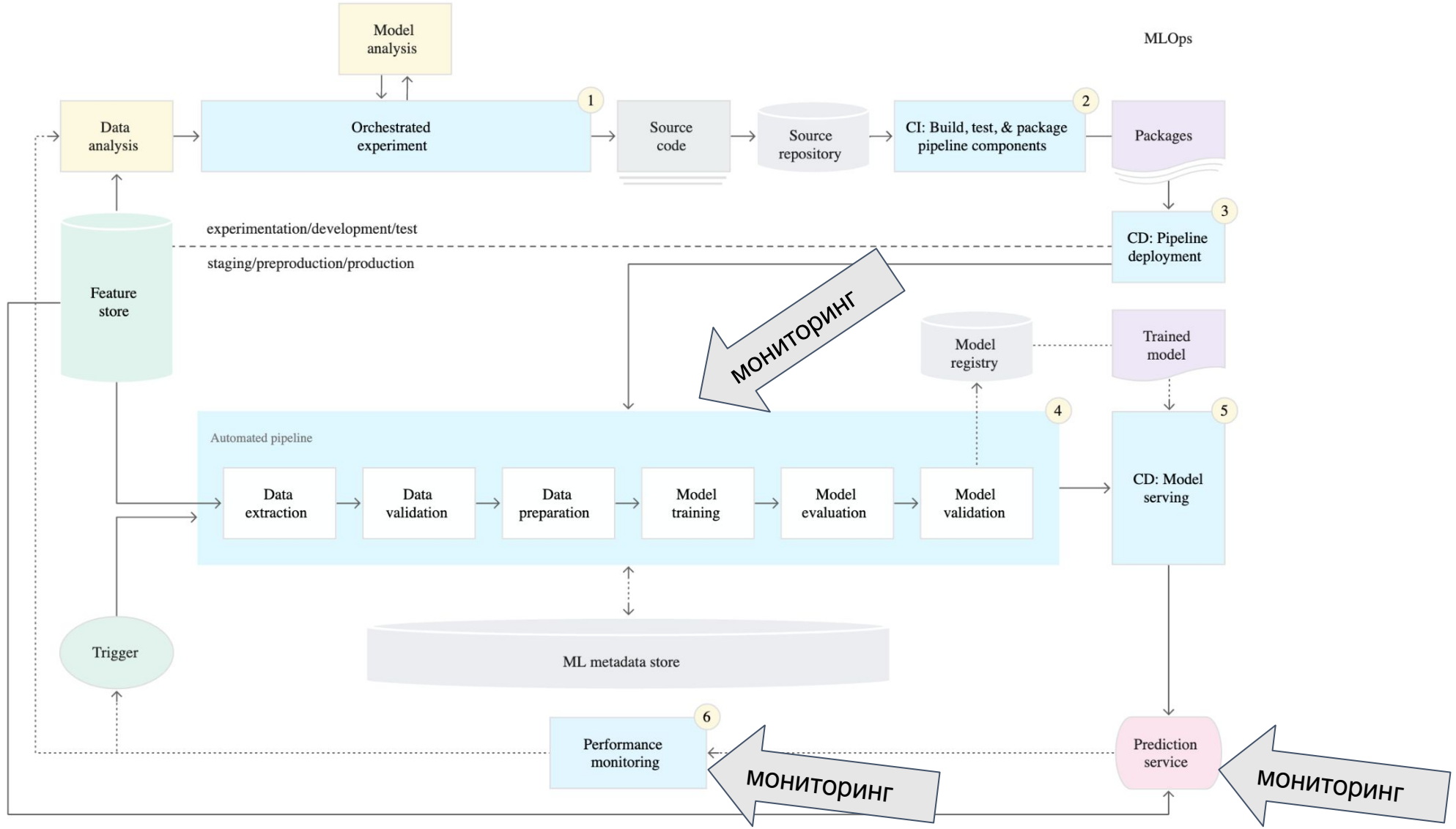
академия больших данных

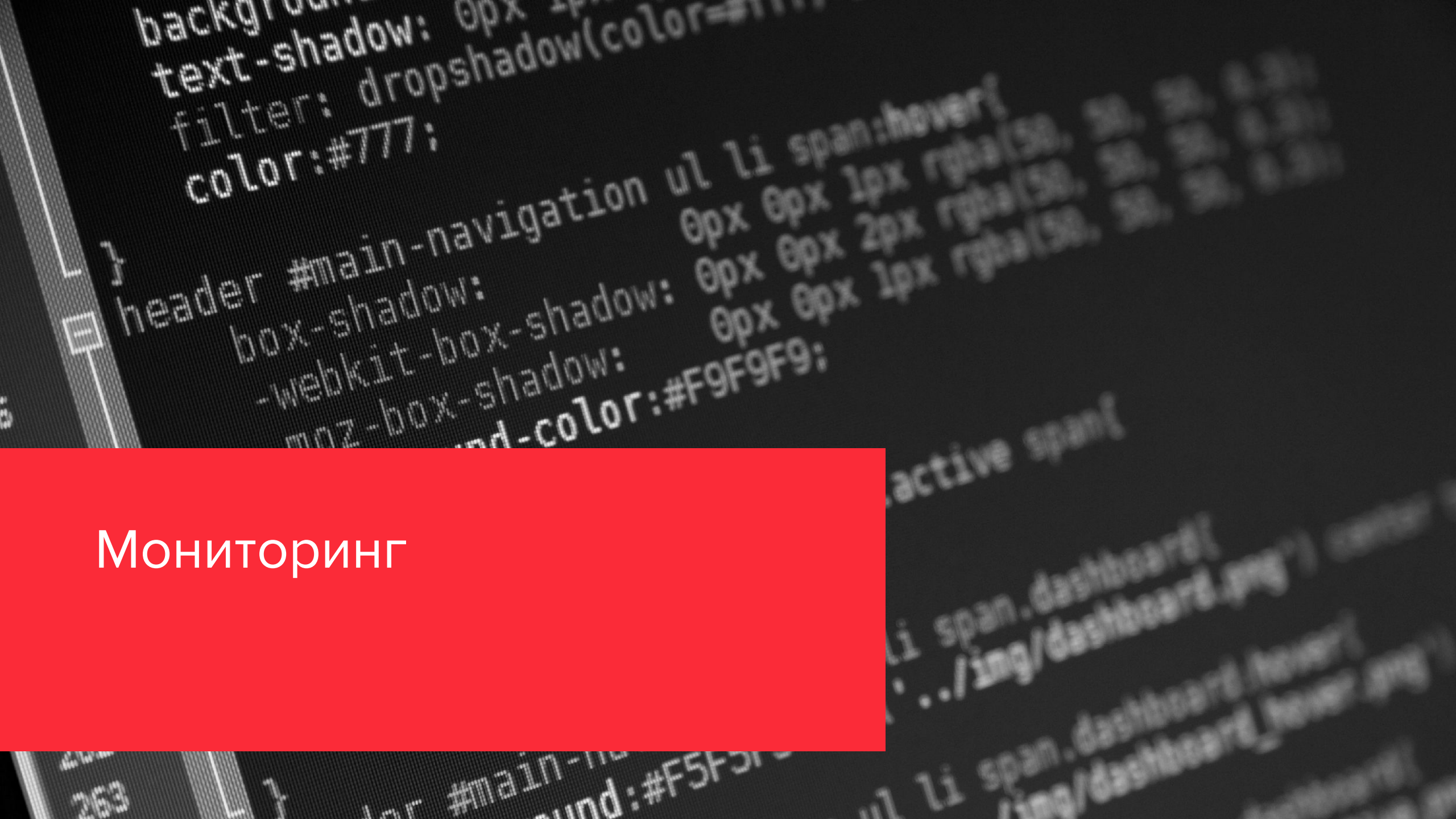
mail.ru group

Мониторинг, постепенные выкатки

Михаил Марюфич, MLE

О чем мы сегодня?




text-shadow:
filter: dropshadow(color=
color:#777;
header #main-navigation ul li span:hover{
box-shadow:
-webkit-box-shadow:
moz-box-shadow:
0px 0px 1px
0px 0px 2px
0px 0px 1px
-color:#F9F9F9;
active span{

Мониторинг

Ожидания от мониторинга

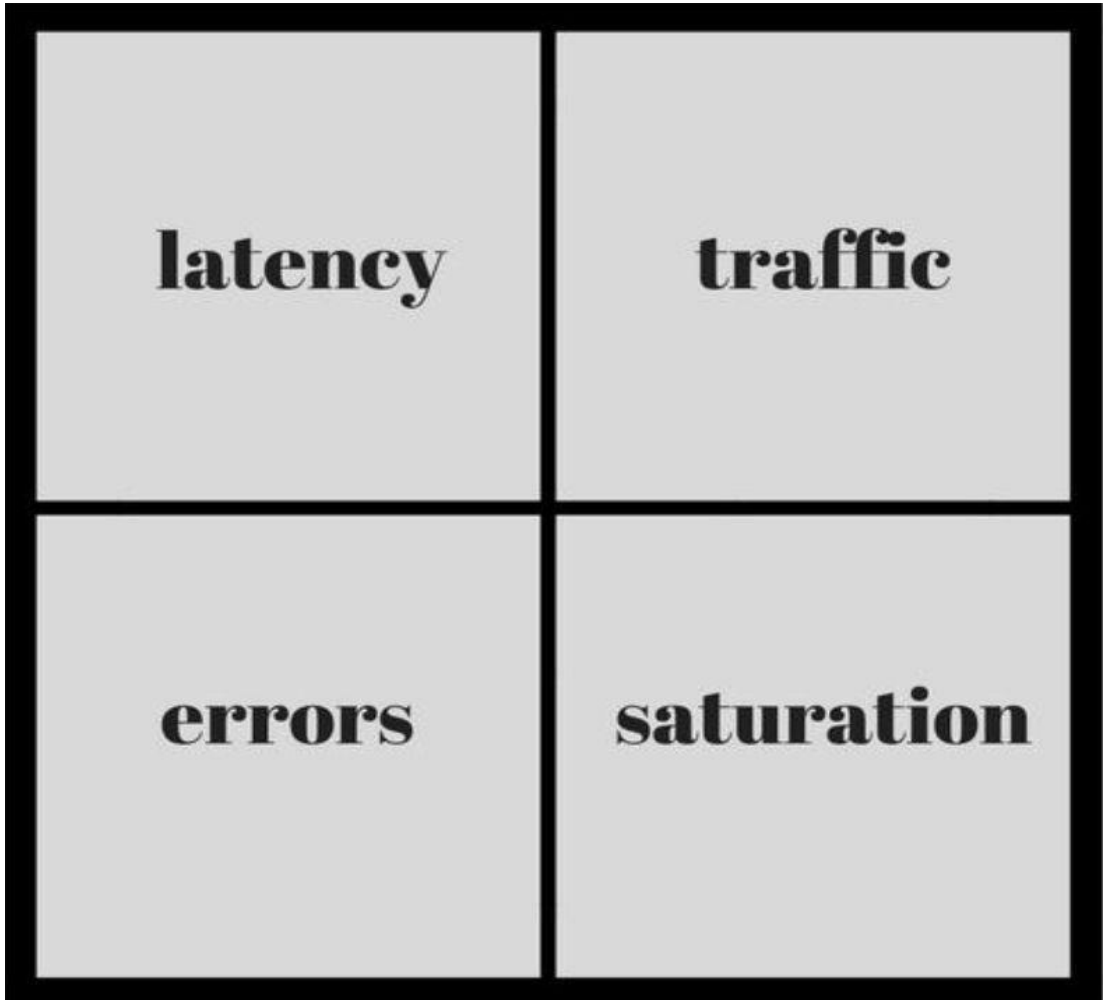
- Алертинг о сбоях в системе
- Предупреждения о возможных сбоях и проблемах
- Отражение состояния системы/сервера/сервиса/компонента сервиса
- Сбор статистики и визуализаций
- Отчеты
- Дашборды

The Four Golden Signals

Подход был озвучен в SRE handbook

Latency — Время необходимое на обслуживание запроса
Traffic — Количество запросов отправляемых на вашу систему
Errors — Количество ошибок
Saturation — Насколько полон ваш сервис

The Four Golden Signals



The USE

Resource - все компоненты физического\виртуального сервера (CPU, Disk, RAM, etc.)

