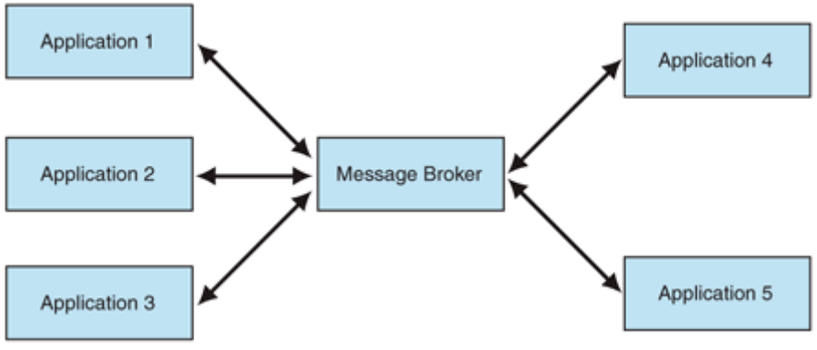
SRE

План занятия

- Очереди сообщений
- Событийно-ориентированная архитектуры
- Метрики
- Мониторинг
- Пару слов о трейсинге и нагрузочных тестах

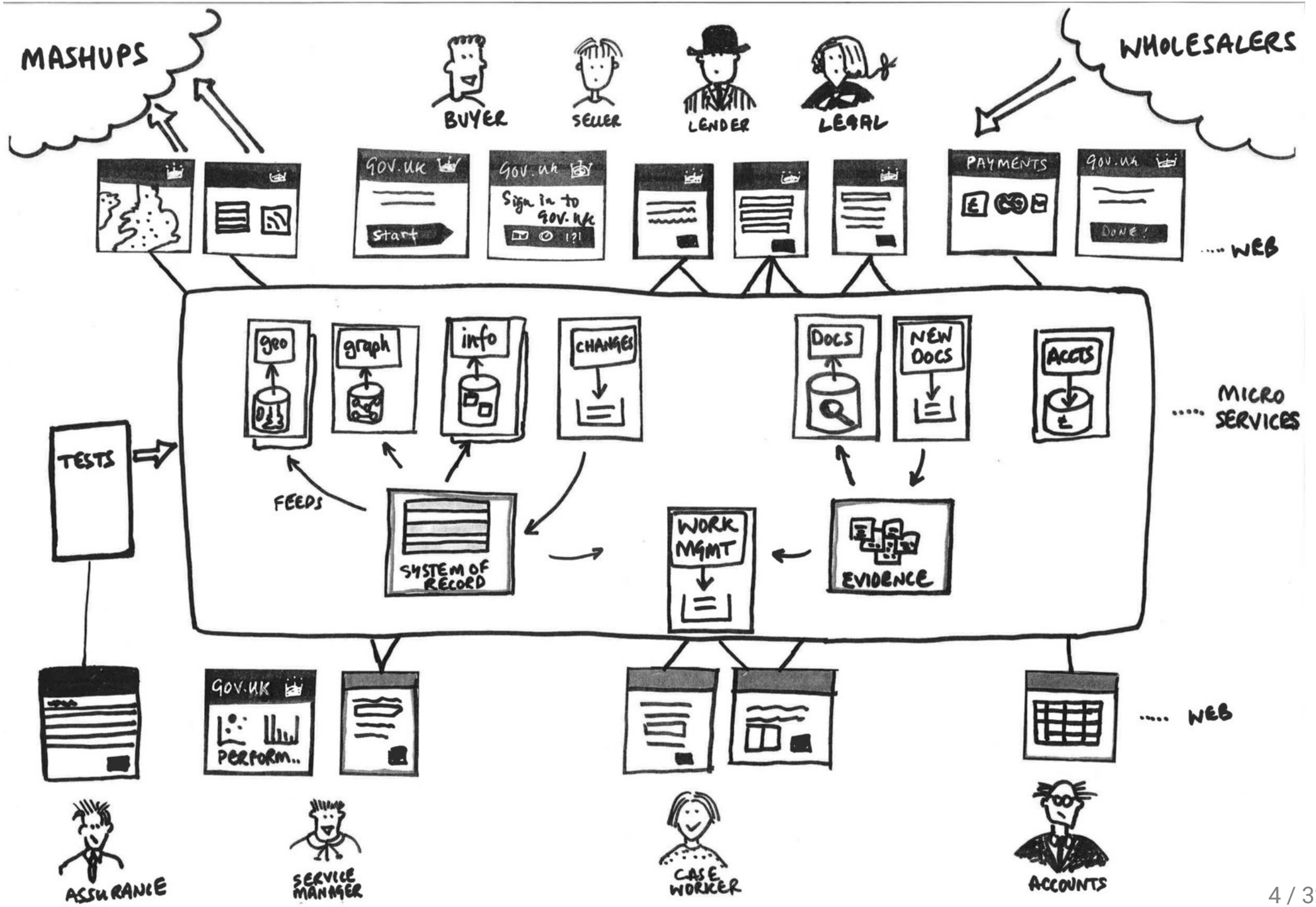
Очередь сообщений / Message Broker



- Yandex Message Queue (https://cloud.yandex.ru/services/message-queue)
- Amazon Web Services (AWS) Simple Queue Service (SQS)
- Apache ActiveMQ
- Apache Kafka
- Redis (pubsub)
- RabbitMQ
- NATS

etc.

