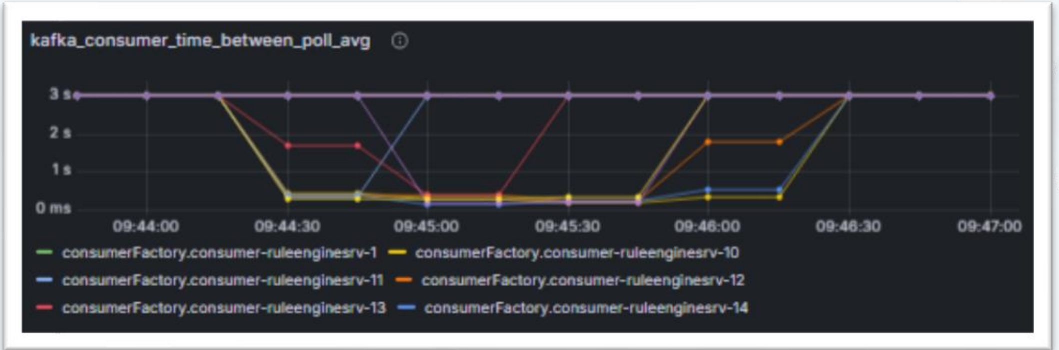


- Их около 60
- Позволяет получить информацию по количеству запросов, по количеству bytes (i/o), latency и т.д

docs.confluent.io/platform/current/kafka/monitoring.html#consumer-metrics

KAFKA CONSUMER

Native metrics



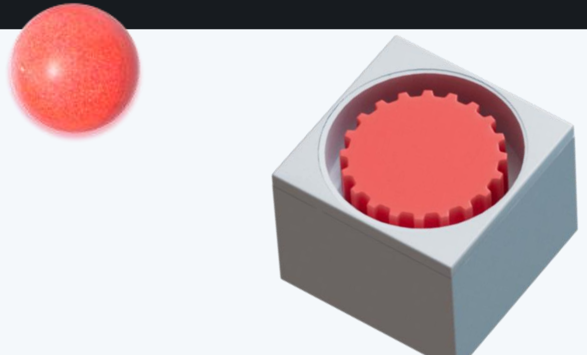
- kafka_consumer_fetch_manager_records_per_request_avg
- kafka_consumer_fetch_manager_fetch_size_avg
- kafka_consumer_time_between_poll_avg
- kafka_consumer_fetch_manager_records_lag_avg

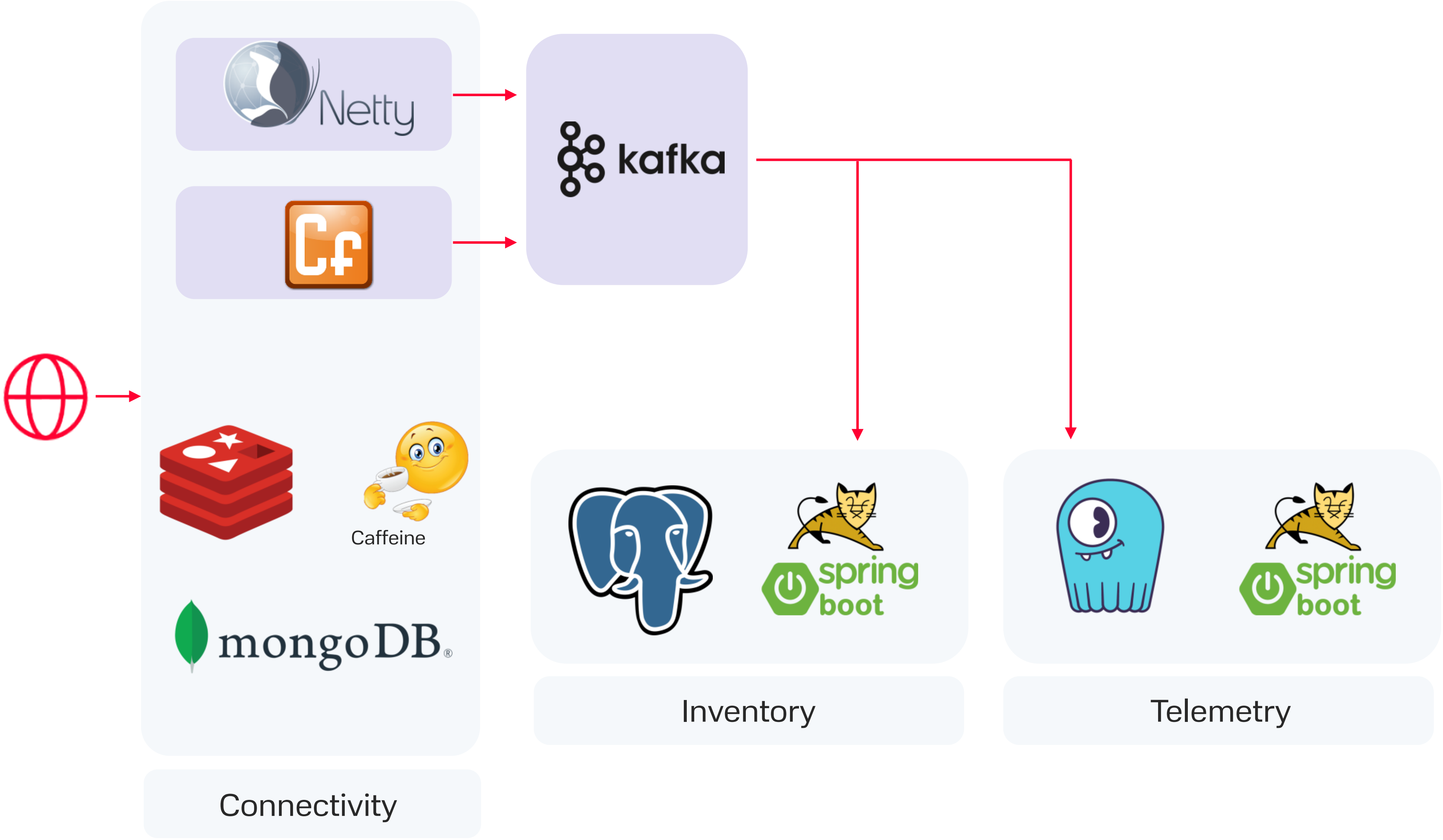
KAFKA CONSUMER by Micrometer

```
@Bean
public ConsumerFactory<String, String> consumerFactory(
                        ObjectProvider<DefaultKafkaConsumerFactoryCustomizer> customizers) {

   Map<String, Object> props = new HashMap<>();
    ...
   DefaultKafkaConsumerFactory<String, Object> factory = new DefaultKafkaConsumerFactory<>(props);
   customizers.orderedStream().forEach(c -> c.customize(factory));
   return factory;
}
```