Utilization - время которое затрачивает ресурс на выполнение задач

Saturation - показатель указывающий на количество тасков которые не могут быть выполнены, при этом попадая в очередь

Errors - количество ошибок

Ресурсы

- CPU
- DISK
- RAM
- GPU
- NETWORK

The RED

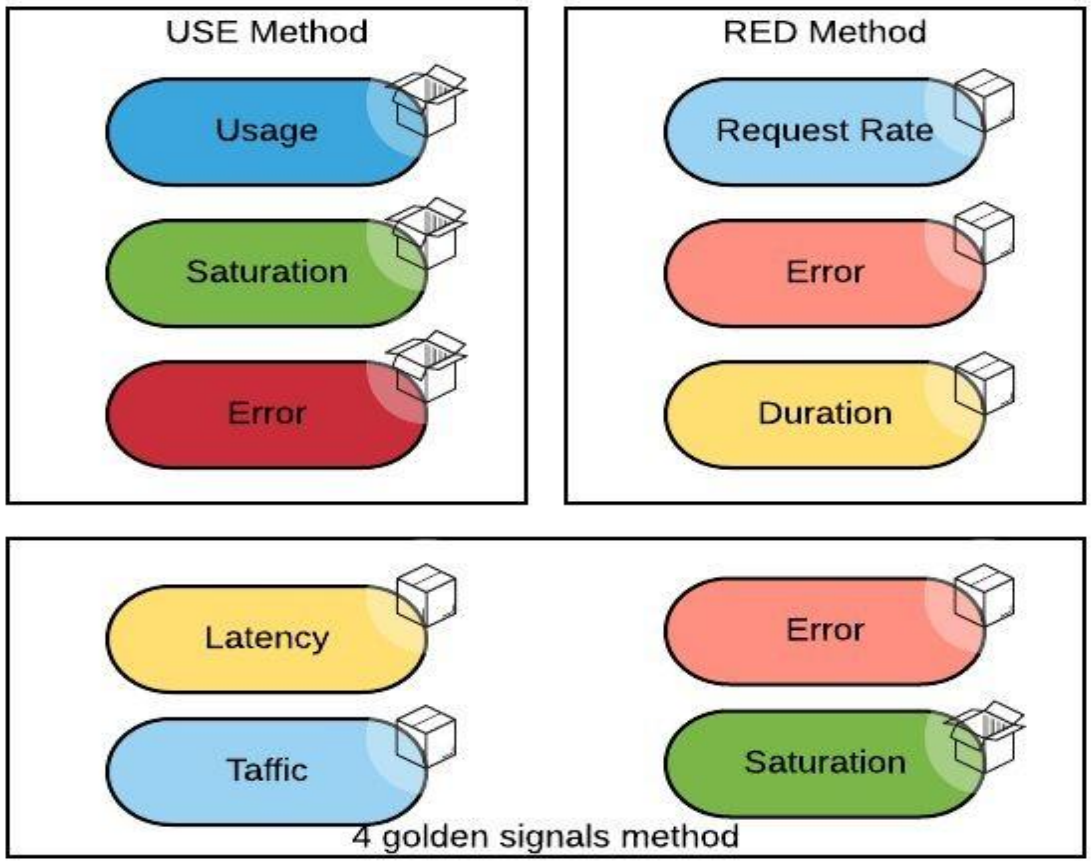
Rate - количество запросов в секунду

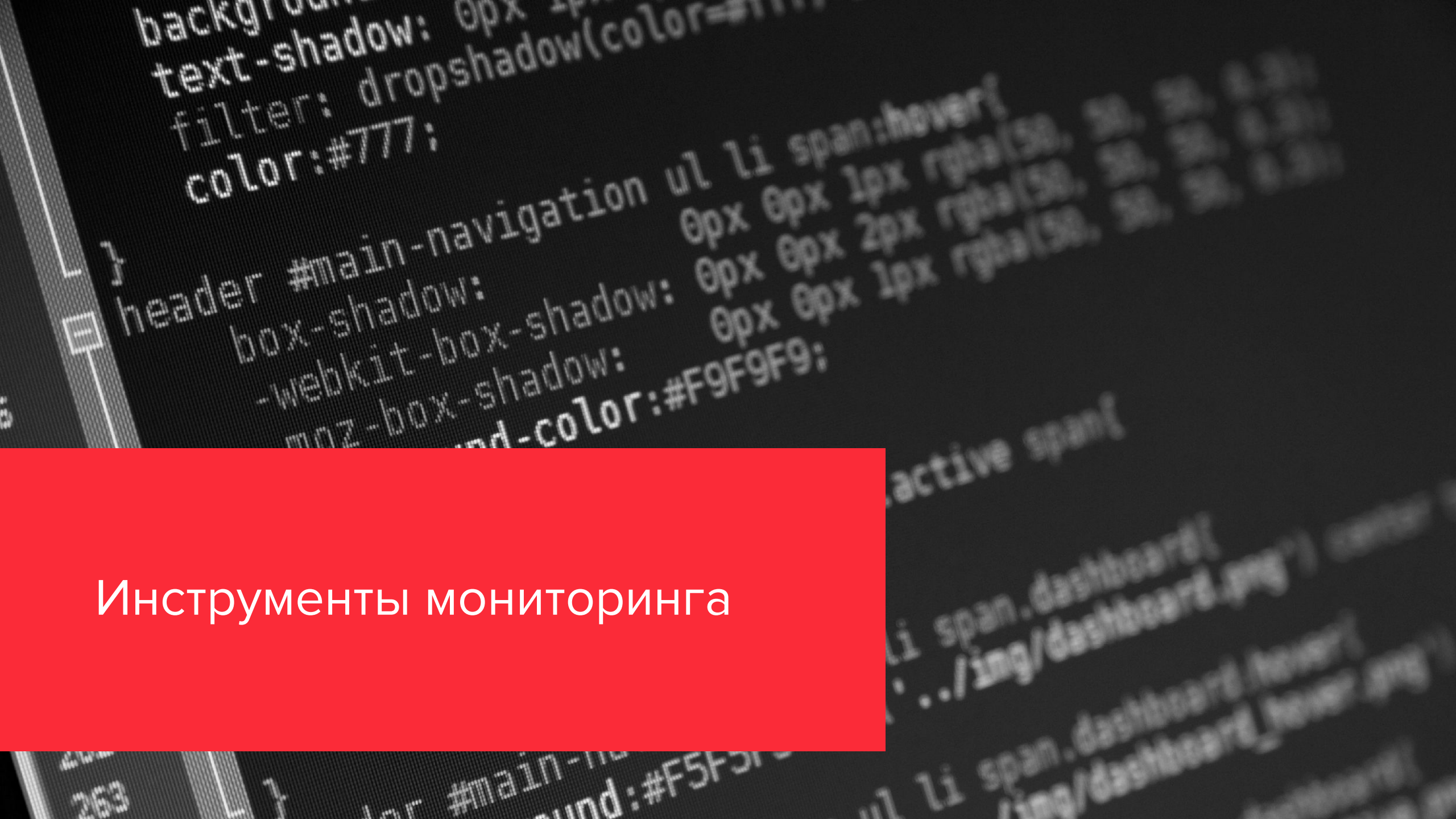
Errors - количество запросов завершившихся с ошибкой

Duration - количество времени которое занимает каждый запрос

```
The USE method is for resources and the RED method is for my services
```

Подходы к мониторингу





Инструменты мониторинга

Blackbox

Мониторинг, оценивающий внешнее состояние сервиса/системы:

- ping
- HTTP request
- Открытый порт
- Наличие процессов

Whitebox

Мониторинг, базирующийся на метриках, которое дает само приложение/сервис:

- через логи
- интерфейсы
- API

Метрики

Технические метрики:
Все, что касается ресурсов (средняя загрузка процессора, занятость дисков)

Сервисные метрики:
Все, что касается обработки запросов (процент успешных запросов, latency, etc) —

Бизнес метрики

Исторические данные (time series)

Как их собирать, где хранить, сколько хранить, как с ними работать?

Зачем нужно исторические данные?

1) Для анализа проблем
2) Для обучения моделей, которые будут искать аномалии на них =)

Prometheus

Pushgateway

Prometheus server

Alertmanager

Web-UI

PUSH

Ваше приложение отправляет куда-то свои метрики

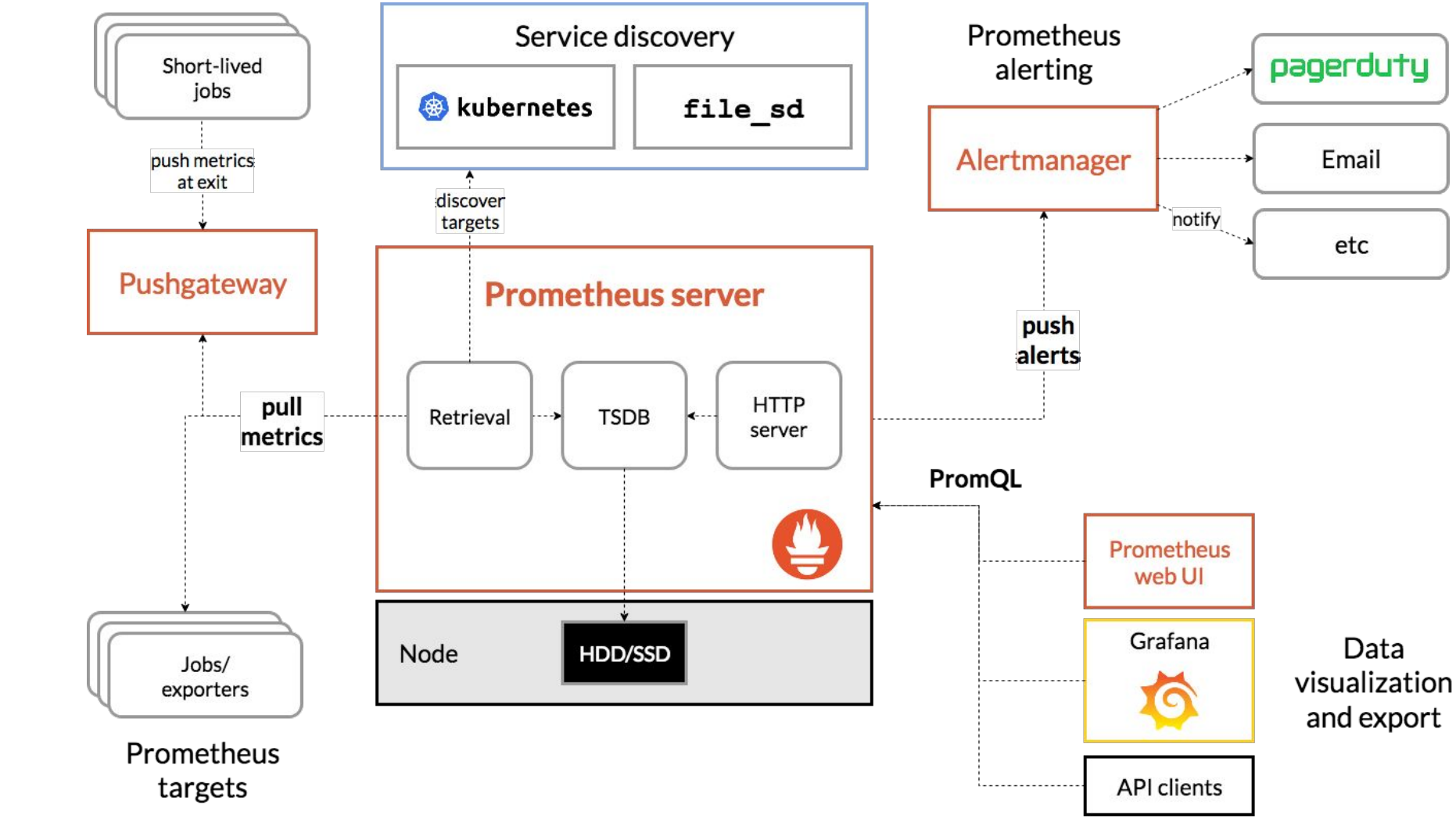
Pull vs Push

PULL

Ваше приложение выставляет endpoint с метриками

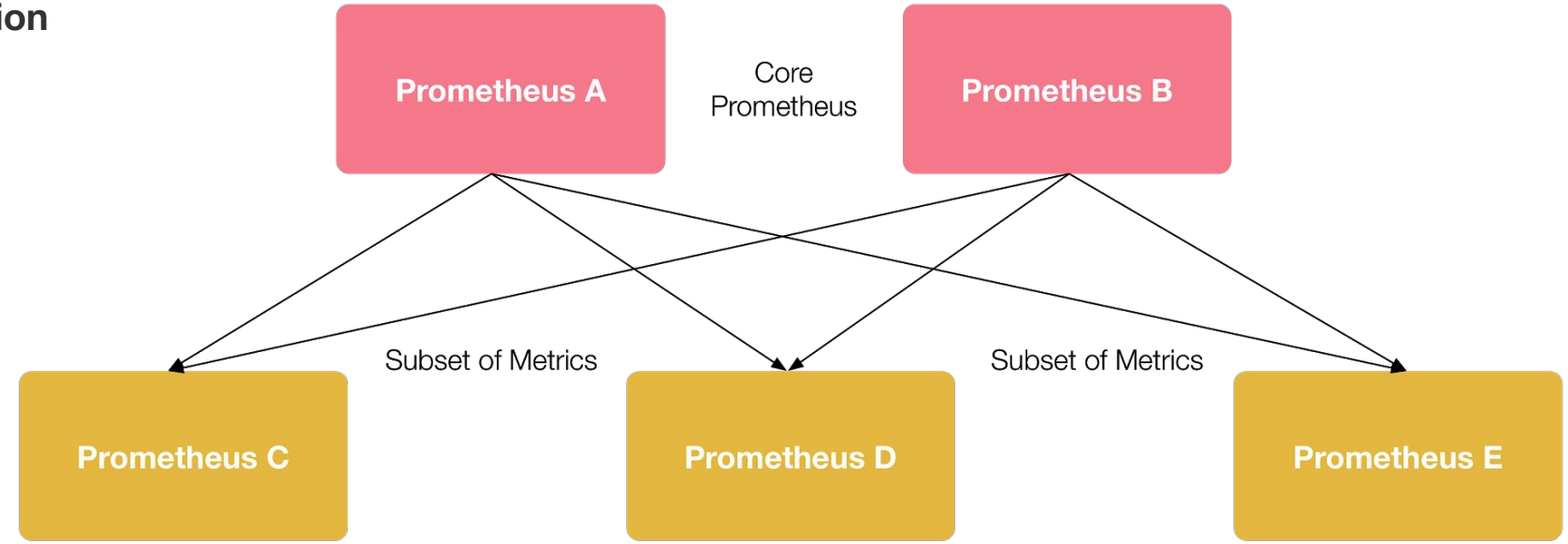
Pull vs Push

Architecture



Federation

Hierarchical federation
Cross-service federation



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

Типы метрик — Counter

Counter

A *counter* is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart. For example, you can use a counter to represent the number of requests served, tasks completed, or errors.

Do not use a counter to expose a value that can decrease. For example, do not use a counter for the number of currently running processes; instead use a gauge.

Примеры: количество успешных запросов, количество ошибок

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

Типы метрик — Gauge

Gauge

A *gauge* is a metric that represents a single numerical value that can arbitrarily go up and down.