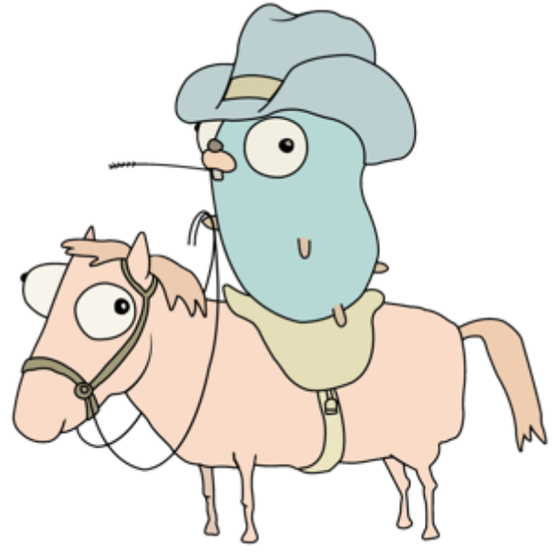
Вернемся к проблемам микросервисов...



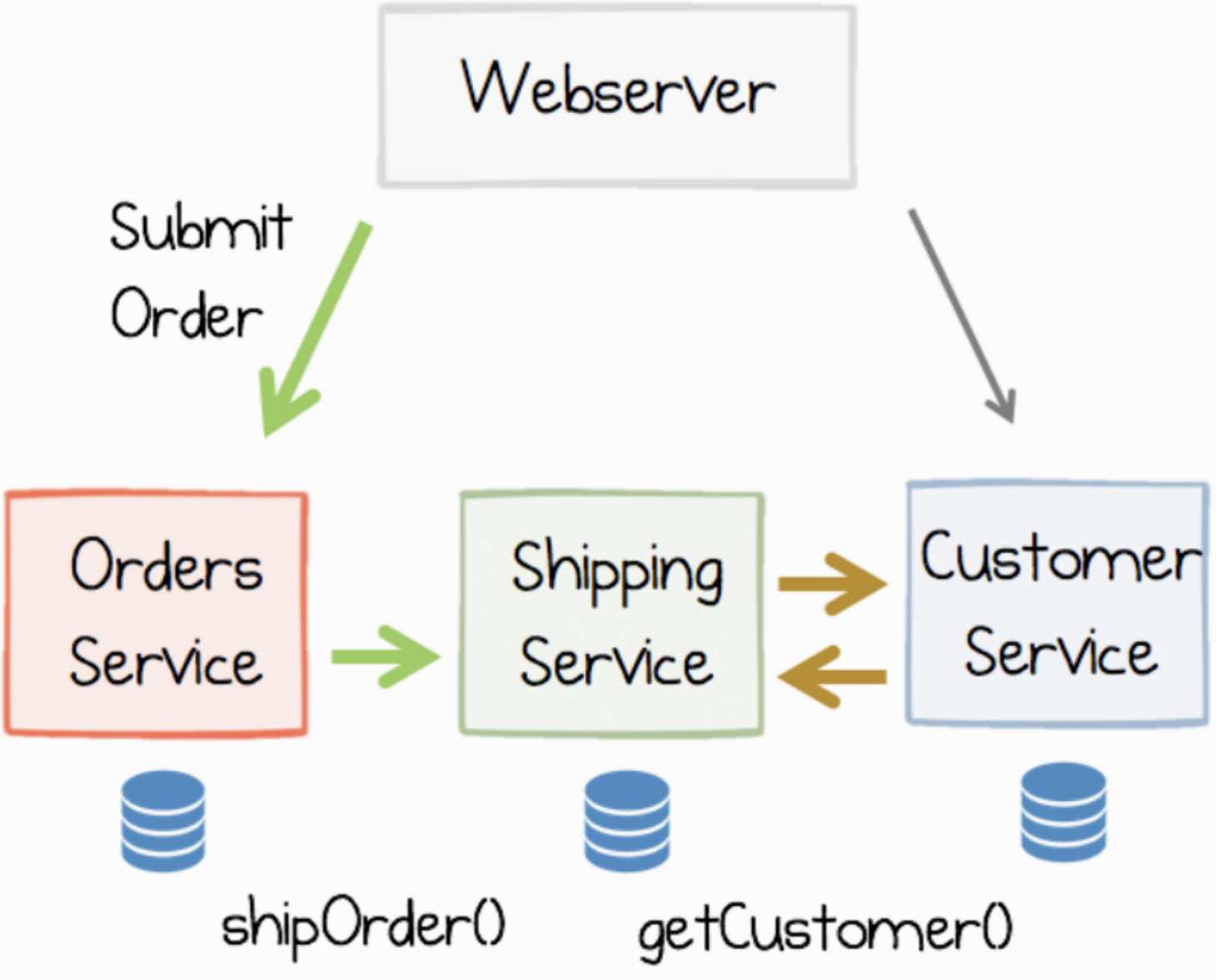
Очереди сообщений

- Слабое связывание
- Масштабируемость*
- Эластичность
- Отказоустойчивость
- Гарантированная доставка*
- Гарантированный порядок доставки*
- Буферизация
- Понимание потоков данных
- Асинхронная связь