Где будут созданы:

- *MicrometerConsumerListener*
- *KafkaConsumerMetrics*





Connectivity

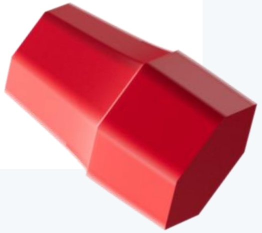
Inventory

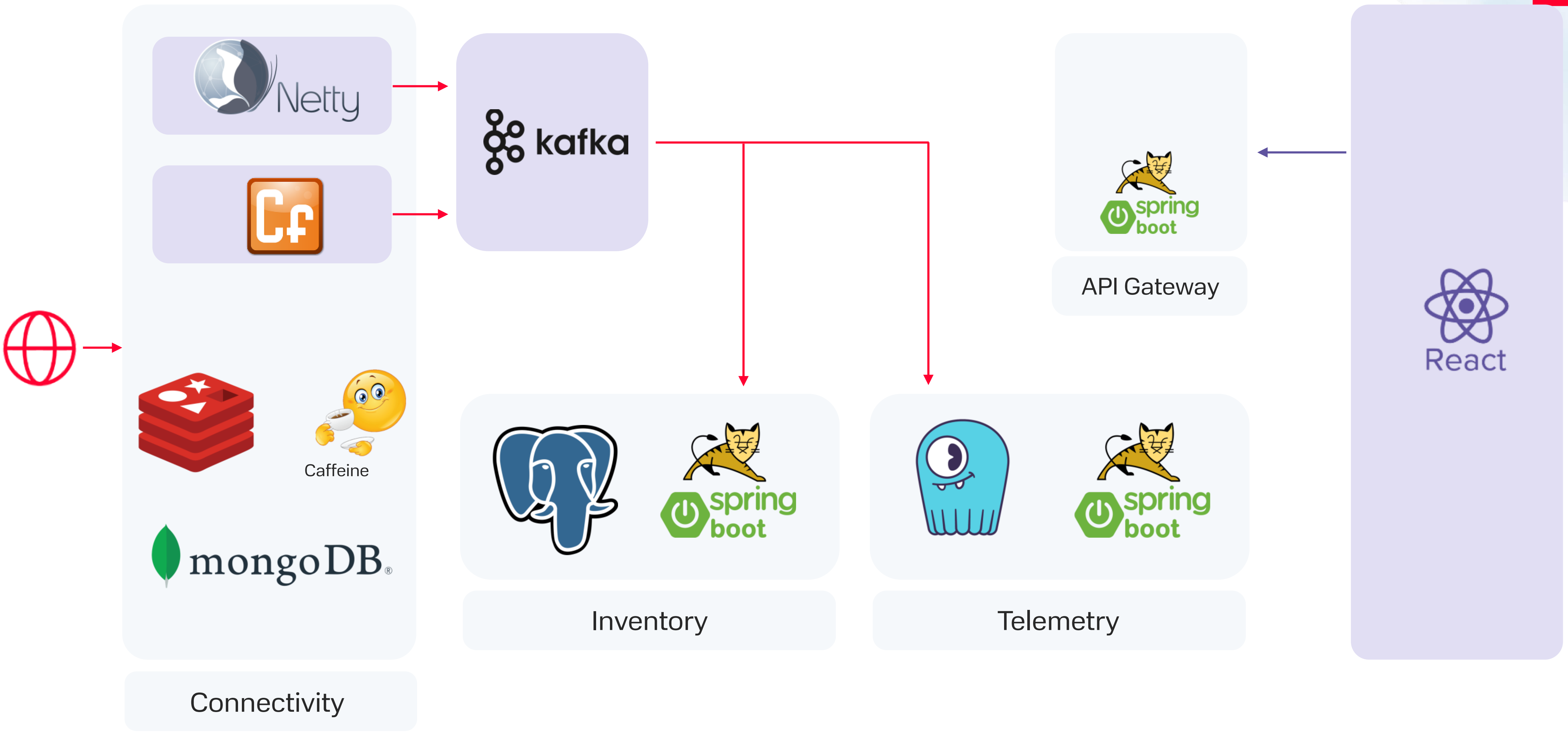
Telemetry

SPRING REPOSITORY

Название - spring_data_repository_invocations

- NoSql (Mongo)
- RDBMS (PostgreSQL)
- Wide-column (Scylla)





Caffeine

Connectivity

Inventory

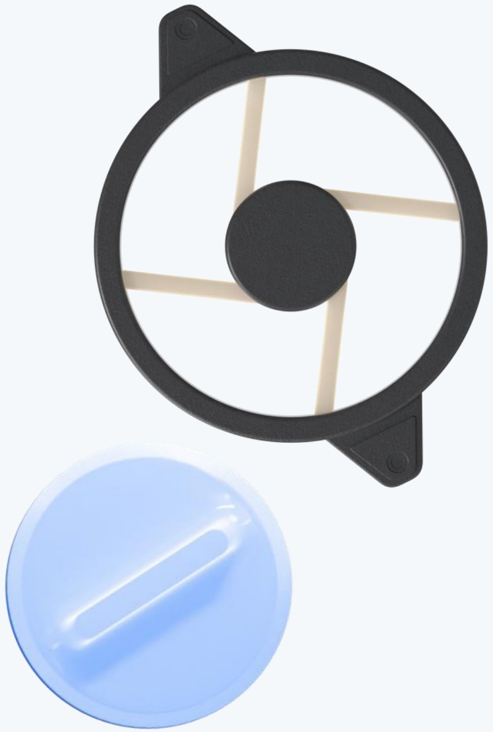
Telemetry

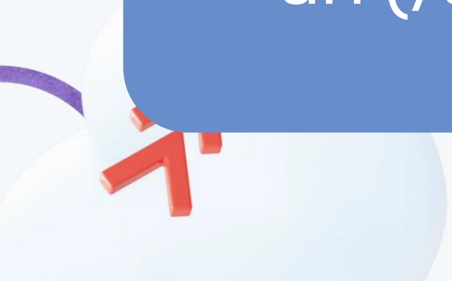
API Gateway

HTTP SERVER

Название - http.server.requests

- exception (TimeoutException)
- method (POST)
- outcome (SUCCESS)
- status (200)
- uri (/api/device)





HTTP SERVER (JSON RPC)