Gauges are typically used for measured values like temperatures or current memory usage, but also "counts" that can go up and down, like the number of concurrent requests.

Примеры: количество машин, количество задач в работе

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

Типы метрик — Histogram

Histogram

A *histogram* samples observations (usually things like request durations or response sizes) and counts them in configurable buckets. It also provides a sum of all observed values.

A histogram with a base metric name of `<basename>` exposes multiple time series during a scrape:

- cumulative counters for the observation buckets, exposed as `<basename>_bucket{le="<upper inclusive bound>"}`
- the **total sum** of all observed values, exposed as `<basename>_sum`
- the **count** of events that have been observed, exposed as `<basename>_count` (identical to `<basename>_bucket{le="+Inf"}` above)

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

Типы метрик — Summary

Histogram

A *histogram* samples observations (usually things like request durations or response sizes) and counts them in configurable buckets. It also provides a sum of all observed values.

A histogram with a base metric name of `<basename>` exposes multiple time series during a scrape:

- cumulative counters for the observation buckets, exposed as `<basename>_bucket{le="<upper inclusive bound>"}`
- the **total sum** of all observed values, exposed as `<basename>_sum`
- the **count** of events that have been observed, exposed as `<basename>_count` (identical to `<basename>_bucket{le="+Inf"}` above)

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

Python client

Counter

Counters go up, and reset when the process restarts.

```
from prometheus_client import Counter
c = Counter('my_failures', 'Description of counter')
c.inc()     # Increment by 1
c.inc(1.6)  # Increment by given value
```

Gauge

Gauges can go up and down.

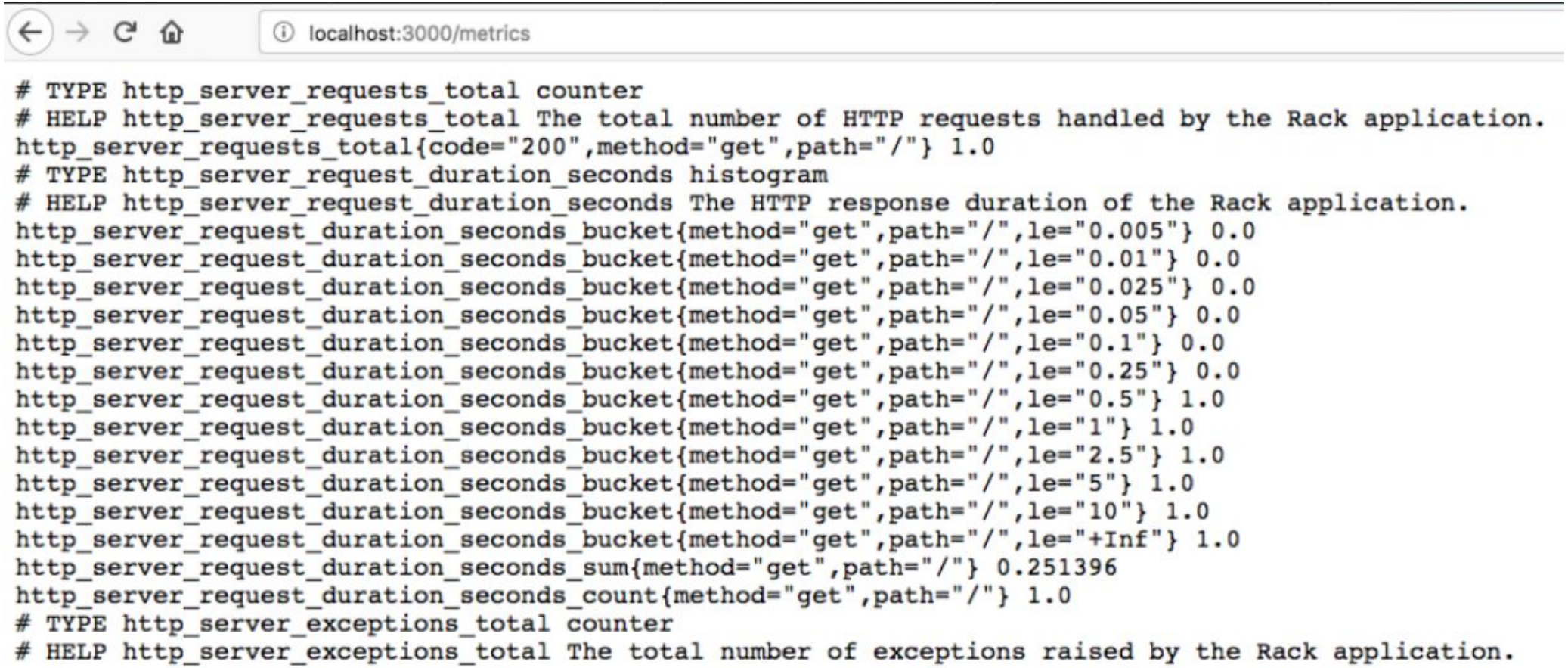
```
from prometheus_client import Gauge
g = Gauge('my_inprogress_requests', 'Description of gauge')
g.inc()      # Increment by 1
g.dec(10)    # Decrement by given value
g.set(4.2)   # Set to a given value
```

https://www.cloudbees.com/blog/monitoring-your-asynchronous-python-web-applications-using-prometheus

https://github.com/prometheus/client_python

Metric endpoint

Приложение выставляет endpoint для метрик



PROM QL

Selecting series

Select latest sample for series with a given metric name:

```
node_cpu_seconds_total
```

Open in PromLens

Select 5-minute range of samples for series with a given metric name:

```
node_cpu_seconds_total[5m]
```

Open in PromLens

Only series with given label values:

```
node_cpu_seconds_total{cpu="0",mode="idle"}
```

Open in PromLens

Complex label matchers:

```
node_cpu_seconds_total{cpu!="0",mode=~"user|system"}
```

Open in PromLens

- `=`: Equality
- `!=`: Non-equality
- `=~`: Regex match
- `!~`: Negative regex match

Aggregating over multiple series

Sum over all series:

```
sum(node_filesystem_size_bytes)
```

Open in PromLens

Preserve the `instance` and `job` label dimensions:

```
sum by(job, instance) (node_filesystem_size_bytes)
```

Open in PromLens

Aggregate away the `instance` and `job` label dimensions:

```
sum without(instance, job) (node_filesystem_size_bytes)
```

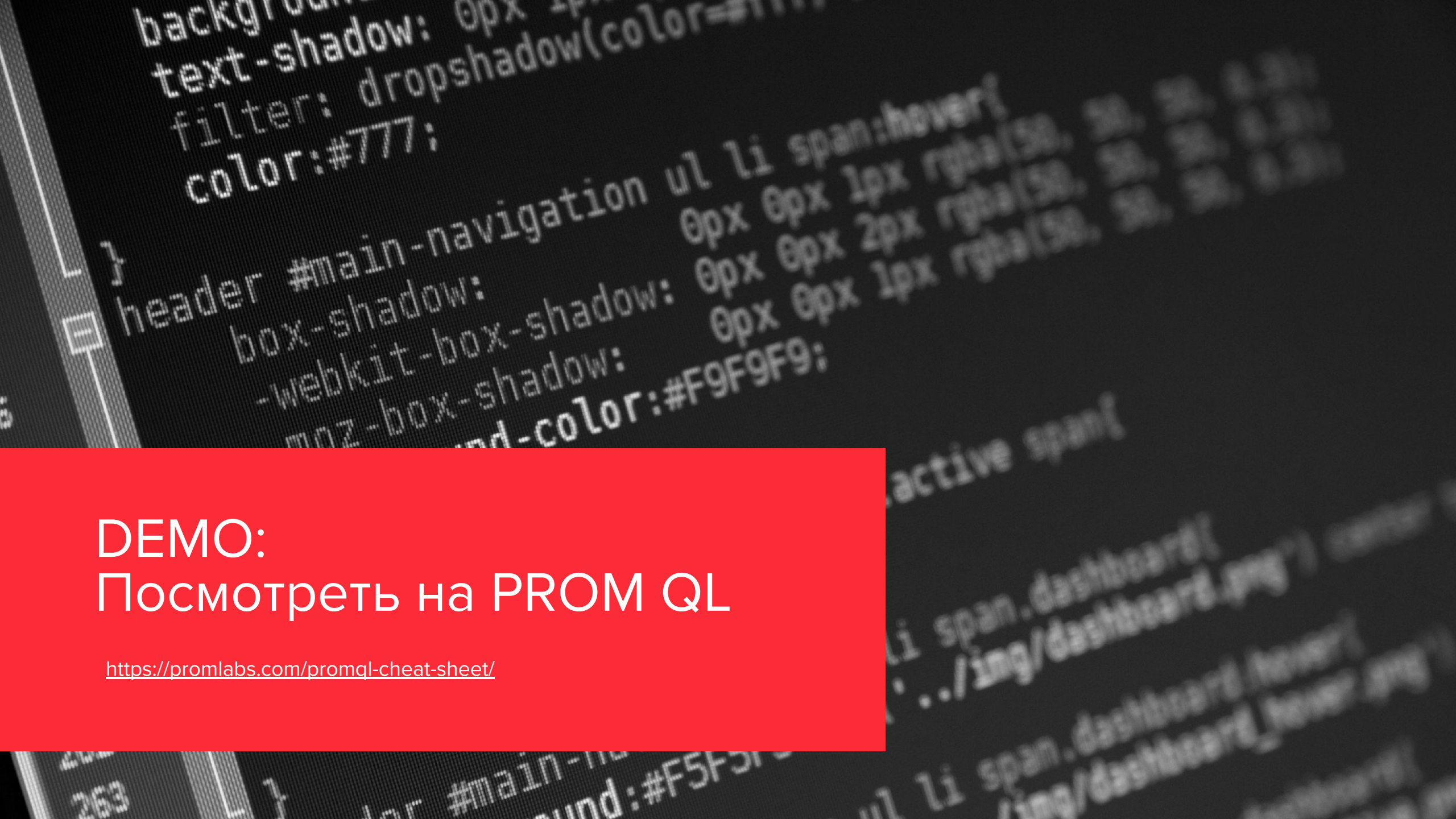
Open in PromLens

Available aggregation operators: `sum()`, `min()`, `max()`, `avg()`, `stddev()`, `stdvar()`, `count()`, `count_values()`, `group()`, `bottomk()`, `topk()`, `quantile()`

PROM QL VS SQL

```
node_network_receive_bytes_total{device="eth0"}
node_network_receive_bytes_total{device="eth1"}
node_network_receive_bytes_total{device="eth2"}
```

```
SELECT
  ts.metric_name_plus_tags,
  r.timestamps,
  r.values
FROM (
  (SELECT
    time_series_id,
    array_agg(timestamp ORDER BY timestamp) AS timestamps,
    array_agg(value ORDER BY timestamp) AS values
  FROM
    metrics
  WHERE
    time_series_id IN (
      SELECT id FROM time_series
      WHERE metric_name = 'node_network_receive_bytes_total'
    )
  GROUP BY
    time_series_id
  )
) AS r JOIN time_series AS ts ON (r.time_series_id = ts.id)
```



DEMO:
Посмотреть на PROM QL

https://promlabs.com/promql-cheat-sheet/

GRAFANA

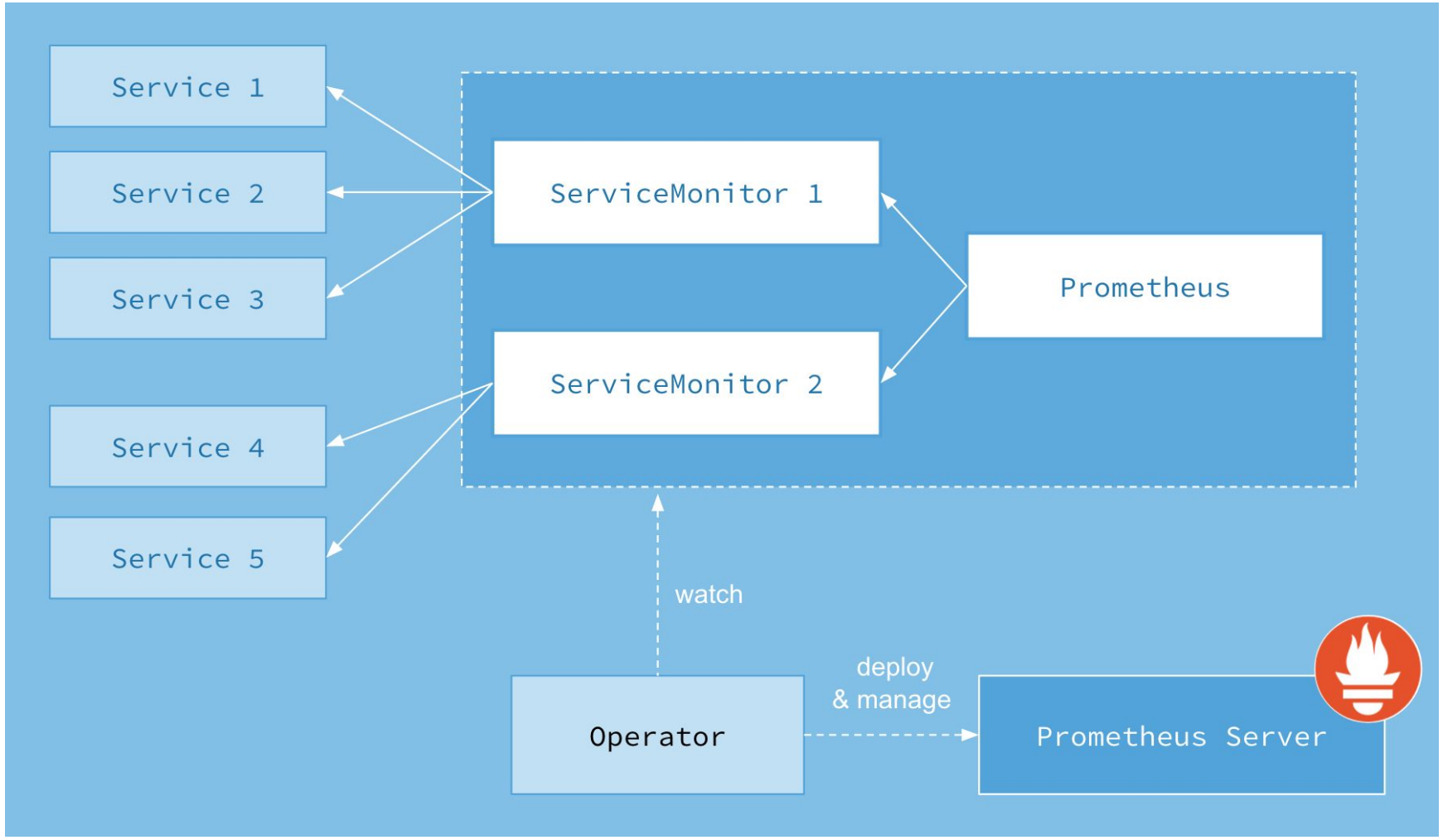
https://play.grafana.org/d/000000012/grafana-play-home?orgId=1