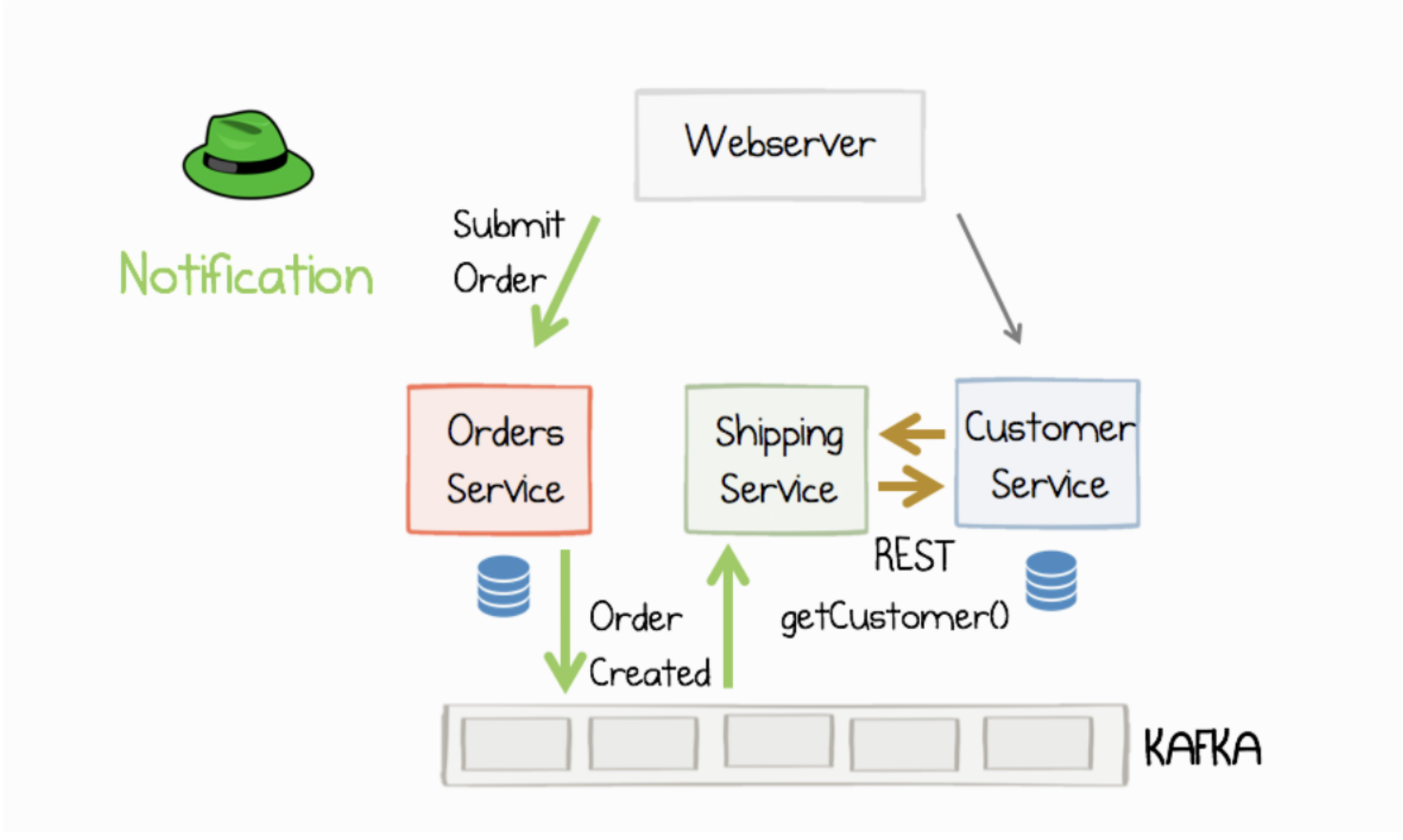
Событийно-ориентированная архитектура



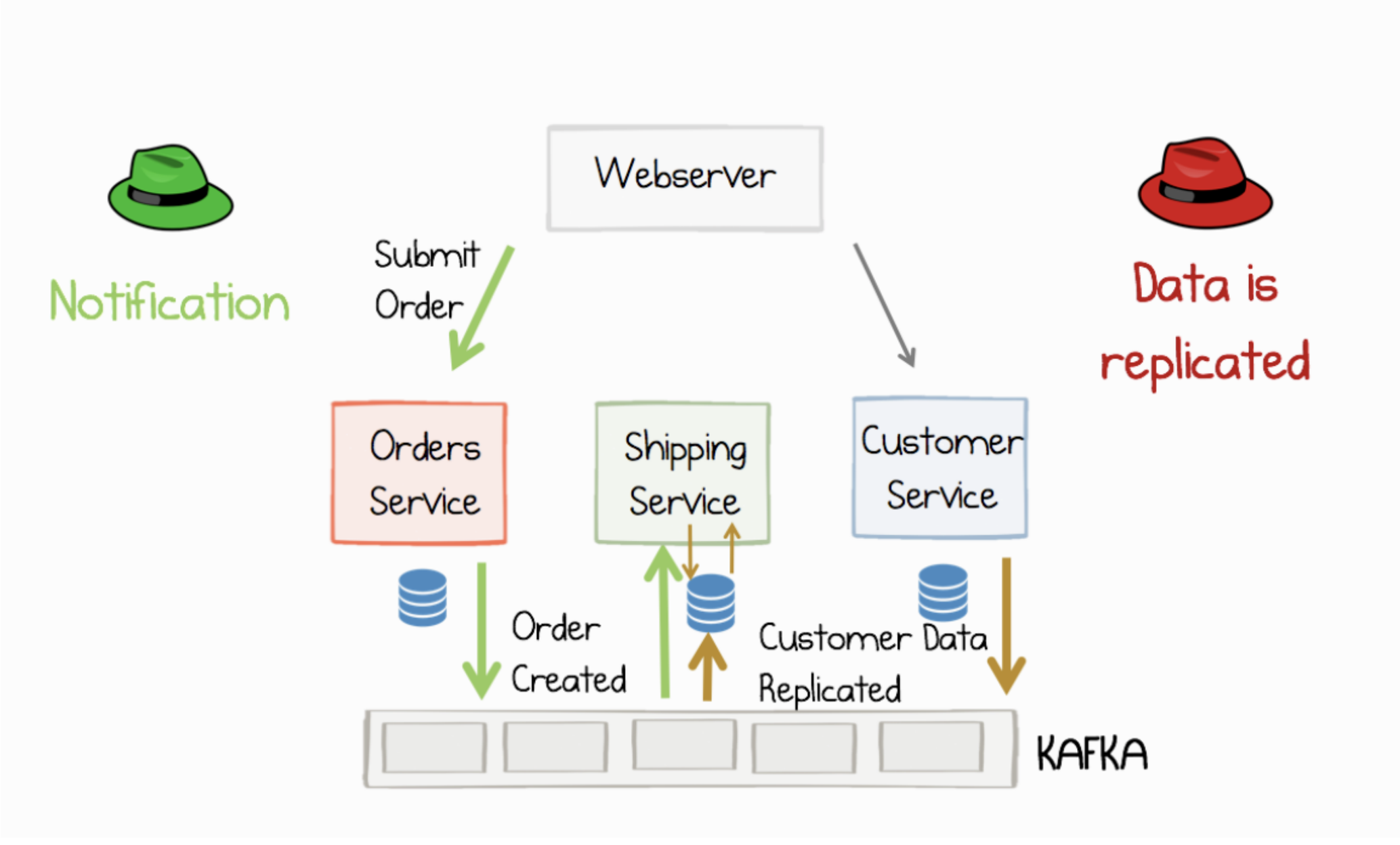
Паттерны: Event Notification



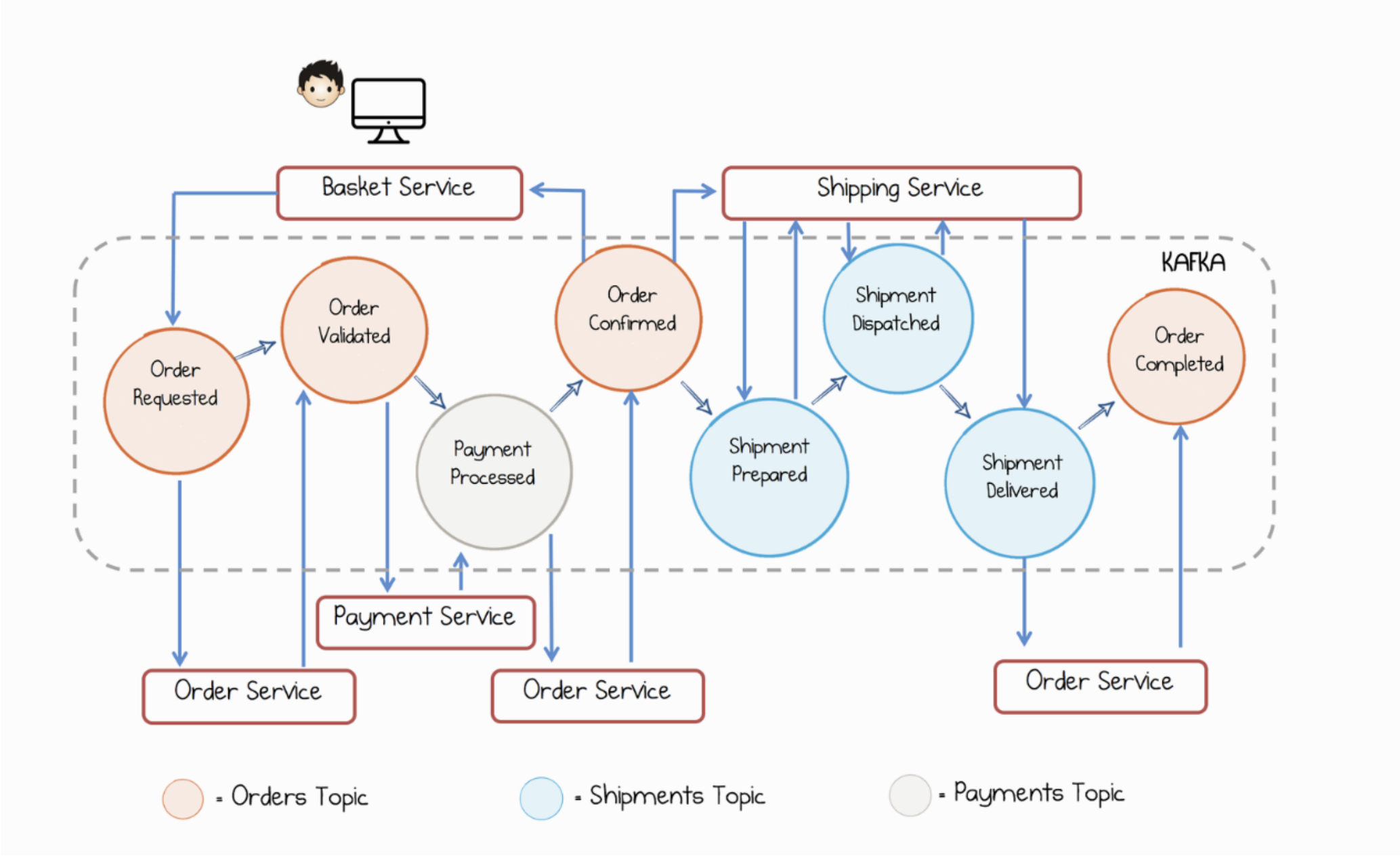
Паттерны: Event Notification



Паттерны: State Transfer



Паттерны: Event Collaboration

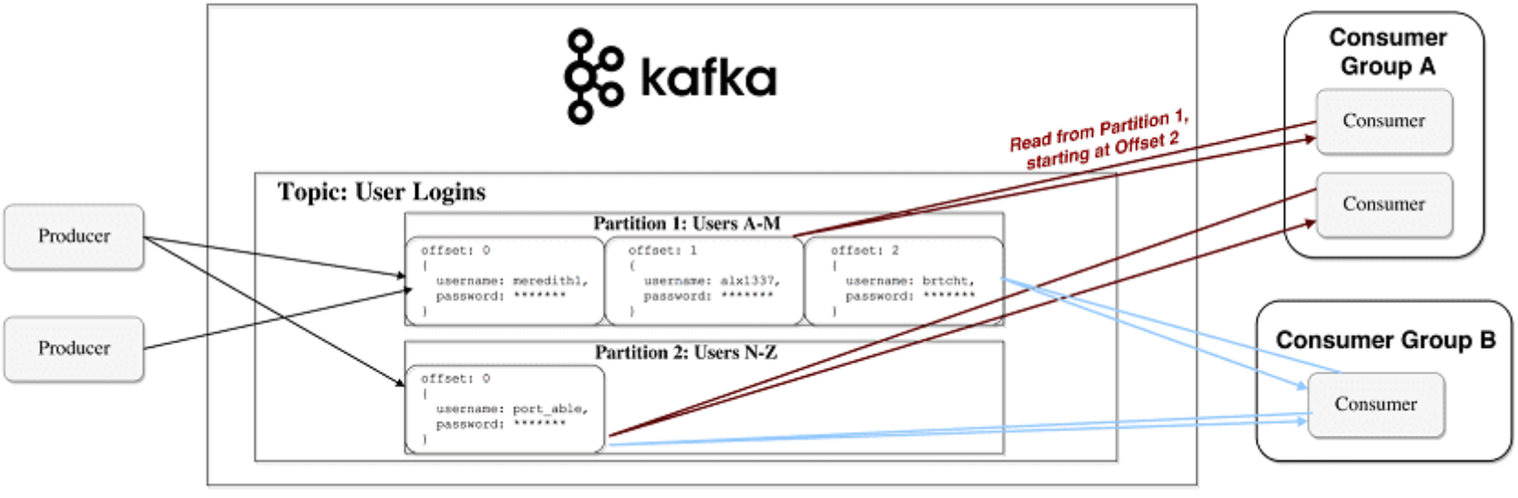


Ссылки

- Разработка транзакционных микросервисов с помощью Агрегатов, Event Sourcing и CQRS
  https://habr.com/ru/company/nix/blog/322214/
- Основы CQRS
  https://habr.com/ru/company/simbirsoft/blog/329970/

Kafka

https://www.gentlydownthe.stream/
Apache Kafka — это распределенная система обмена сообщениями «публикация-подписка» и надежная очередь, которая может обрабатывать большой объем данных и позволяет передавать сообщения из одной конечной точки в другую.
Сообщения Kafka сохраняются на диске и реплицируются в кластере для предотвращения потери данных.
Kafka построен поверх службы синхронизации ZooKeeper.