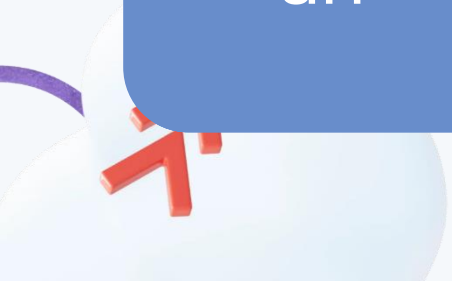
Название - http.server.requests

- exception
- method
- outcome
- status
- uri

HTTP SERVER (JSON RPC)

Название - http.server.requests

- exception
- method (всегда **POST**)
- outcome
- status
- uri



HTTP SERVER (JSON RPC)

Название - http.server.requests

- exception
- method (всегда **POST**)
- outcome (всегда **SUCCESS**)
- status
- uri

HTTP SERVER (JSON RPC)

Название - http.server.requests

- exception
- method (всегда **POST**)
- outcome (всегда **SUCCESS**)
- status (всегда **200**)
- uri

HTTP SERVER (JSON RPC)

Название - http.server.requests

- exception
- method (всегда **POST**)
- outcome (всегда **SUCCESS**)
- status (всегда **200**)
- uri (всегда **одинаковый**)

HTTP SERVER (JSON RPC)

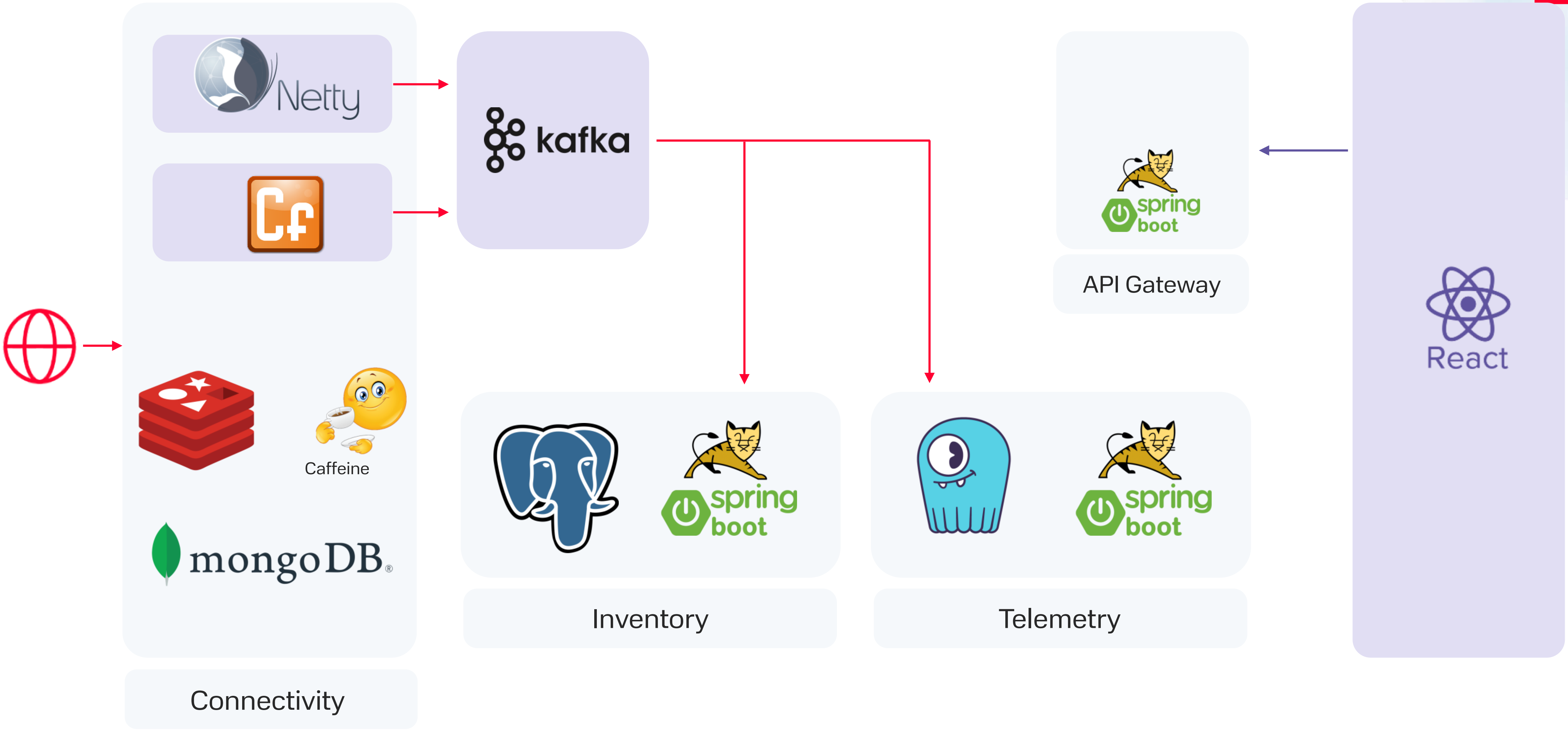
WebMvcTagsProvider

HTTP SERVER (JSON RPC)

```
@Configuration
public class WebMvcCustomTagsProvider implements WebMvcTagsProvider {

  @Override
  public Iterable<Tag> getTags(HttpServletRequest request, ...) {

            ...

    if (contextHolder.getCurrentContext() != null) {
      String jsonRpcMethod = contextHolder.getCurrentContext().getJsonRpcMethod();
      if (jsonRpcMethod != null) {
        tags = tags.and(Tag.of(JSON_RPC_METHOD_TAG, jsonRpcMethod));
      }
    }

    return tags;
  }
```