Популярнейший инструмент построения дашбордов, имеет интеграцию с Prometheus

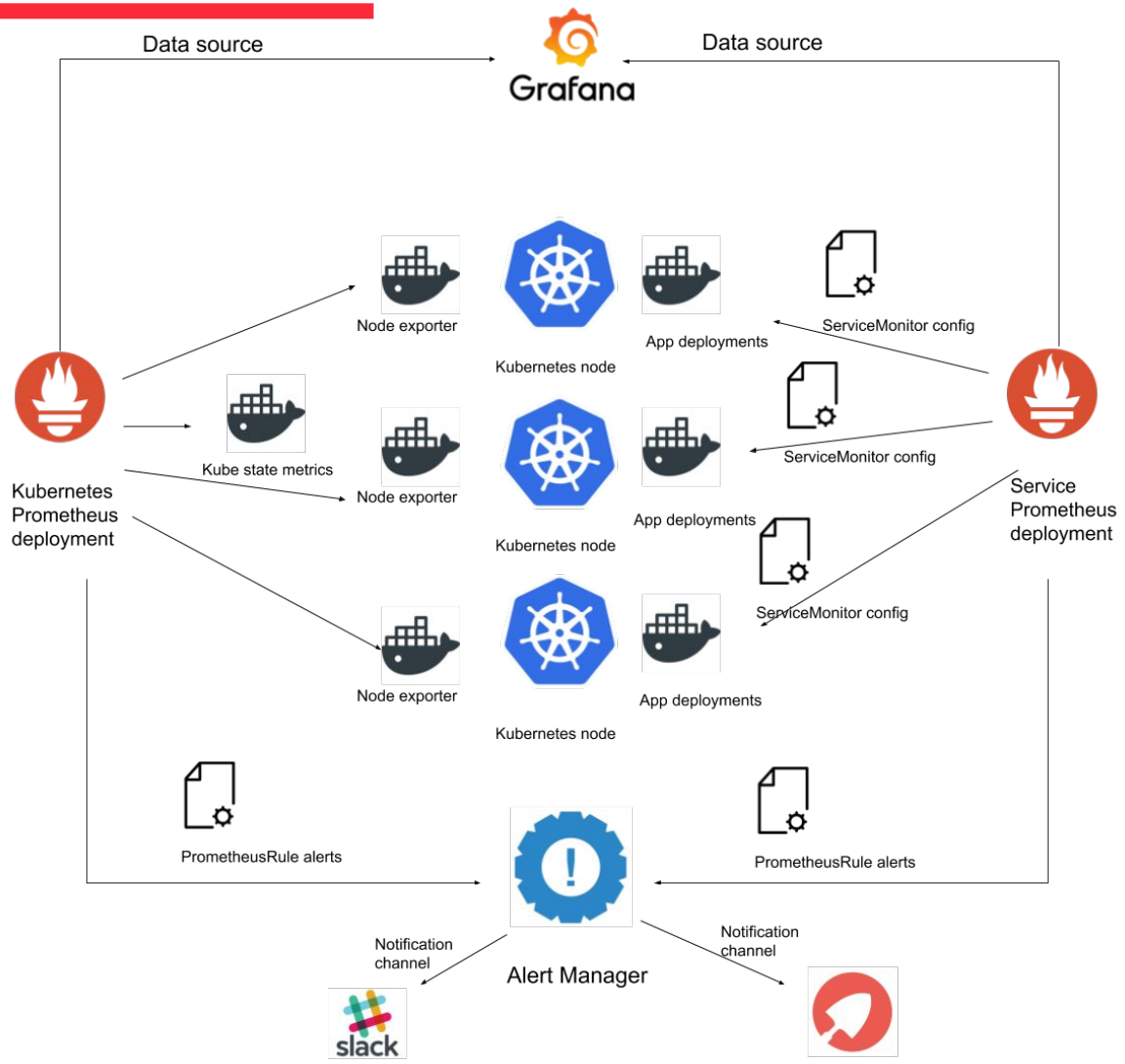
DEMO: GRAFANA

https://play.grafana.org/

PROMETHEUS OPERATOR

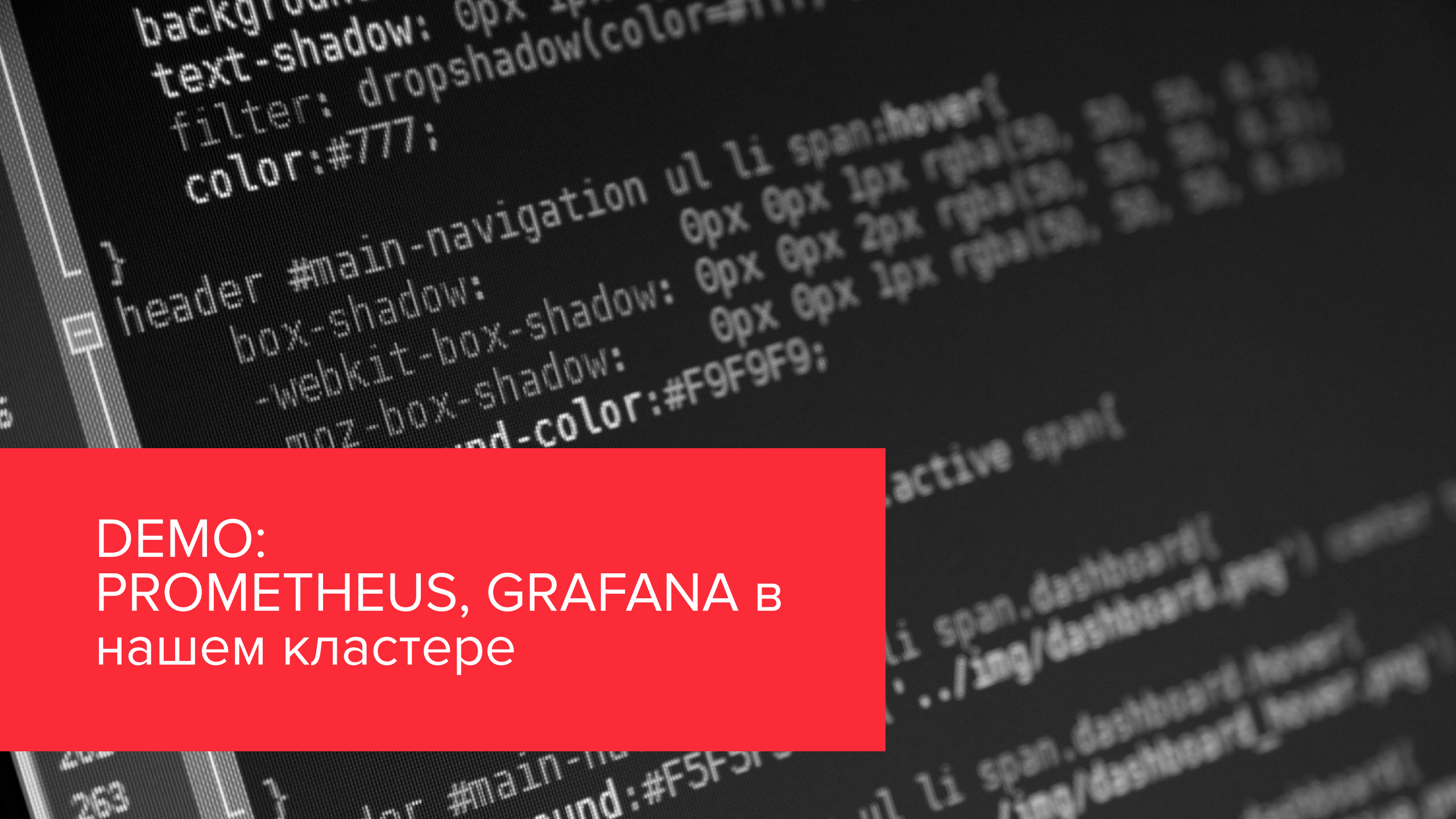


PROMETHEUS OPERATOR



SERVICE MONITOR

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: example-app
  labels:
    team: frontend
spec:
  selector:
    matchLabels:
      app: example-app
  endpoints:
  - port: web
```



DEMO: PROMETHEUS, GRAFANA в нашем кластере

ALERT MANAGER

Есть возможность создавать алерты и гибко настраивать уведомления о НИХ

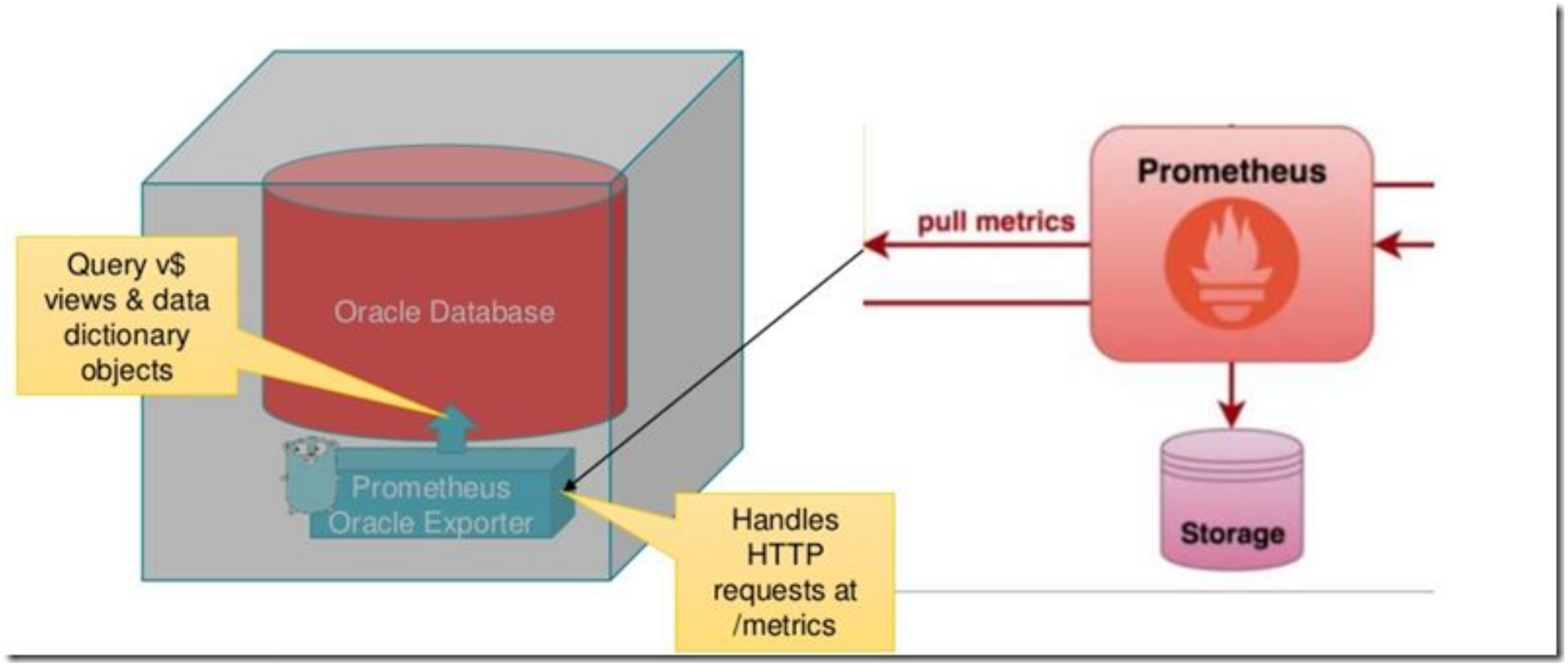
```
groups:
- name: example
  rules:
  - alert: HighRequestLatency
    expr: job:request_latency_seconds:mean5m{job="myjob"} > 0.5
    for: 10m
    labels:
      severity: page
    annotations:
      summary: High request latency
```

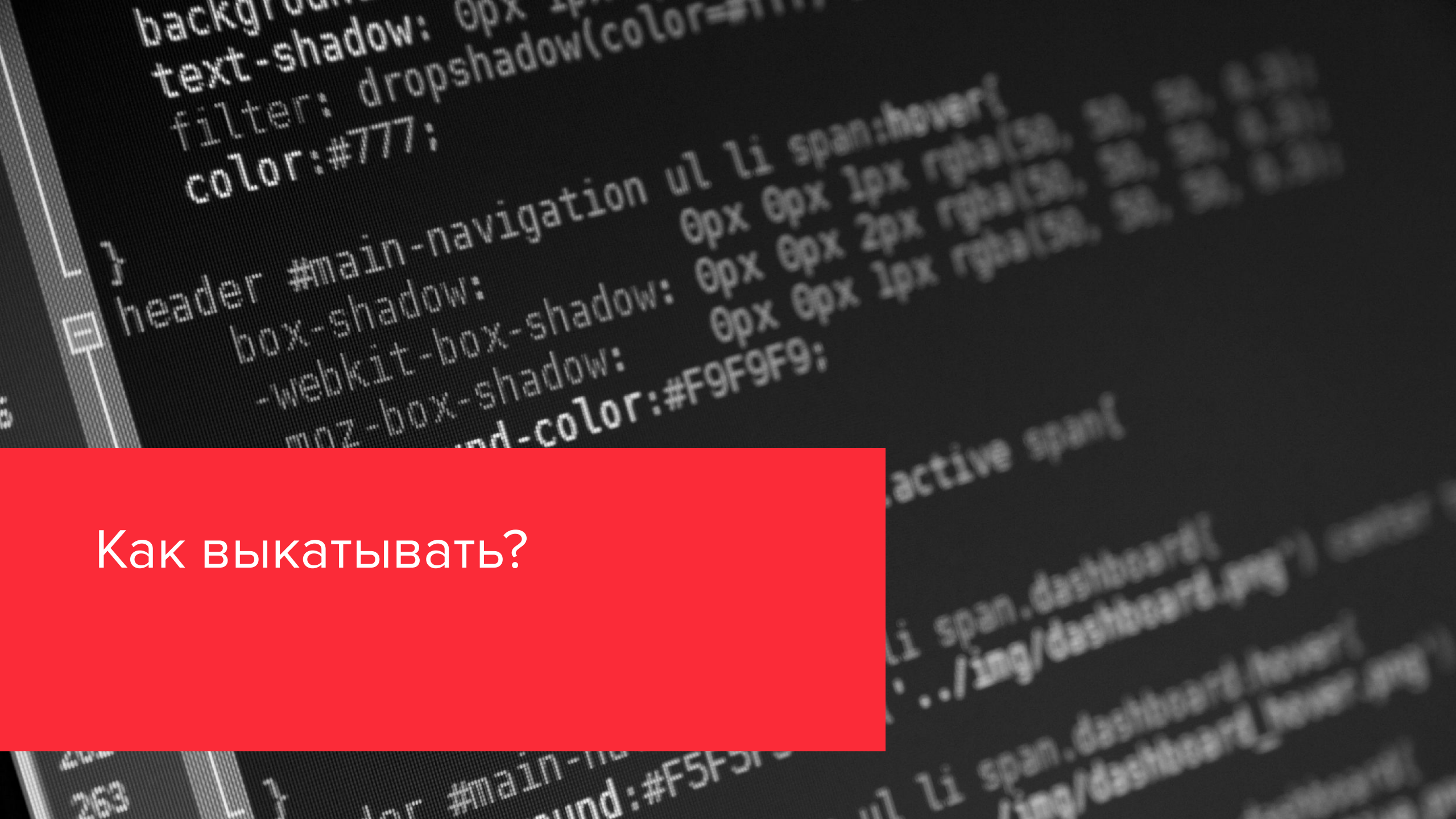
Airflow

Airflow умеет отдавать метрики в формате прометея, что позволяет встраивать его в систему мониторинга

https://airflow.apache.org/docs/apache-airflow/stable/logging-monitoring/metrics.html#counters

Exporters


Query v$ views & data dictionary objects
Oracle Database
Prometheus Oracle Exporter
Handles HTTP requests at /metrics
pull metrics
Prometheus
Storage


Как выкатывать?

Progressive Delivery

Механизм доставки кода до Production, при котором пользователи получают доступ к новому функционалу постепенно (progressively):

- percentage rollouts pattern - рост трафика на новый релиз осуществляется гранулированно и, возможно, только в случае успешного прохождения определенных проверок

- feature flags - отделение доставки кода до прода от момента доступности изменений для конечных пользователей

- ring deployments - фичи становятся доступны для ограниченной группы пользователей (internal users -> canary group -> beta testers -> general release). (c) Microsoft

Стратегии деплоя

- recreate
- rolling-update
- blue/green
- canary

RECREATE

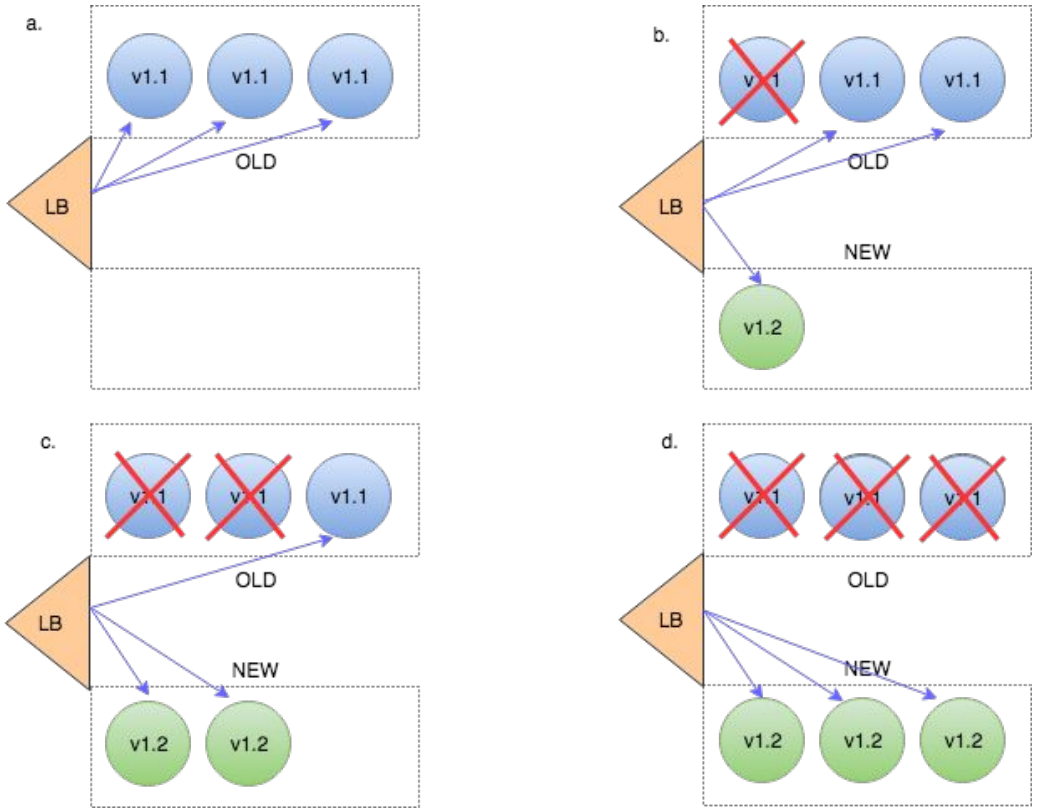
погасили версию А, поставили версию Б

+ лёгкая настройка
+ состояние приложения полностью обновлено
– downtime

ROLLING-UPDATE

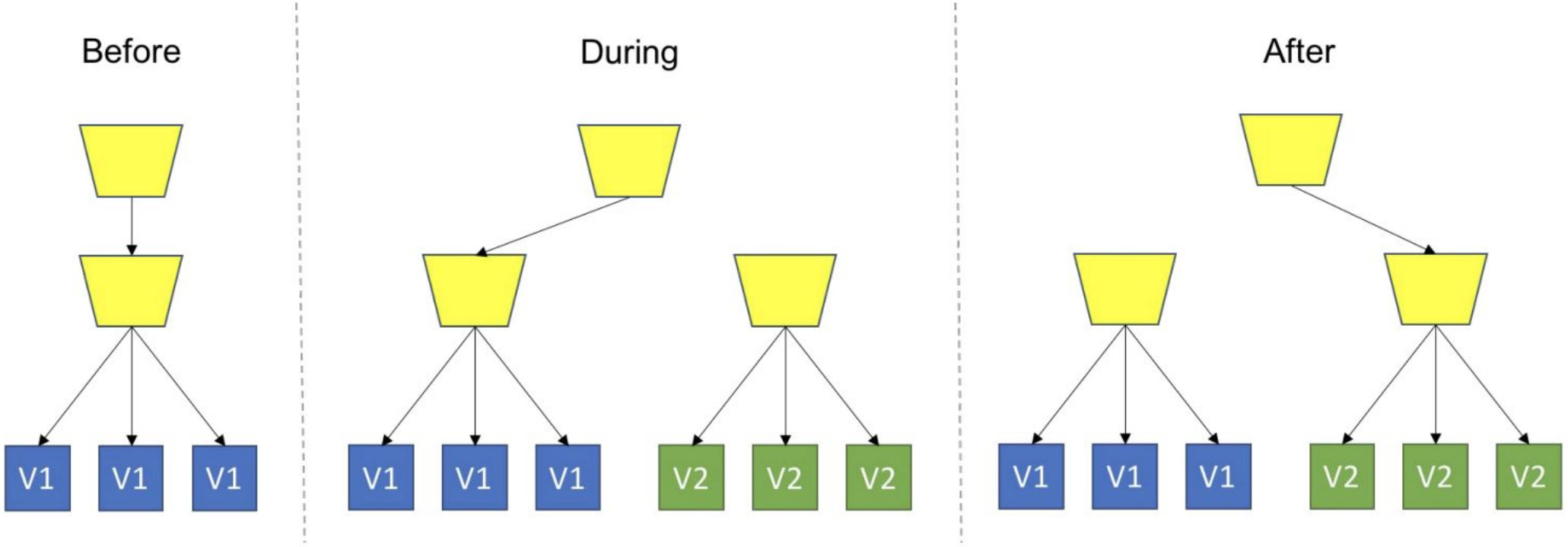
версия Б медленно выкатывается в прод и заменяет версию А