- **Producer** — процесс, которое производит сообщения.
- **Consumer** — процесс, который читает эти сообщения.
- **Topic** — основная абстракция Apache Kafka. Это место, в котором хранятся все эти записи, каждый топик состоит из Partition.
- **Partition** - следующий уровень абстракции, который основан на разбиении каждого топика на 1, 2 и более частей. Каждое сообщение, находящееся в любом из partition, имеет так называемый offset.
- **Offset** — порядковый номер сообщения в partition. Тут полная аналогия с памятью, чем меньше offset, тем старше сообщение.

Запуск Kafka в Docker

```
version: "3"
services:
zookeeper:
  image: 'bitnami/zookeeper:latest'
  ports:
    - '2181:2181'
  environment:
    - ALLOW_ANONYMOUS_LOGIN=yes
kafka:
  image: 'bitnami/kafka:latest'
  ports:
    - '9092:9092'
  environment:
    - KAFKA_BROKER_ID=1
    - KAFKA_LISTENERS=PLAINTEXT://:9092
    - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://127.0.0.1:9092
    - KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
    - ALLOW_PLAINTEXT_LISTENER=yes
  depends_on:
    - zookeeper
```

Посмотрим на код

Kafka: что почитать

- Designing Event Driven Systems
  http://www.benstopford.com/2018/04/27/book-designing-event-driven-systems/
- Kafka: The Definitive Guide
  https://www.confluent.io/wp-content/uploads/confluent-kafka-definitive-guide-complete.pdf

RabbitMQ vs Kafka

- https://jack-vanlightly.com/blog/2017/12/4/rabbitmq-vs-kafka-part-1-messaging-topologies
- https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka

И еще почитать

- RabbitMQ против Kafka: два разных подхода к обмену сообщениями
  https://habr.com/ru/company/itsumma/blog/416629/
- Understanding When to use RabbitMQ or Apache Kafka
  https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka
- Apache Kafka: обзор
  http://habr.com/ru/company/piter/blog/352978/
- Kafka и микросервисы: обзор
  https://habr.com/ru/company/avito/blog/465315/
- Apache Kafka и миллионы сообщений в секунду
  https://habr.com/ru/company/tinkoff/blog/342892/
- Apache Kafka и RabbitMQ: семантика и гарантия доставки сообщений
  https://habr.com/ru/company/itsumma/blog/437446/

Observability

Observability (наблюдаемость) - мера того, насколько по выходным данным можно восстановить информацию о состоянии системы.

Примеры:
- логирование (zap, logrus -> fluentd -> kibana)
- мониторинг (zabbix, prometheus)
- алертинг (chatops, pagerduty, opsgenie)
- трейсинг (jaeger, zipkin, opentracing, datadog apm)
- профилирование (pprof)
- сбор ошибок и аварий (sentry)

Operability

Operability (работоспособность) - мера того, насколько приложение умеет сообщать о своем состоянии здоровья, а инфраструктура управлять этим состоянием.

Примеры:
- простейшие хелсчеки
- liveness и readiness в Kubernetes

Виды мониторинга

- Количественный / Событийный (y/n)
- Whitebox / Blackbox
- Push / Pull

Push vs Pull

Push - агент, работающий на окружении (например, сайдкар), подключается к серверу мониторинга и отправляет данные.

Особенности:
- мониторинг специфических/одноразовых задач
- может работать за NAT
- не нужно открывать никакие URL'ы/порты на стороне приложения
- из приложения нужно конфигурировать подключение

Примеры: `graphite`, `statsd`

Pull - сервис мониторинга сам опрашивает инфраструктуры/сервисы и агрегирует статистику.

Особенности:
- не нужно подключаться к агенту на стороне приложения
- нужно открывать URL или порт, который будет доступен сервису мониторинга
- более отказоустойчивый
- не требует авторизации /верификации источника