TAGS PROVIDERS

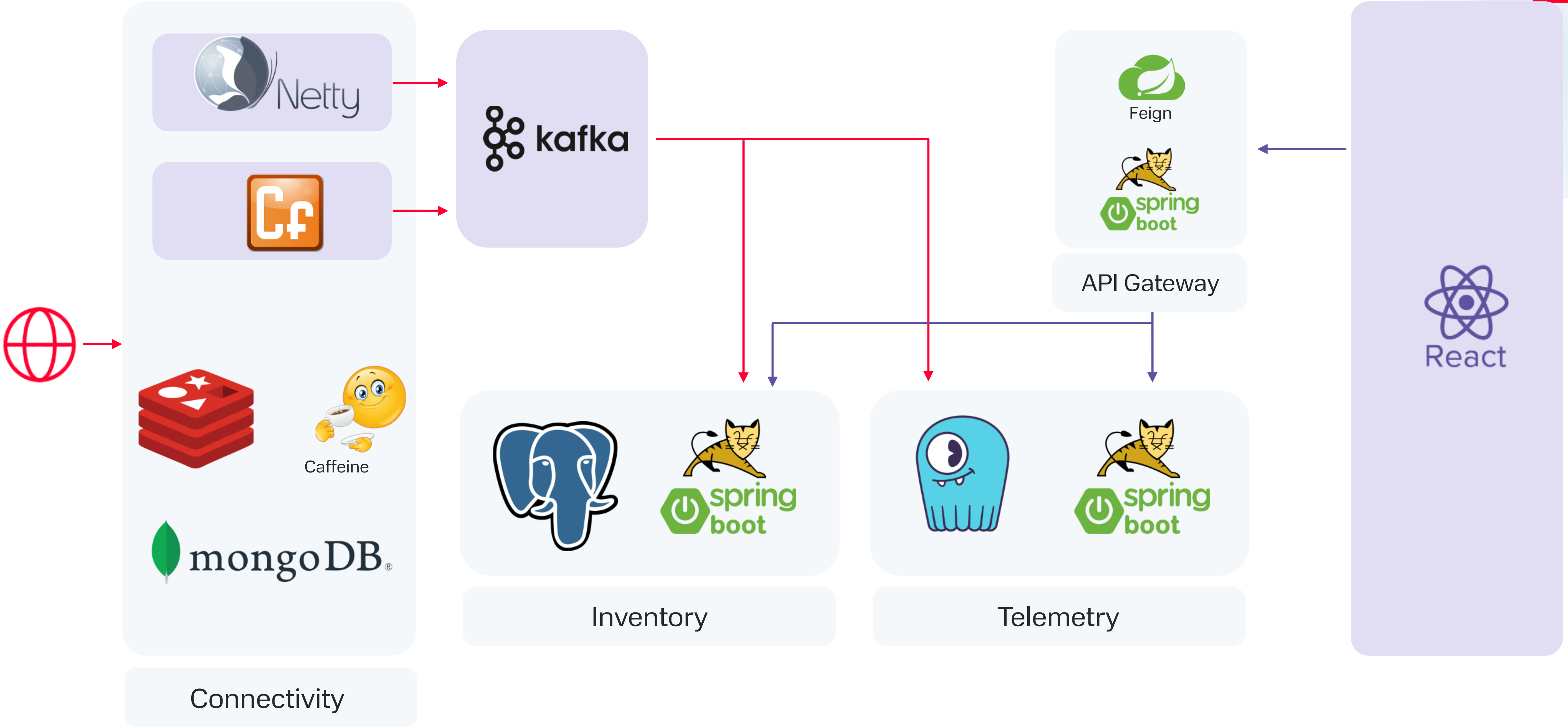
- WebMvcTagsProvider
- WebFluxTagsProvider
- RestTemplateExchangeTagsProvider
- WebClientExchangeTagsProvider



Connectivity

Inventory

Telemetry



Feign

API Gateway

Caffeine

Inventory

Telemetry

Connectivity

HTTP CLIENT

Название - http.client.requests

- exception
- method
- status
- uri
- clientName (auth-srv)

HTTP CLIENT

Название - http.client.requests

- exception
- method
- status
- uri
- clientName (auth-srv)

Пустые метрики

FEIGN CLIENT

По дефолту не поддерживается

FEIGN CLIENT

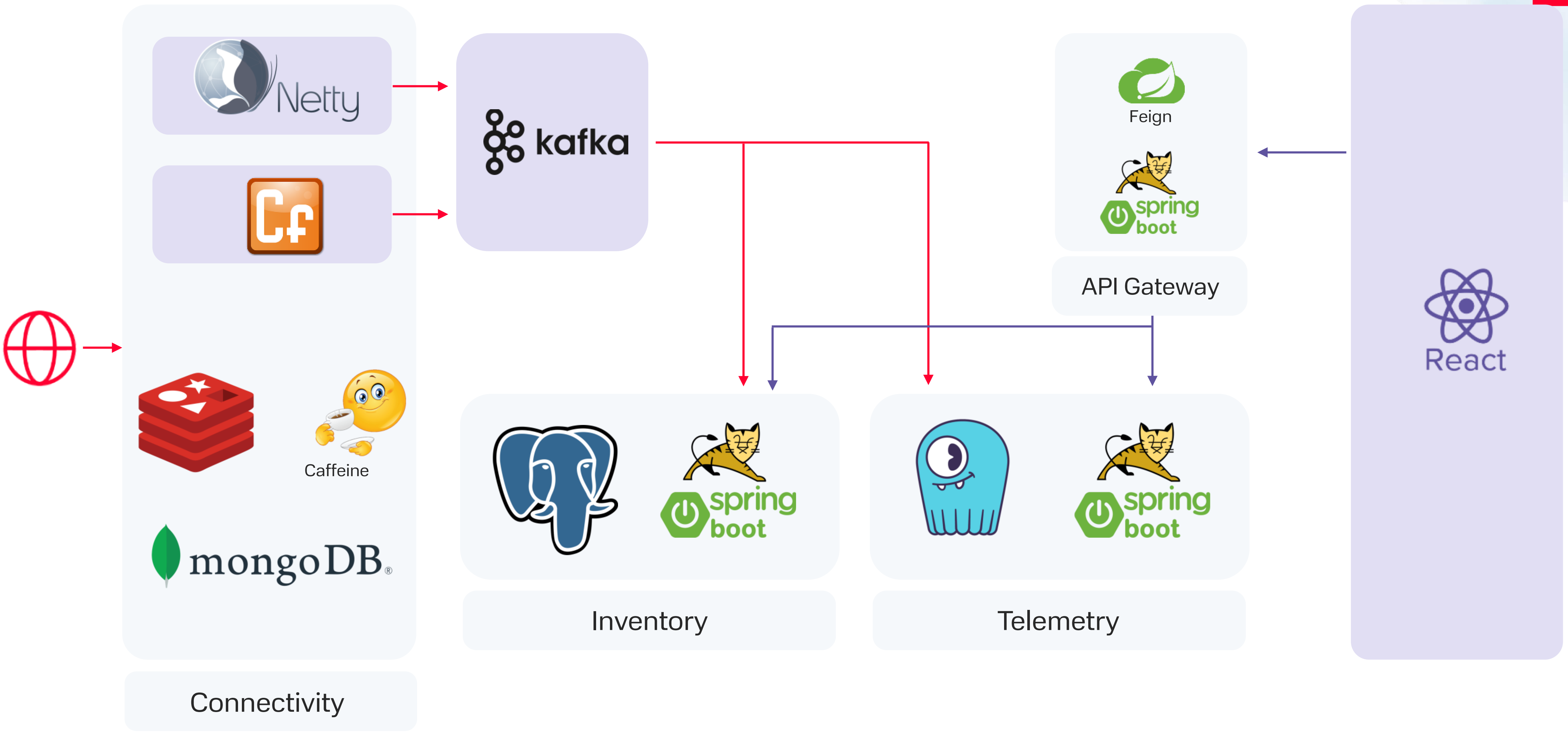
По дефолту не поддерживается

```
implementation 'io.github.openfeign:feign-micrometer'
```