+лёгкая настройка
+ версия неспешно катится по всем инстансам
- может занять долгое время
– саппорт двух версий API
– плохой контроль трафика



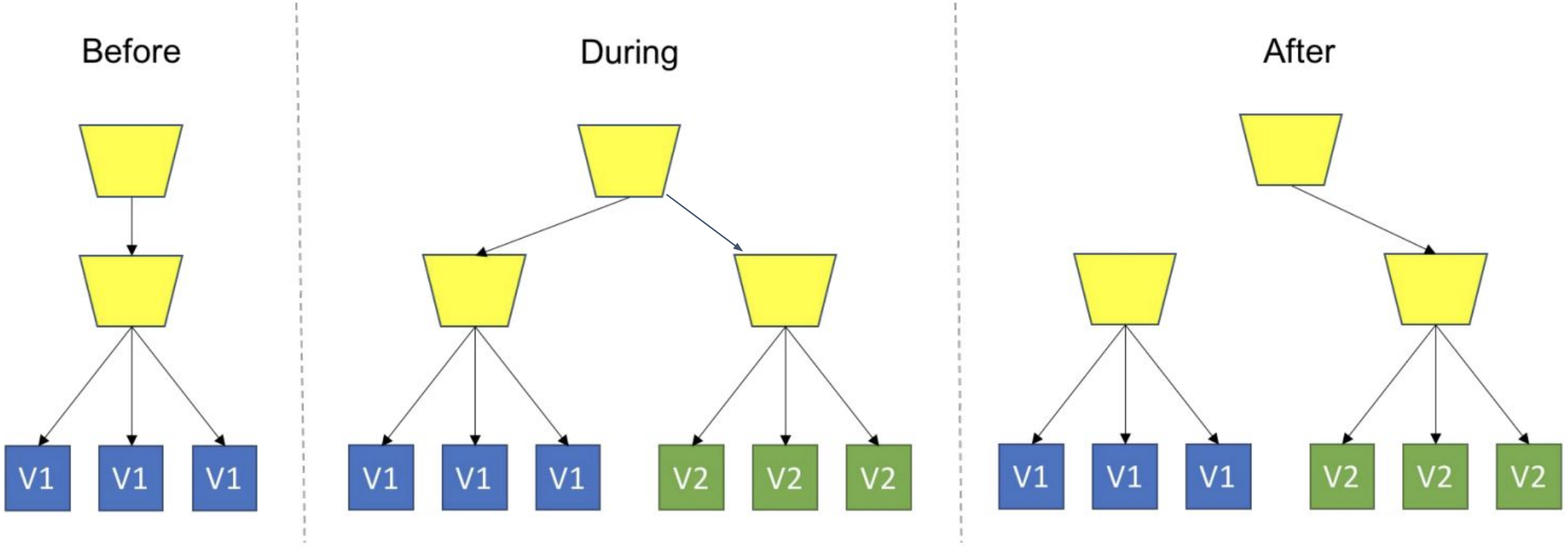
Blue-Green

Включаем и старую и новую
модель
+ мгновенное переключение
− дорого и требует дублицирования ресурсов



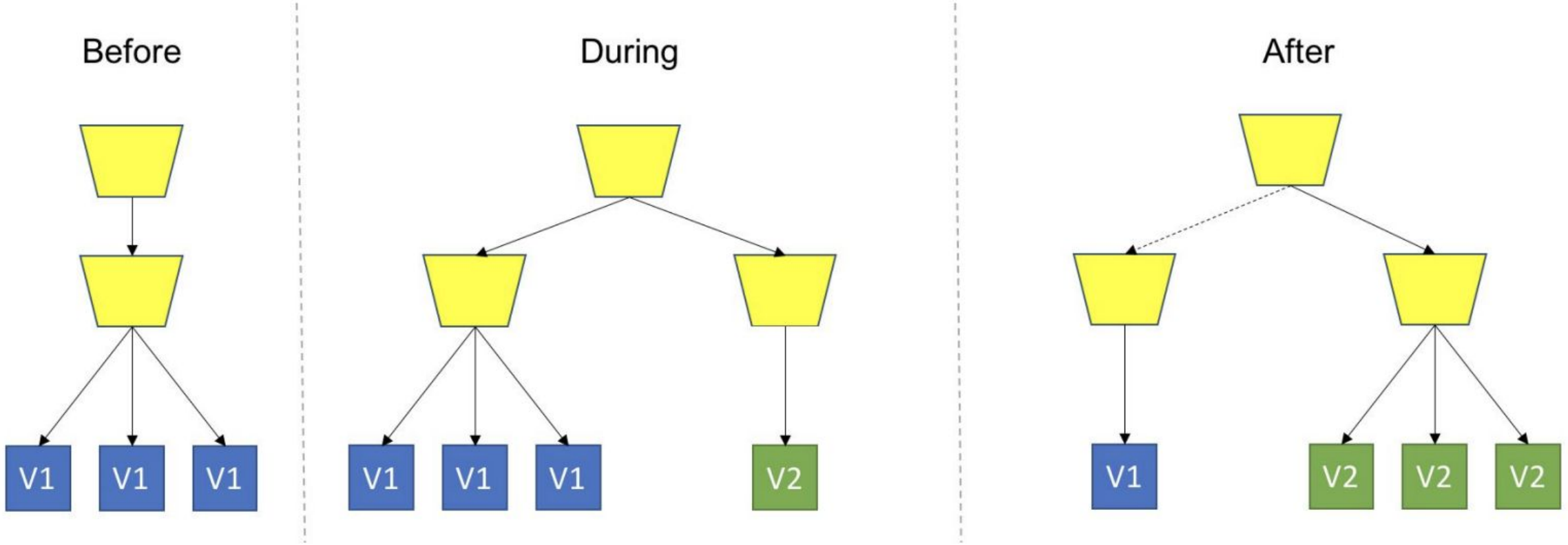
Shadow

Тоже самое, что BLUE-GREEN, но GREEN обрабатывает трафик



Canary

- выкатываем мало инстансов модели на часть пользователей и если что — сразу откатываем



DEPLOYMENT STRATEGIES

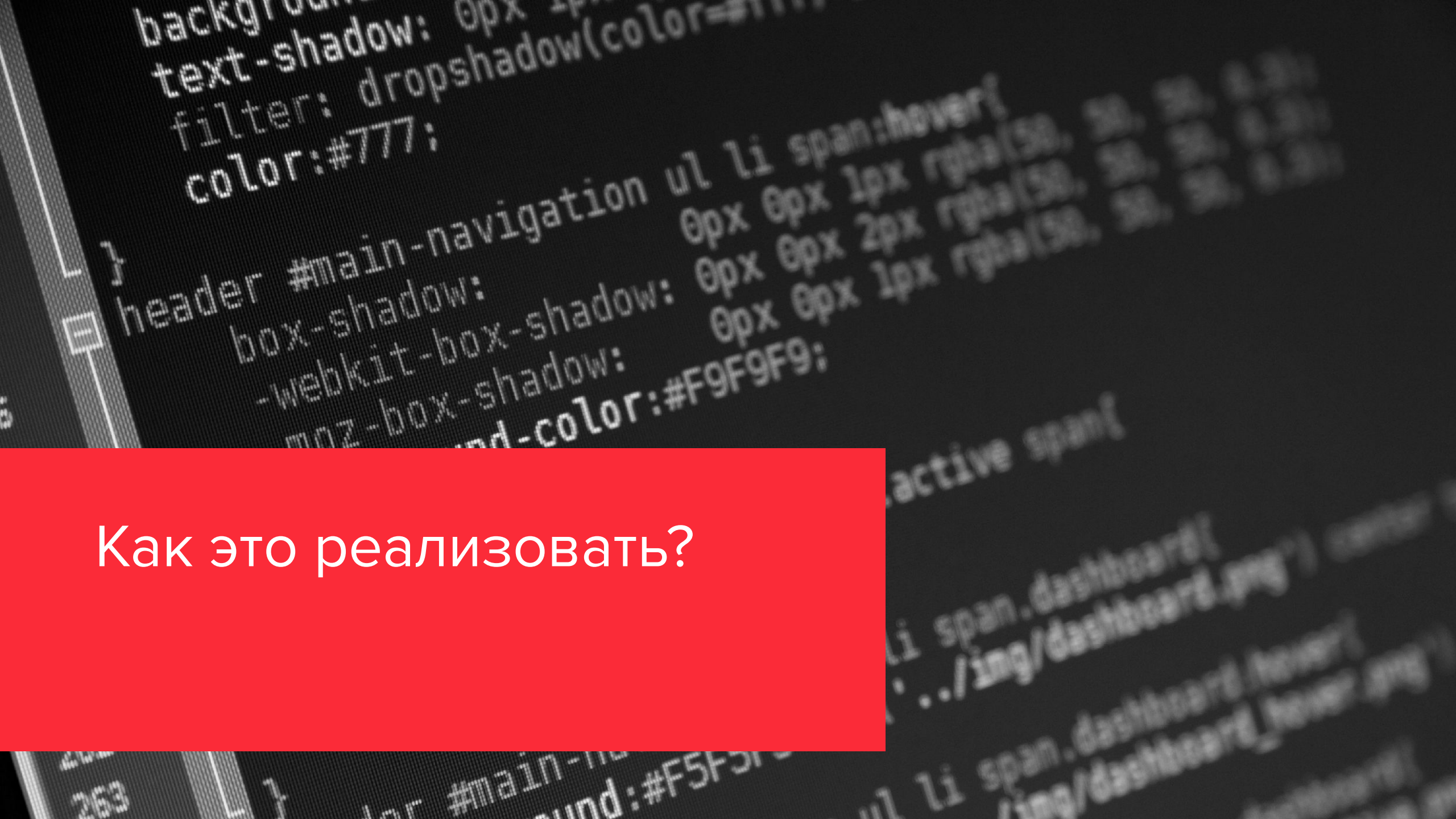
When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.

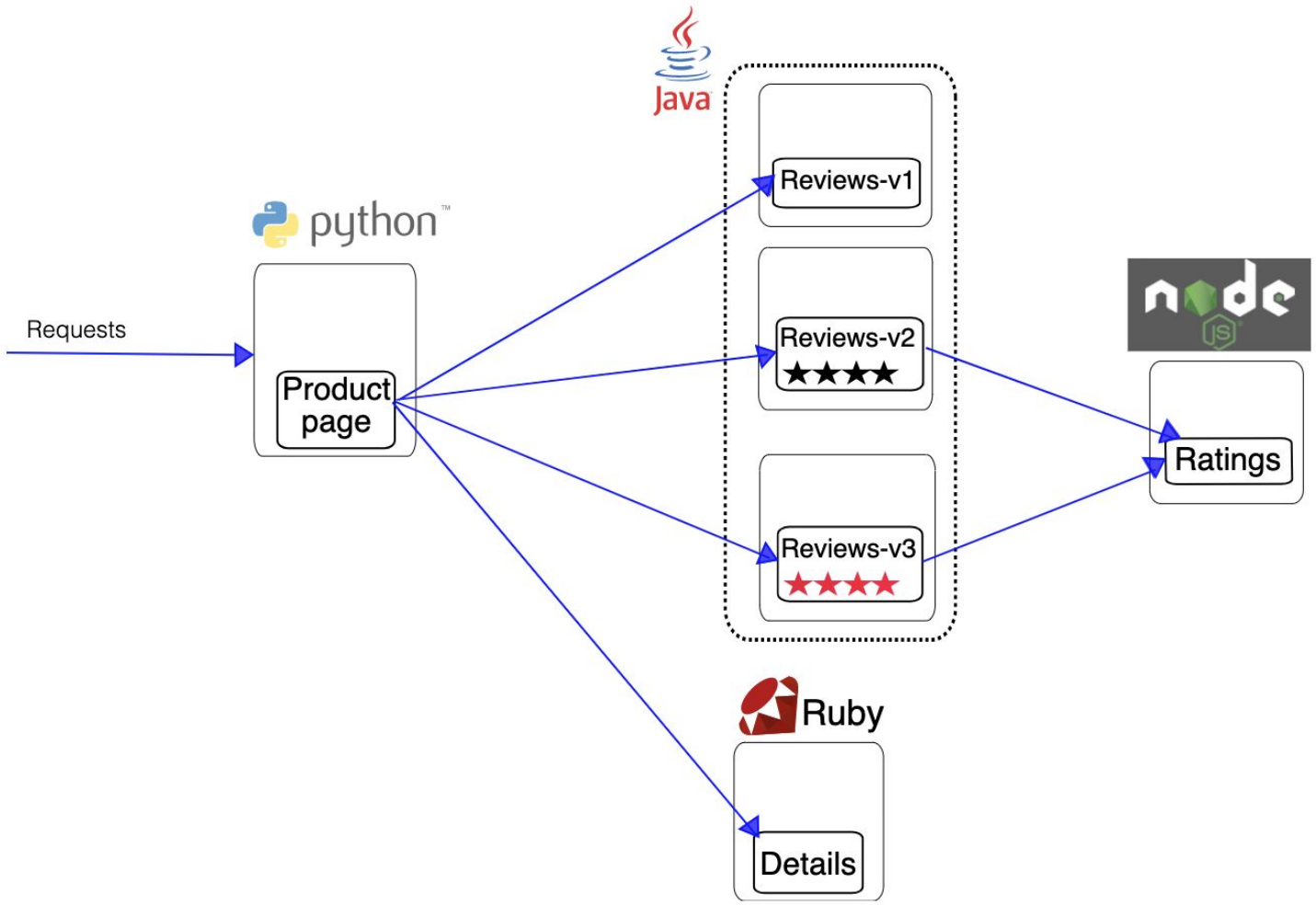
Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■□□	■■■	■■■	□□□
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■□□	■■■	■□□	■□□
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■■■	□□□	■■□	■■□
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■□□	■□□	■□□	■■□
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■□□	■□□	■□□	■■■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■■■	□□□	□□□	■■■

Container Solutions



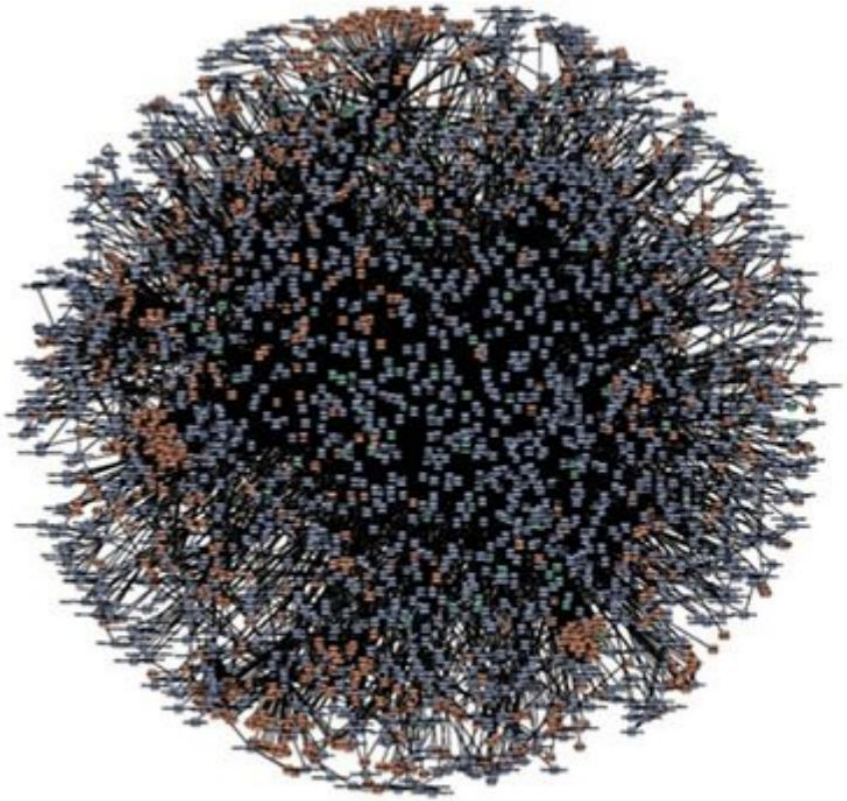
Как это реализовать?

Типовое микросервисное приложение



Bookinfo Application without Istio

Общая картина



amazon.com

NETFLIX

Service Mesh

Выделенный слой инфраструктуры для обеспечения безопасного, быстрого и надёжного взаимодействия между сервисами.

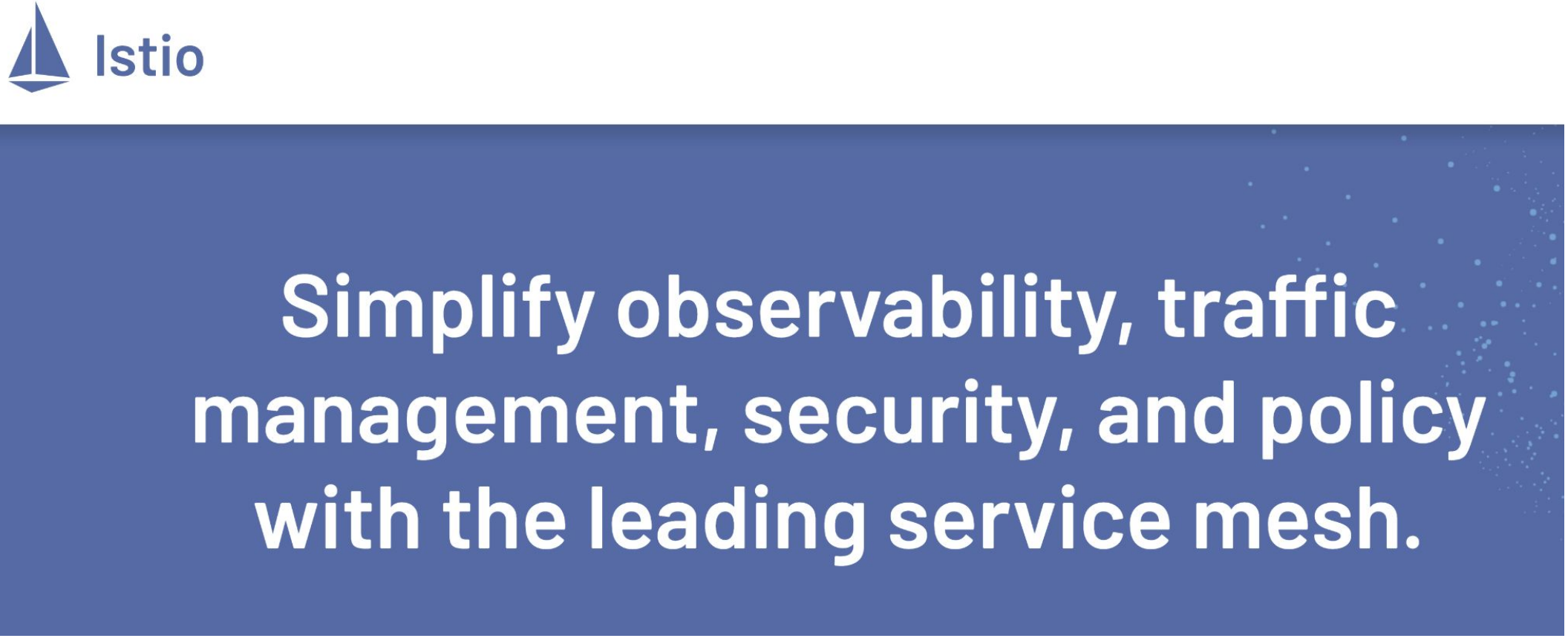
Service mesh (также встречаются термины «сеть микросервисов», «mesh-сеть микросервисов») — это абстрактный слой инфраструктуры, который определяет взаимодействие между микросервисами приложения.

Наши потребности

- Observability
- Управление трафиком (Splitting, mirroring)
- Security
- Обработка сетевых ошибок не на уровне приложения (retry, etc)
- Сбор метрик

Istio

Самая популярная open source реализация service mesh



Istio

Предоставляет возможности для

- гибкого управления трафиков
- observability
- Security

Traffic management

Позволяет по гибким правилам маршрутизировать трафик.

Пример:
Направить 10% запросов на новую модель, 90% на старую
Если пользователь относится к группе Б, то с направить его на новую модель.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v3
```