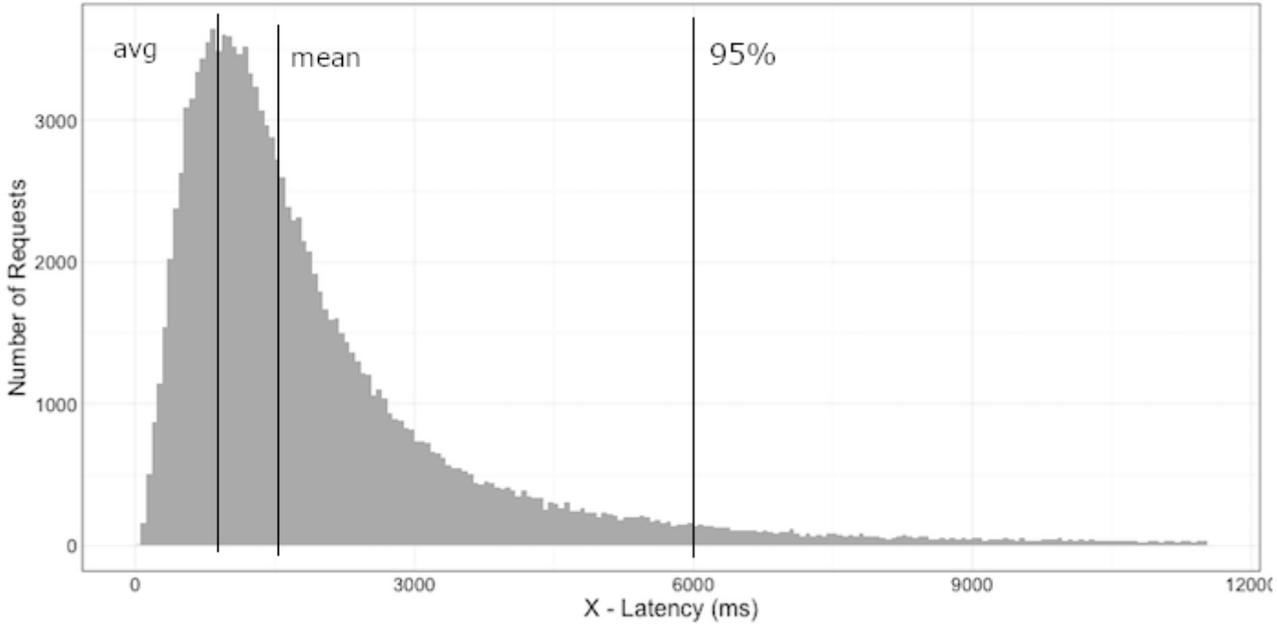
Примеры: `datadog-agent`, `prometheus`

Мониторинг Web/API серверов

- RPS (request per second)
- Response time
- Задержка между компонентами приложения (latency)
- Код ответа (HTTP status 200/500/5xx/4xx)
- Разделение по методам API

Для детального анализа: трейсинг, например,
https://opentracing.io/

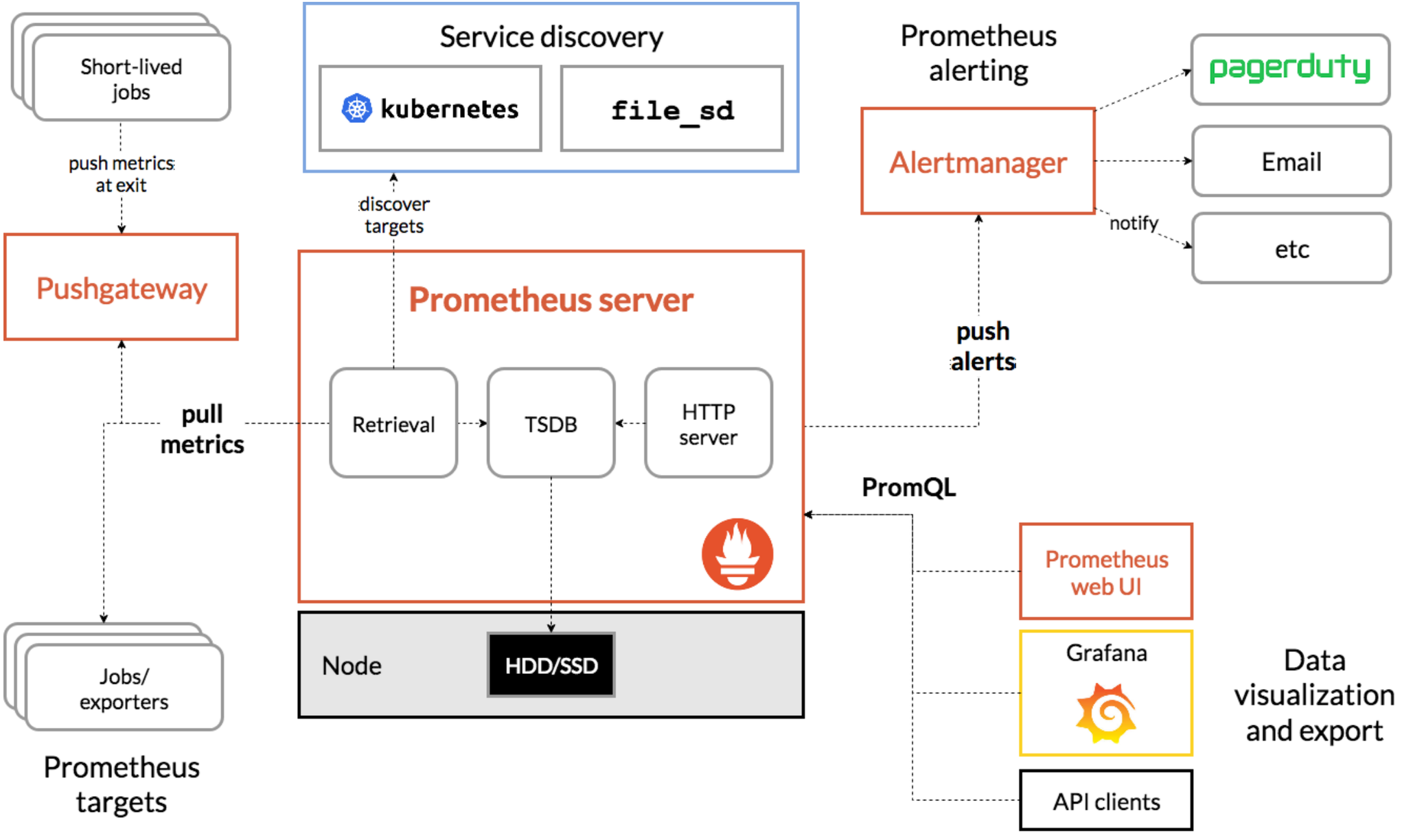
Распределение значений



Среднее значение (`avg` , `mean`) или даже медиана (`median`) не отображают всей картины!