FEIGN CLIENT

Название - feign.Client

- + тоже самое, что и RestTemplate
- host (= clientName)
- client (ru.mts.iot.core.auth.service.feign.AuthClient)



Feign

API Gateway

Caffeine

Inventory

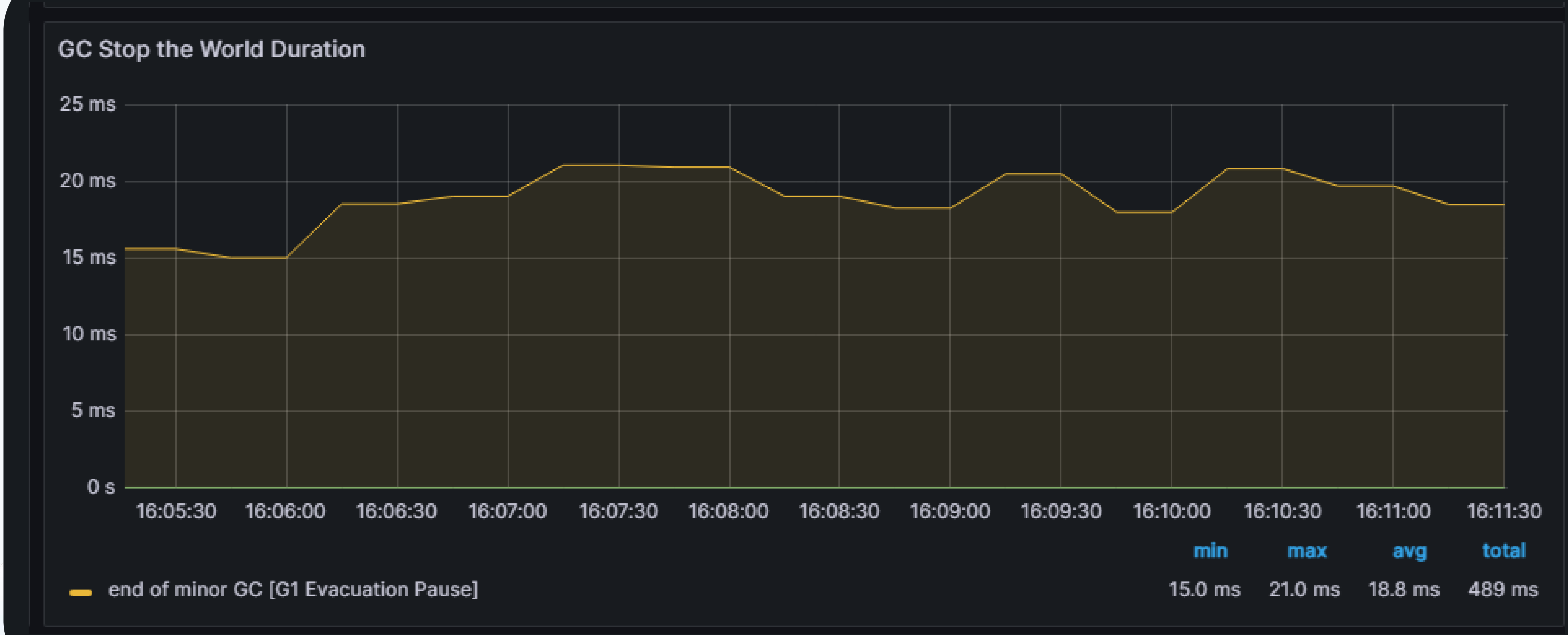
Telemetry

Connectivity

GC OVERHEAD

Можно ли нивелировать погрешность от срабатывания GC для метрик?

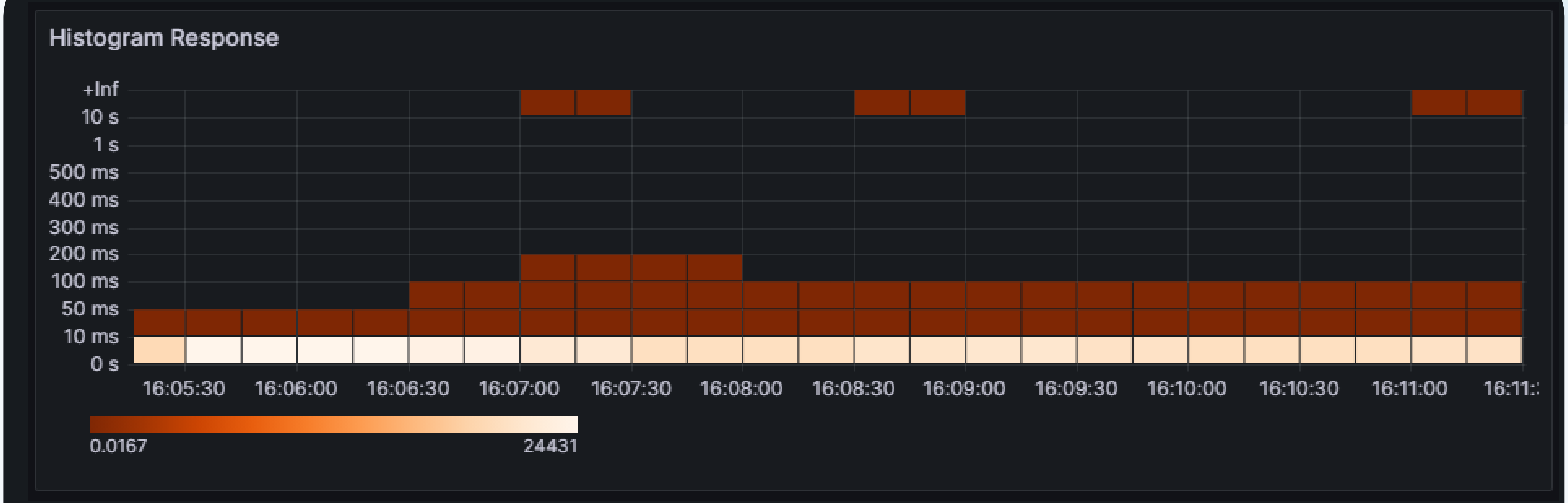
GC OVERHEAD



	min	max	avg	total
▬ end of minor GC [G1 Evacuation Pause]	15.0 ms	21.0 ms	18.8 ms	489 ms