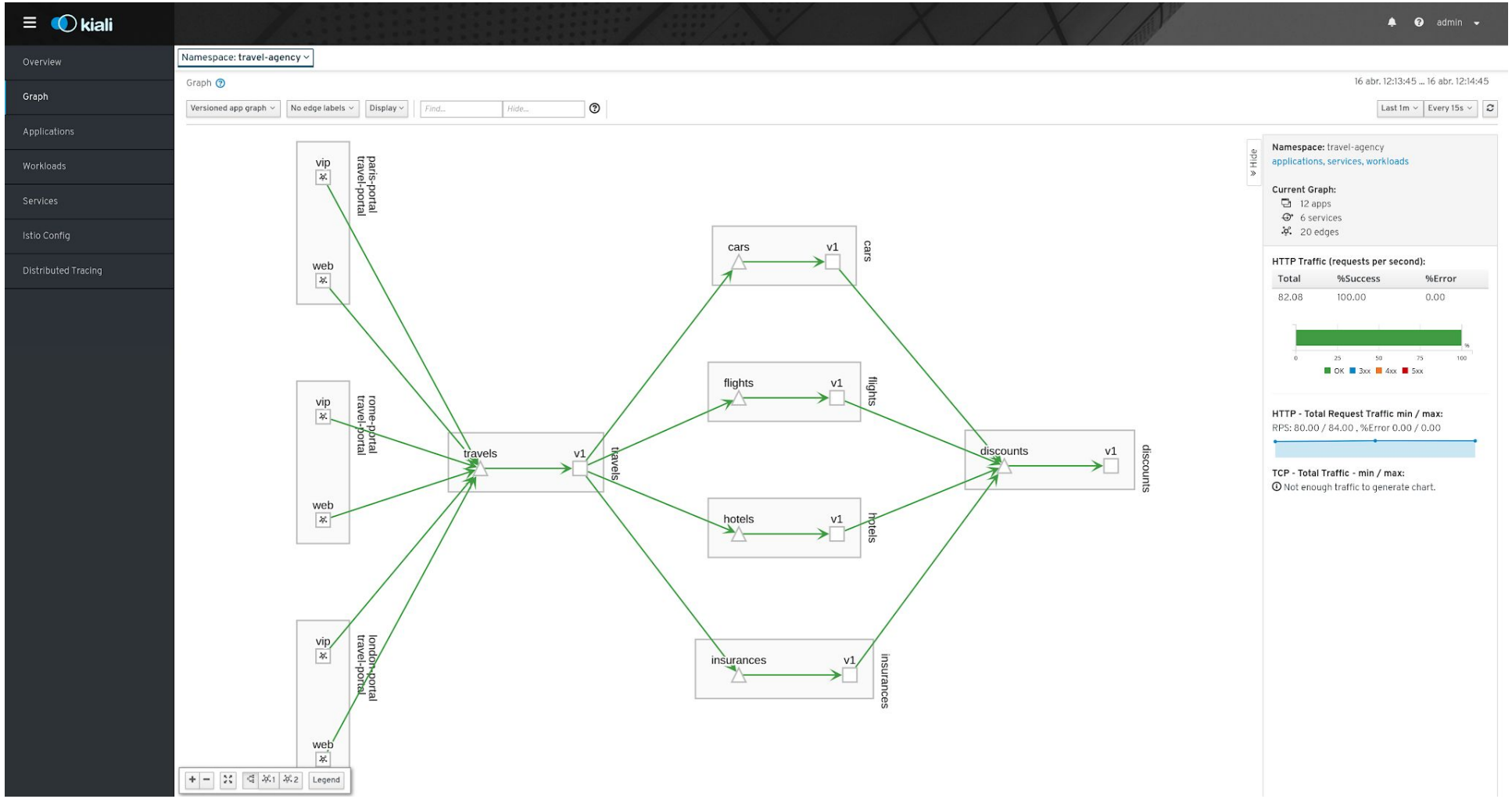
```yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: reviews-cb-policy
spec:
  host: reviews.prod.svc.cluster.local
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 100
      http:
        http2MaxRequests: 1000
        maxRequestsPerConnection: 10
    outlierDetection:
      consecutive5xxErrors: 7
      interval: 5m
      baseEjectionTime: 15m
```