Полезно измерять *процентили* (percentile): время в которое укладываются 95% или, например, 99% запросов.

Prometheus


Short-lived jobs
push metrics at exit
Pushgateway
Service discovery
kubernetes
file_sd
discover targets
Prometheus server
Retrieval
TSDB
HTTP server
pull metrics
Node
HDD/SSD
Jobs/ exporters
Prometheus targets
Prometheus alerting
Alertmanager
pagerduty
Email
etc
notify
push alerts
PromQL
Prometheus web UI
Grafana
API clients
Data visualization and export

Prometheus - установка и запуск сервера

```
docker run \
    -p 9090:9090 \
    -v /tmp/prometheus.yml:/etc/prometheus/prometheus.yml \
    prom/prometheus
```

Настройка `/tmp/prometheus.yml`

```
global:
  scrape_interval:  15s  # как часто опрашивать exporter-ы

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'app'
    static_configs:
      - targets: ['localhost:9100', 'localhost:9102', 'localhost:9103', 'localhost:9187']
```

https://prometheus.io/docs/prometheus/latest/installation/

Prometheus - запуск

С настройками по умолчанию Prometheus будет доступен на порту 9090: http://127.0.0.1:9090/

Prometheus - мониторинг

- мониторинг сервера:
  https://github.com/prometheus/collectd_exporter (collectd + collectd-exporter)
  https://github.com/prometheus/node_exporter

- мониторинг базы: postgres-exporter
  https://github.com/wrouesnel/postgres_exporter

- визуализация
  https://grafana.com/docs/grafana/latest/installation/docker/

Prometheus - протокол

Простой способ исследовать: `wget -O - http://localhost:9103/metrics`

```
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 1.036096e+06

collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="blocked"} 0
collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="paging"} 0
collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="running"} 1
collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="sleeping"} 57
collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="stopped"} 0
collectd_processes_ps_state{instance="mialinx-test-ub.ru-central1.internal",processes="zombies"} 0

go_gc_duration_seconds{quantile="0"} 4.0147e-05
go_gc_duration_seconds{quantile="0.25"} 6.950600000000001e-05
go_gc_duration_seconds{quantile="0.5"} 0.000108126
go_gc_duration_seconds{quantile="0.75"} 0.001107202
go_gc_duration_seconds{quantile="1"} 0.039212351
go_gc_duration_seconds_sum 0.49406203400000004
go_gc_duration_seconds_count 282
```

Prometheus - типы метрик

- `Counter` - монотонно возрастающее число, например, число запросов
- `Gauge` - текущее значение, например, потребление памяти
- `Histogram` - распределение значений по бакетам (сколько раз значение попало в интервал)
- `Summary` - похоже на `histogram`, но по квантилям
- Векторные типы для подсчета данных по меткам

Документация: https://prometheus.io/docs/concepts/metric_types/

Отличная документация в godoc: https://godoc.org/github.com/prometheus/client_golang/prometheus

Prometheus - мониторинг Go HTTP сервисов

```
import (
  "log"
  "net/http"
  "github.com/prometheus/client_golang/prometheus/promhttp"
  metrics "github.com/slok/go-http-metrics/metrics/prometheus"
  "github.com/slok/go-http-metrics/middleware"
)

func myHandler(w http.ResponseWriter, r *http.Request) {
  w.WriteHeader(http.StatusOK)
  w.Write([]byte("hello world!"))
}

func main() {
  // middleware для мониторинг
  mdlw := middleware.New(middleware.Config{
    Recorder: metrics.NewRecorder(metrics.Config{}),
  })
  h := mdlw.Handler("", http.HandlerFunc(myHandler))
  // HTTP exporter для prometheus
  go http.ListenAndServe(":9102", promhttp.Handler())
  // Ваш основной HTTP сервис
  if err := http.ListenAndServe(":8080", h); err != nil {
    log.Panicf("error while serving: %s", err)
  }
}
```

Prometheus - собственные метрики

```
import "github.com/prometheus/client_golang/prometheus"

var regCounter = prometheus.NewCounter(prometheus.CounterOpts{
  Name: "business_registration",
  Help: "Client registration event",
})

func init() {
  prometheus.MustRegister(regCounter)
}

func myHandler(w http.ResponseWriter, r *http.Request) {
  w.WriteHeader(http.StatusOK)
  w.Write([]byte("Hello, world!"))
  regCounter.Inc()
}
```