Coordinated Omission problem

Это любая остановка в масштабах всего процесса или всей системы.



```
sum(
     rate(netty_custom_server_requests_seconds_bucket{namespace="$namespace", job="$service",pod=~"$pod"}
     [$__rate_interval]))
by (le))
```

API

```
private final MeterRegistry registry;

public void registerTimeMetrics(double requestTime, String exceptionName) {
   DistributionSummary
       .builder("netty_custom_server_requests")
       .baseUnit("seconds")
       .description("Summary of netty server requests")
       .tag("exception", exceptionName)
       .serviceLevelObjectives(10, 50, 100, 200, ... 10000)
       .register(registry)
       .record(requestTime);
}
```

API

```
private final MeterRegistry registry;

public void registerTimeMetrics(long requestTime, String exceptionName) {
  Timer
      .builder("netty_custom_server_requests")
      .description("Summary of netty server requests")
      .tag("exception", exceptionName)
      .serviceLevelObjectives(Duration.ofMillis(10), ...)
      .register(registry)
      .record(Duration.ofMillis(requestTime));
}
```

API

```
private final MeterRegistry registry;

public void registerTimeMetrics(long requestTime, String exceptionName) {
  Timer
      .builder("netty_custom_server_requests")
      .description("Summary of netty server requests")
      .tag("exception", exceptionName)
      .serviceLevelObjectives(Duration.ofMillis(10), ...)
      .register(registry)
      .record(Duration.ofMillis(requestTime));
}
```

TIMER VS DISTRIBUTIONSUMMARY

- По своей сути они оба Summary
- Timer более узкоспециализированная версия, которая заточена на регистрацию времени
- Timer имеет pause detector

API

```
private final MeterRegistry registry;
private final PauseDetector detector;

public void registerTimeMetrics(long requestTime, String exceptionName) {
  Timer
     .builder("netty_custom_server_requests")
     .description("Summary of netty server requests")
     .tag("exception", exceptionName)
     .serviceLevelObjectives(Duration.ofMillis(10), ...)
     .pauseDetector(detector)
     .register(registry)
     .record(Duration.ofMillis(requestTime));
}
```

API

```
private final MeterRegistry registry;

/**
 * Рекомендованные значения выбраны из документации:
 * "100ms for both values is a reasonable default
 * to offer decent detection of long pause events
 * while consuming a negligible amount of CPU time" (c)
*/
@Bean
public PauseDetector clockDriftPauseDetector() {
  var detector = new ClockDriftPauseDetector(ofMillis(100), ofMillis(100));
  registry.config().pauseDetector(pauseDetector);
  return detector;
}
```

Pause Detector

- Видимых преимуществ при его использовании замечено не было

Pause Detector

- Видимых преимуществ при его использовании замечено не было

Pause Detection contributes to count when it should just contribute to total time

Switch default pause detector to NoPauseDetector

Миллисекунды

А правильно ли измеряем время?

Time

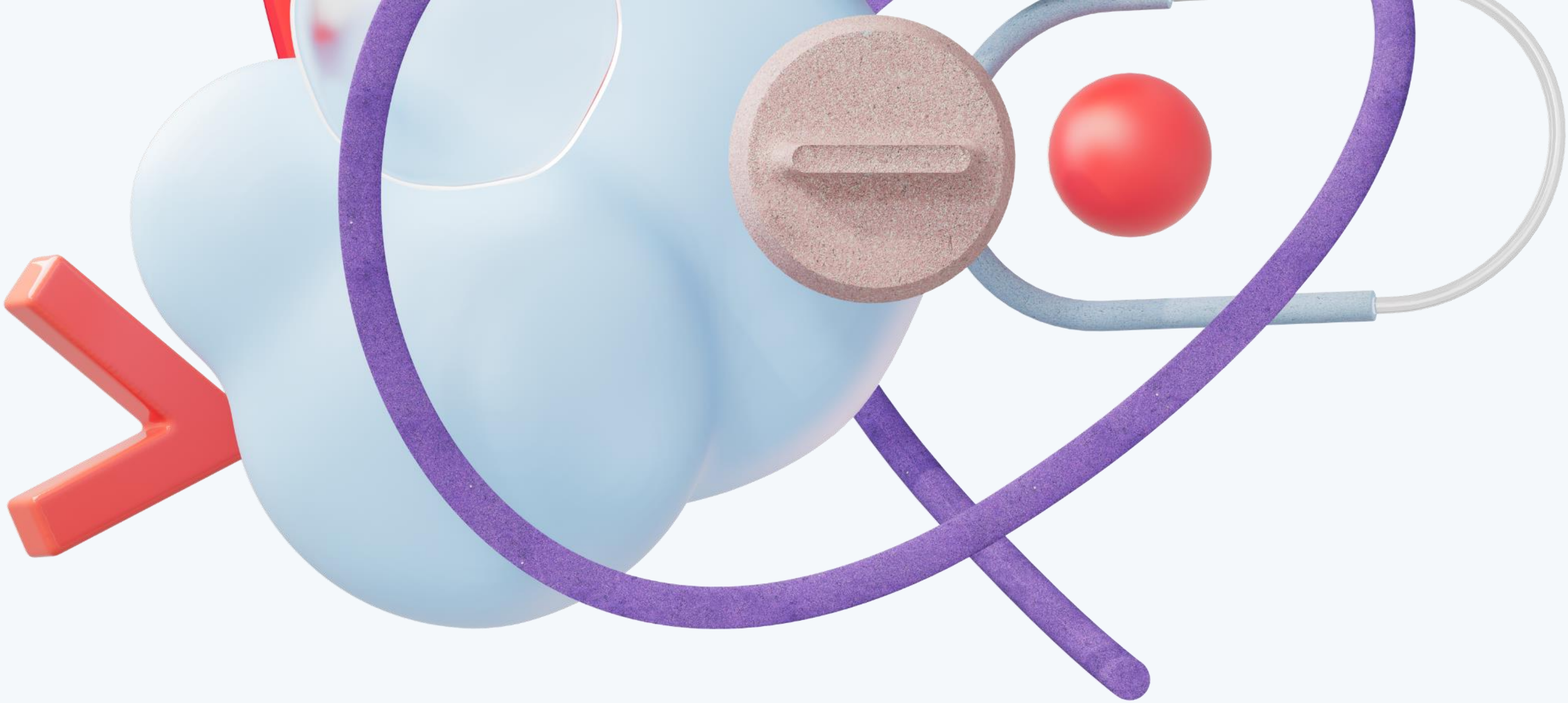
- System.currentTimeMillis()
- System.nanoTime()