Observability

Istio generates the following types of telemetry in order to provide overall service mesh observability:

- **Metrics**. Istio generates a set of service metrics based on the four "golden signals" of monitoring (latency, traffic, errors, and saturation). Istio also provides detailed metrics for the mesh control plane. A default set of mesh monitoring dashboards built on top of these metrics is also provided.
- **Distributed Traces**. Istio generates distributed trace spans for each service, providing operators with a detailed understanding of call flows and service dependencies within a mesh.
- **Access Logs**. As traffic flows into a service within a mesh, Istio can generate a full record of each request, including source and destination metadata. This information enables operators to audit service behavior down to the individual workload instance level.

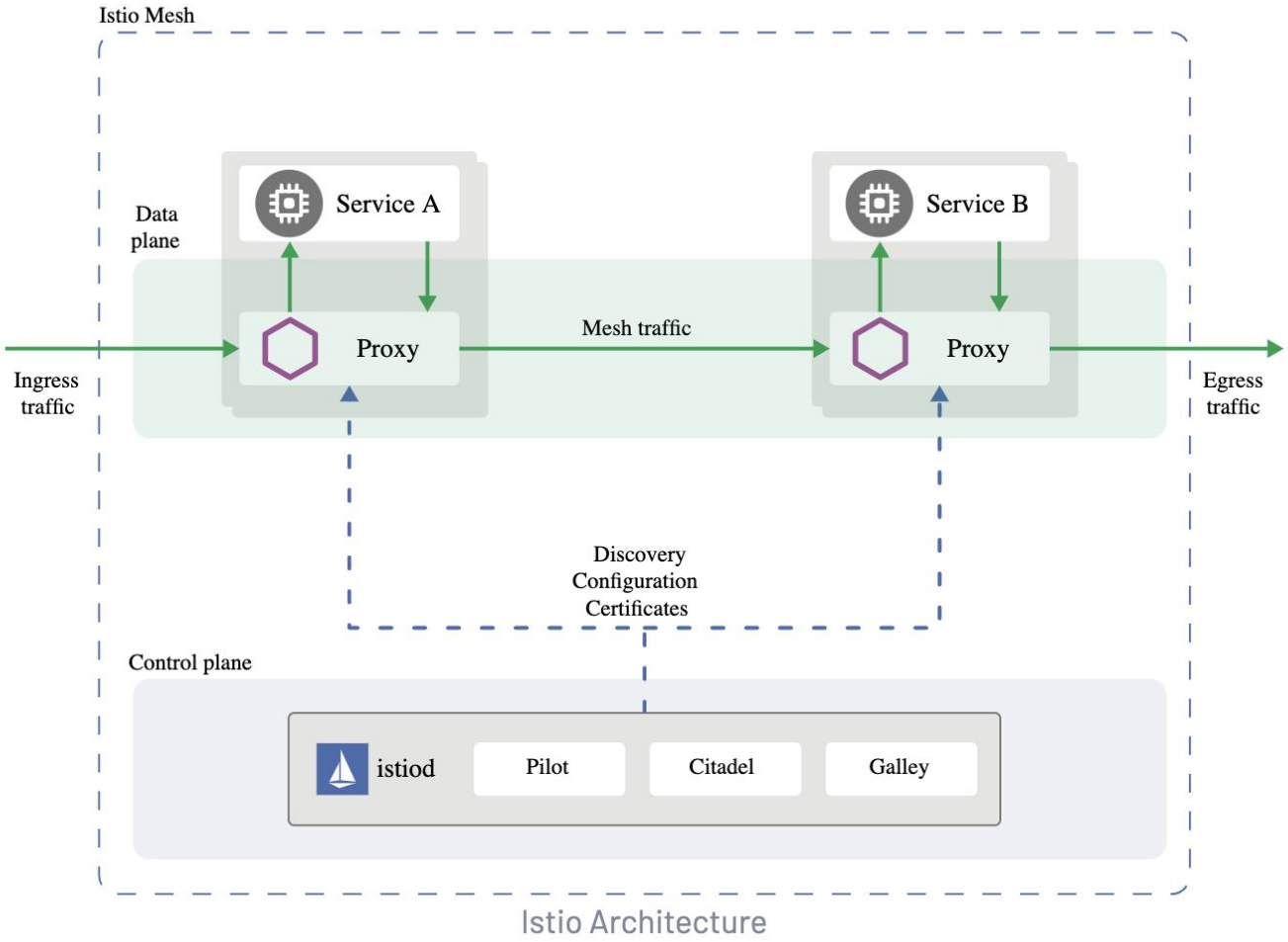
Kiali


kiali
admin
Overview
Graph
Applications
Workloads
Services
Istio Config
Distributed Tracing
Namespace: travel-agency
Graph
16 abr. 12:13:45 ... 16 abr. 12:14:45
Versioned app graph
No edge labels
Display
Find...
Hide...
Last 1m
Every 15s
vip
web
paris-portal
travel-portal
rome-portal
london-portal
travels
v1
cars
flights
hotels
insurances
discounts
Legend
Hide
Namespace: travel-agency
applications, services, workloads
Current Graph:
12 apps
6 services
20 edges
HTTP Traffic (requests per second):
Total
%Success
%Error
82.08
100.00
0.00
OK
3xx
4xx
5xx
HTTP - Total Request Traffic min / max:
RPS: 80.00 / 84.00 , %Error 0.00 / 0.00
TCP - Total Traffic - min / max:
Not enough traffic to generate chart.