Time

- System.currentTimeMillis() - wall-clock time
- System.nanoTime() - monotonic clocks

ВЫВОДЫ

- Micrometer – довольно простой и эффективный инструмент для мониторинга и он так же предоставляет множество дефолтных метрик
- Присутствуют некоторые проблемы с инструментированием (caffeine cache, kafka producer, pause detector)
- Overhead от метрик имеется, но на сколько он для вас критичный - нужно исходить из предметной области



МТС True Tech +

ВОПРОСЫ

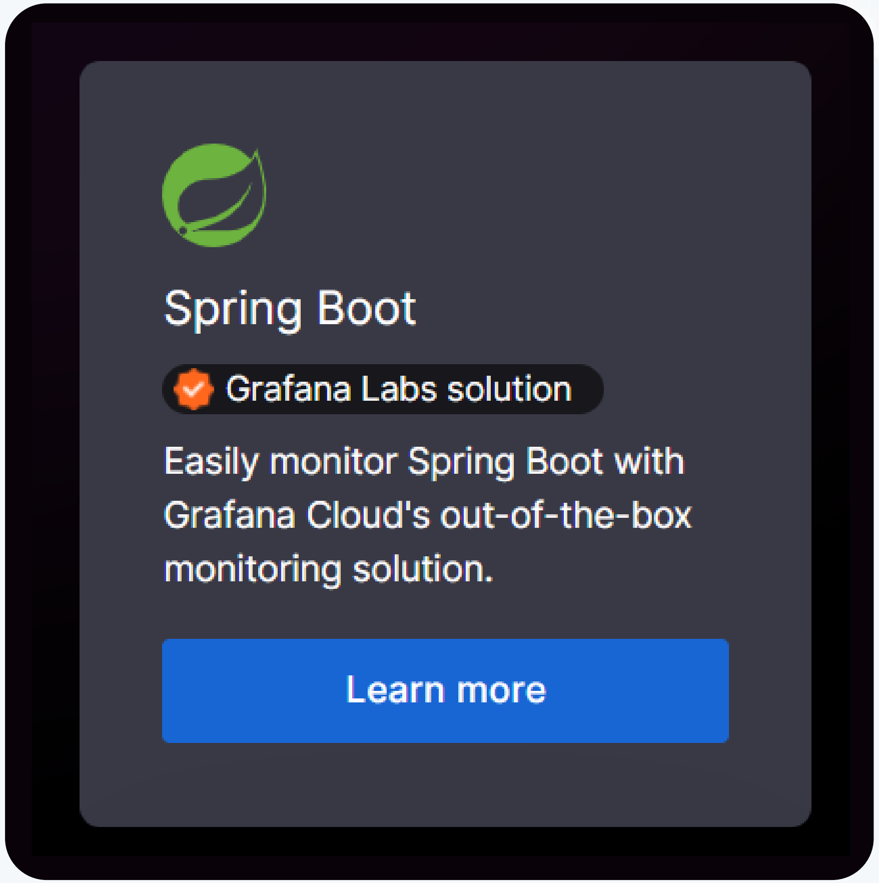
Сурен Калайчьян

Backend developer

Telegram: @websockets

SPRING BOOT DASHBOARD

Grafana Labs



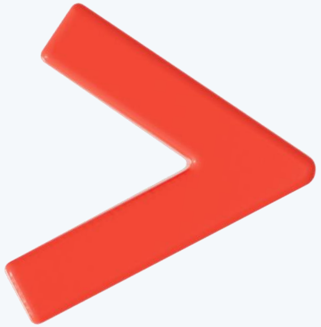
Spring Boot

Grafana Labs solution

Easily monitor Spring Boot with Grafana Cloud's out-of-the-box monitoring solution.

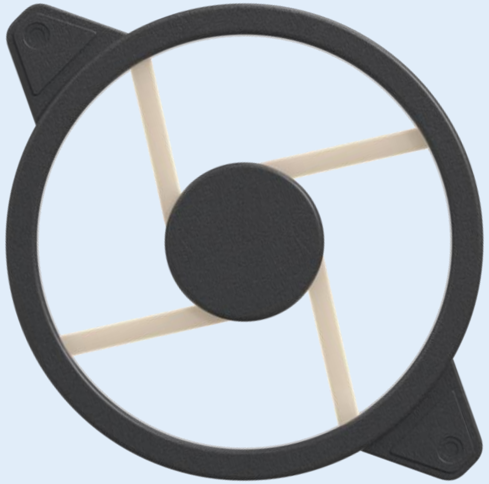
CALIFORNIUM (COAP PROTOCOL)

- `System.currentTimeMillis();`
- `System.nanoTime();`



TOMCAT

Tags implement



GAUGE

```java
private final MeterRegistry registry;
private final Map<String, Double> metricValues;

public void execute(Double value, Tags tags, String metricName, String description) {
  String metricFullName = MetricUtils.getMeterFullName(metricName, tags.stream().toList());

  if (!metricValues.containsKey(metricFullName)) {
     metricValues.put(metricFullName, value);
     buildMetric(tags, metricName, metricFullName, description);
  } else {
     metricValues.put(metricFullName, value);
  }
}

public void buildMetric(Tags tags, String metricName, String metricFullName, String description) {
  Gauge.builder(metricName, metricValues, c -> c.get(metricFullName))
     .description(description)
     .tags(tags)
     .register(registry);
}
```

DISCLAIMER

Большинство дефолтных метрик ориентированы на Timer и Gauge

Time

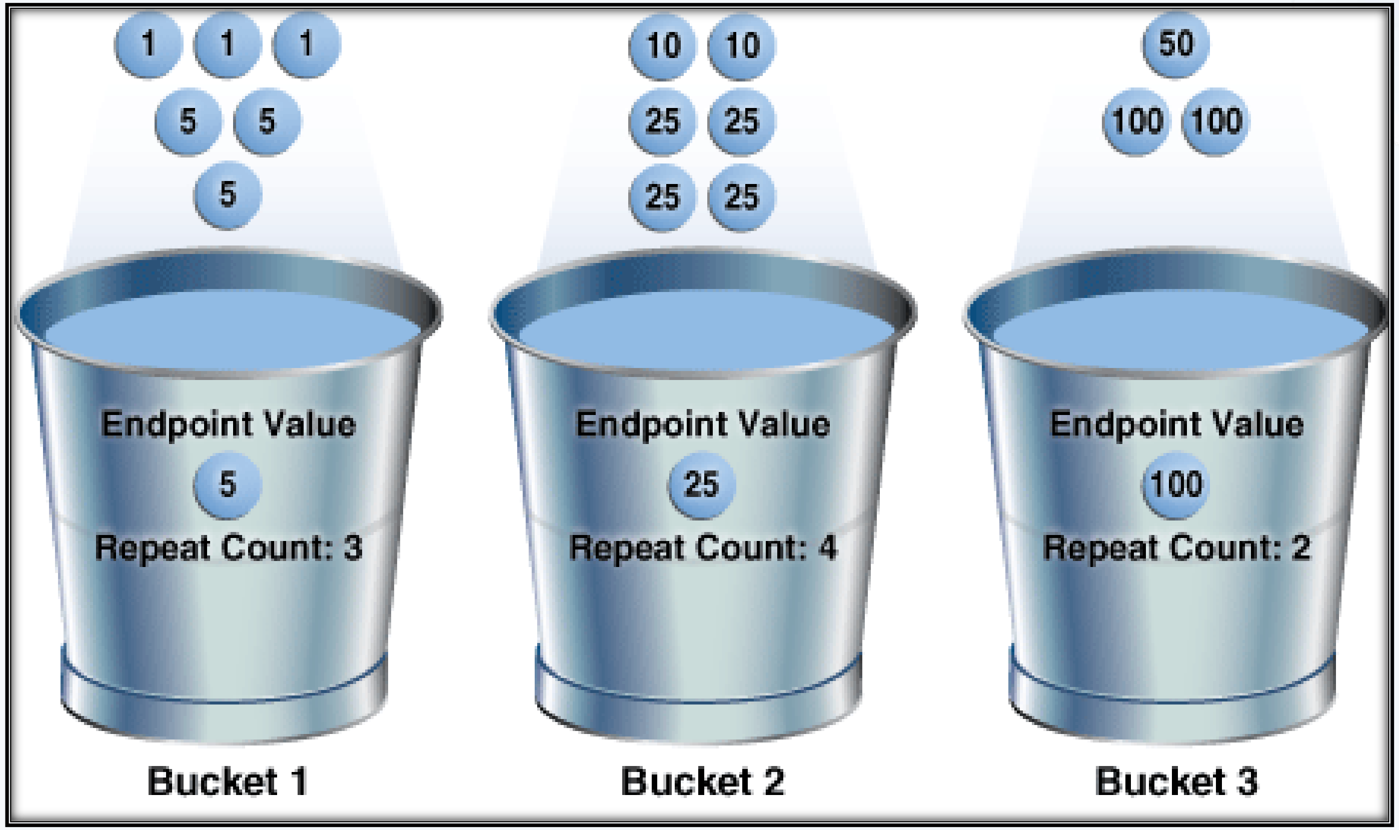
- Time-of-day clock (wall-clock-time)
- Monotonic clocks

Bucket

```
if (value <= 5)
    bucket1(value)
}
else if (value <= 25) {
   bucket2(value)
}
else if (value <= 100) {
   bucket3(value)
}
else {
    infiniti(value)
}
```

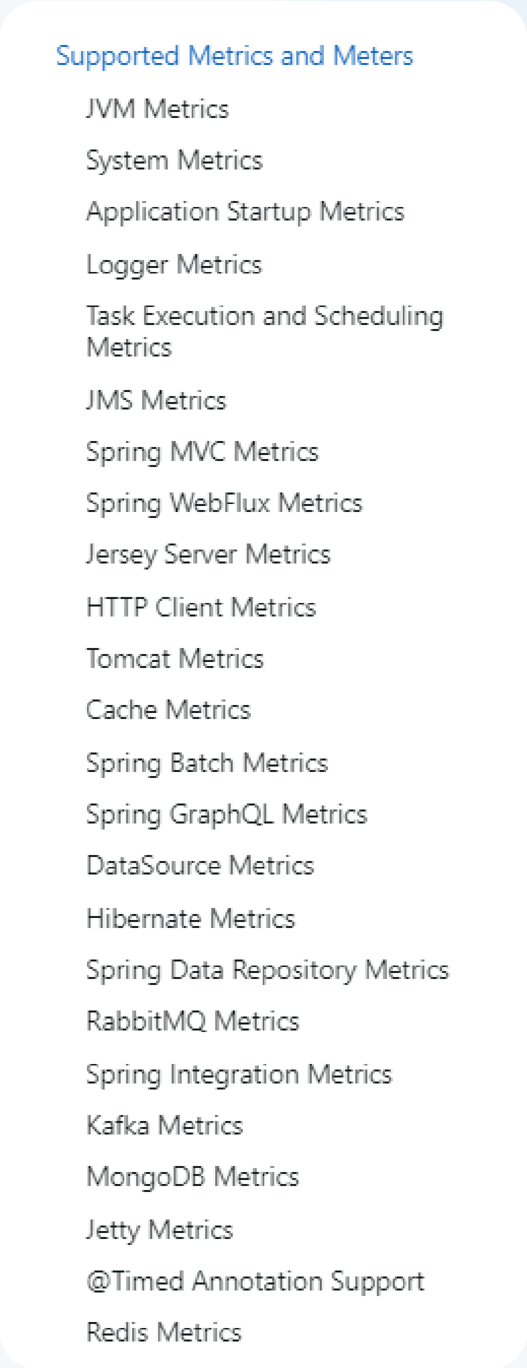
Bucket

```
if (value <= 5)
   bucket1(value)
}
else if (value <= 25) {
  bucket2(value)
}
else if (value <= 100) {
  bucket3(value)
}
else {
   infiniti(value)
}
```



SPRING BOOT ACTUATOR

- Основной инструмент для мониторинга вашего приложения и сбора метрик с него
- Имеет готовый набор production-ready метрик
- Довольно сильная завязка на Micrometer



Supported Metrics and Meters

- JVM Metrics
- System Metrics
- Application Startup Metrics
- Logger Metrics
- Task Execution and Scheduling Metrics
- JMS Metrics
- Spring MVC Metrics
- Spring WebFlux Metrics
- Jersey Server Metrics
- HTTP Client Metrics
- Tomcat Metrics
- Cache Metrics
- Spring Batch Metrics
- Spring GraphQL Metrics
- DataSource Metrics
- Hibernate Metrics
- Spring Data Repository Metrics
- RabbitMQ Metrics
- Spring Integration Metrics
- Kafka Metrics
- MongoDB Metrics
- Jetty Metrics
- @Timed Annotation Support
- Redis Metrics