Demo c Istio

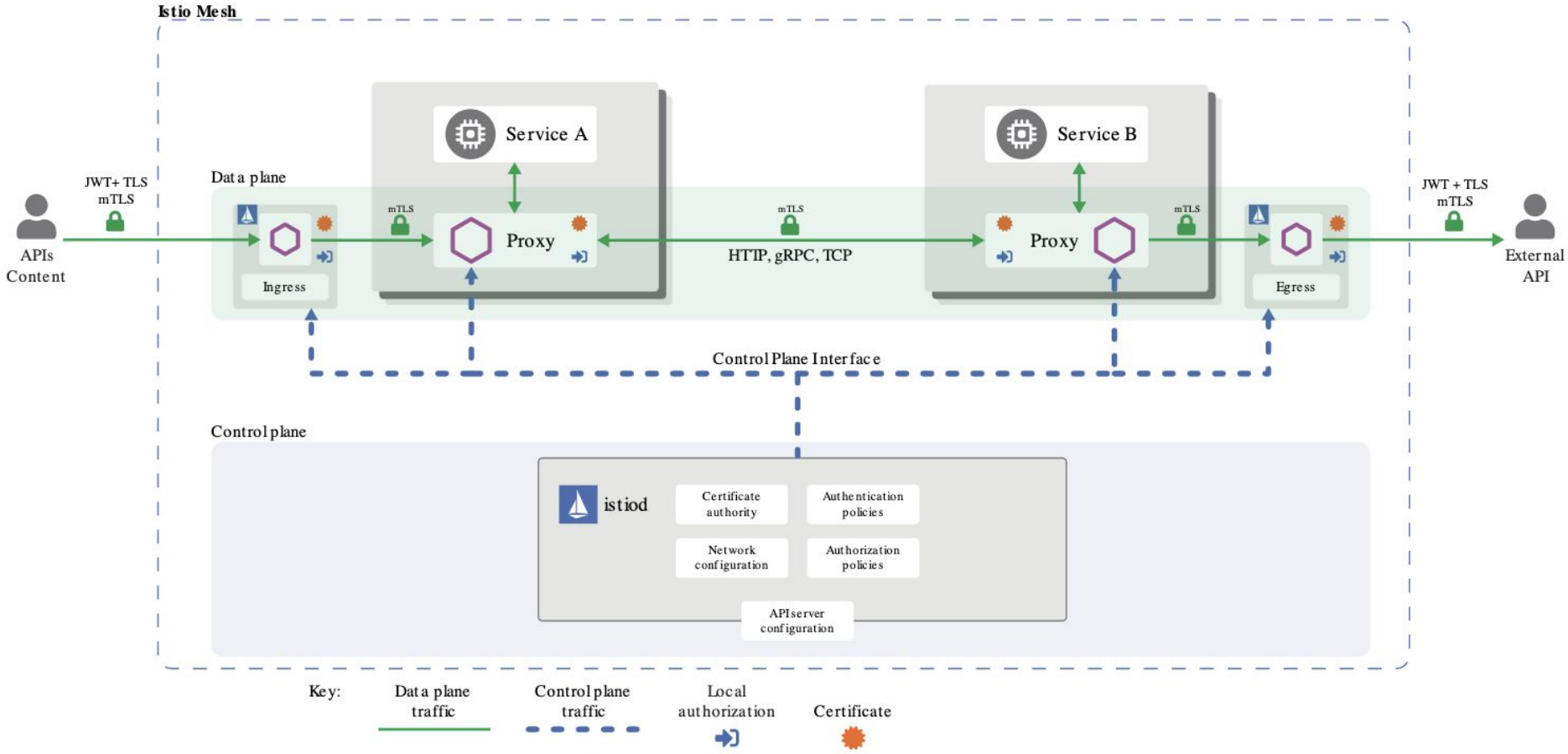
https://istio.io/latest/docs/setup/getting-started/
https://istio.io/latest/docs/tasks/traffic-management/traffic-shifting/

Istio Architecture



https://istio.io/latest/docs/ops/deployment/architecture/

Security

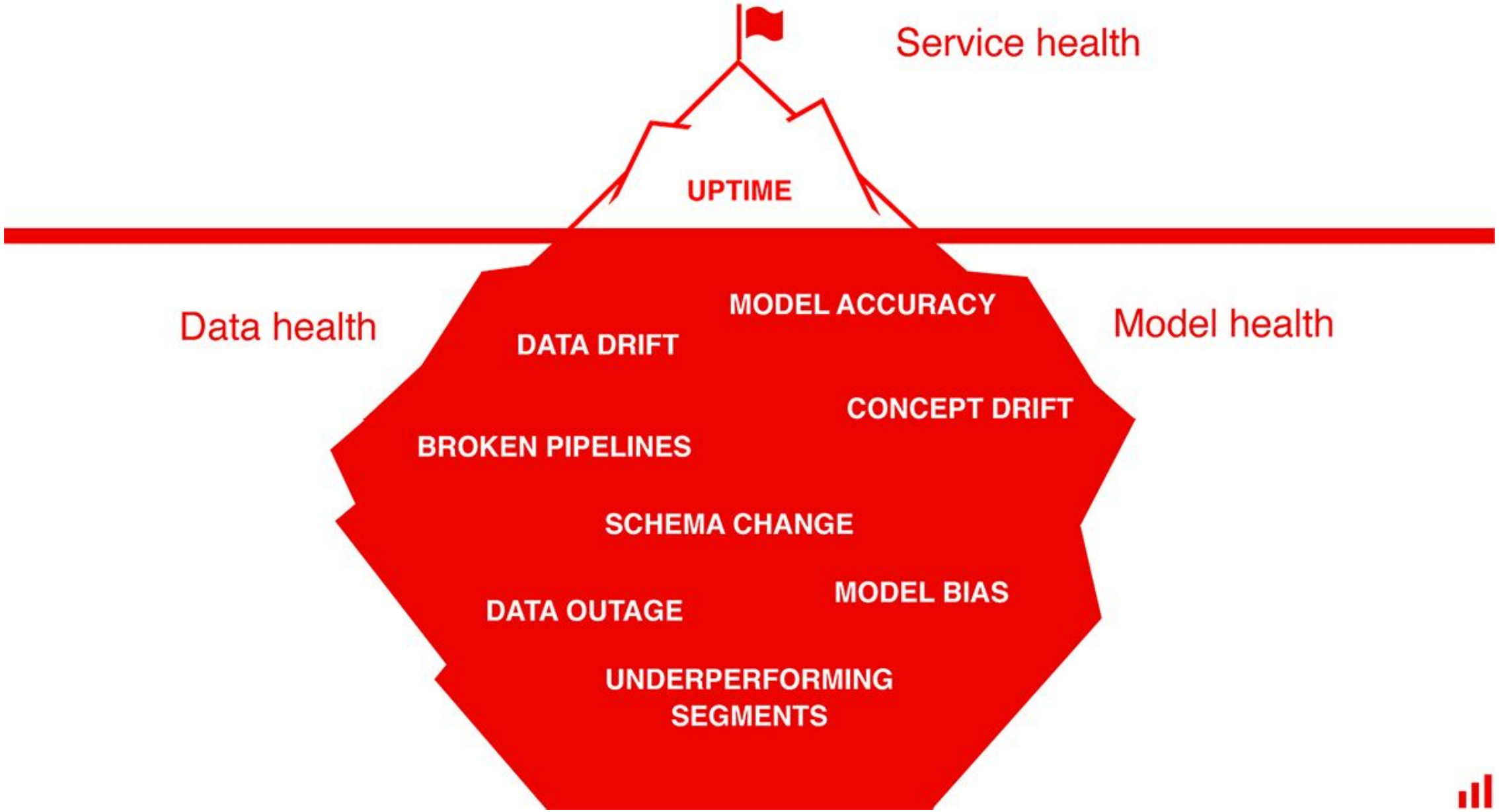


Security Architecture

Istio and Service mesh simply — explained

Итоги

Совмещая мониторинг состояния через систему мониторинга и практики постепенных выкаток — человечество немного приблизилось к возможности спать спокойно!


Service health
UPTIME
Data health
MODEL ACCURACY
DATA DRIFT
Model health
CONCEPT DRIFT
BROKEN PIPELINES
SCHEMA CHANGE
MODEL BIAS
DATA OUTAGE
UNDERPERFORMING
SEGMENTS

Как определить что модель деградировала?

Нужно мерять метрики машинного обучения в онлайне

Как часто нужно переобучать модель?

Зависит от
- предметной области
- стоимости
- от степени деградации

Как собрать GT?

1) Напрямую из данных
2) Разметить

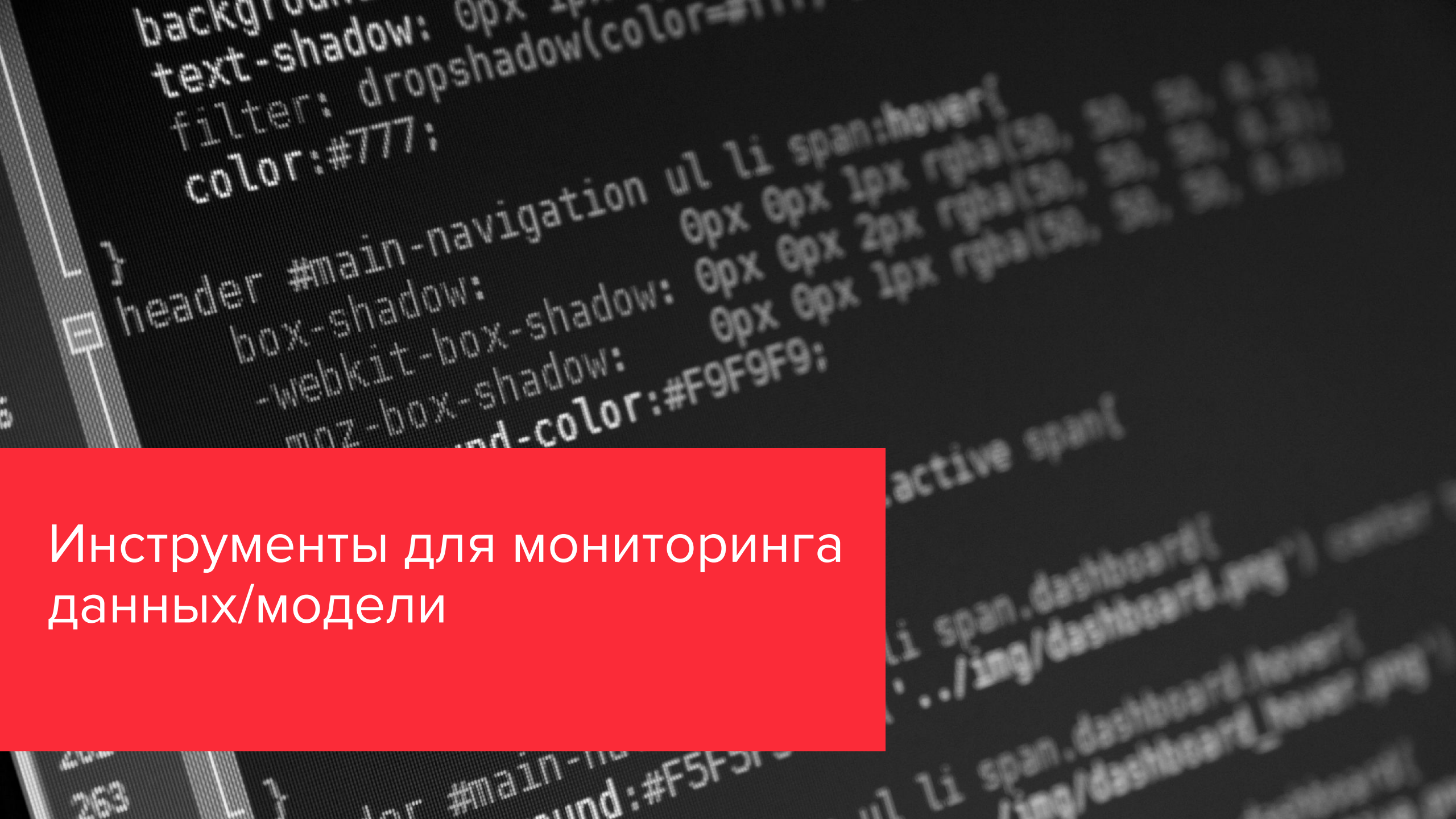
Яндекс
Толока

Почему модель деградировала?

- Технические ошибки
- Данные, которые идут на вход изменились

Что валидировать в данных?

Пропуски
Дубли,
Отличия в распределениях
Недопустимые значения
Отсутствие аномалий



Инструменты для мониторинга данных/модели

GreatExpectation

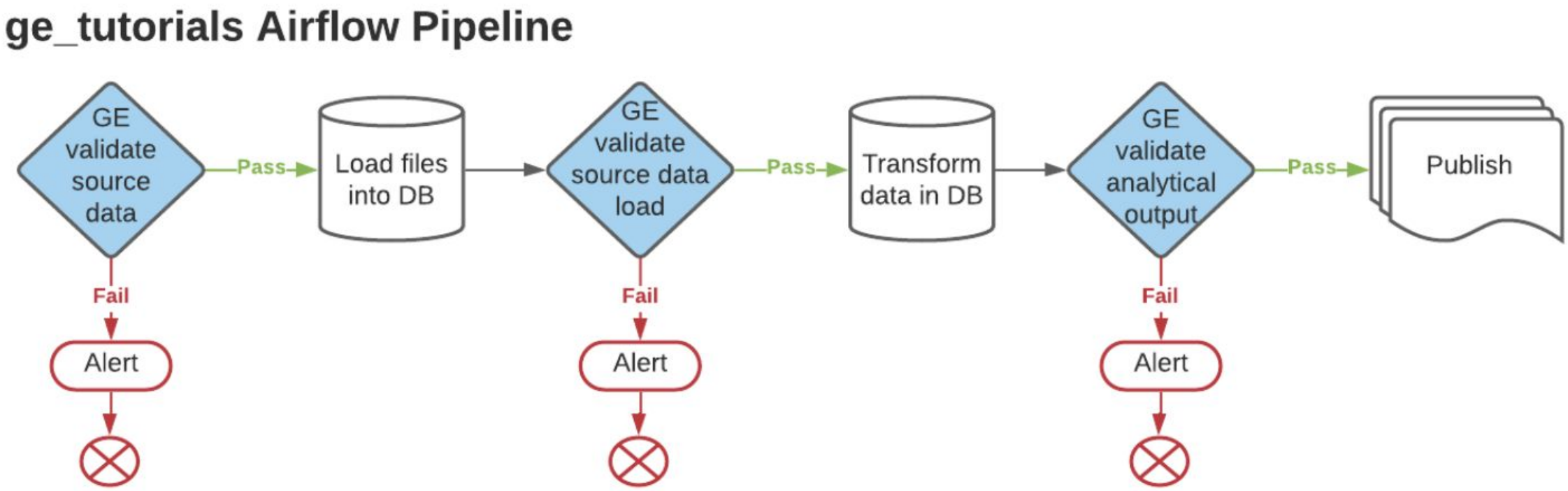
Инструмент, позволяющий писать проверки для данных
Имеет интеграцию со spark

```
expect_column_values_to_be_between(
    column="room_temp",
    min_value=60,
    max_value=75,
    mostly=.95
)
```


"Values in this column should be between 60 and 75, at least 95% of the time."

"Warning: more than 5% of values fell outside the specified range of 60 to 75."

GreatExpectation



GreatExpectation is NOT

Great Expectations is NOT a pipeline execution framework.

We integrate seamlessly with DAG execution tools such as Airflow, dbt, Prefect, Dagster, Kedro, etc. Great Expectations does not execute your pipelines for you, but instead, validation can simply be run as a step in your pipeline.

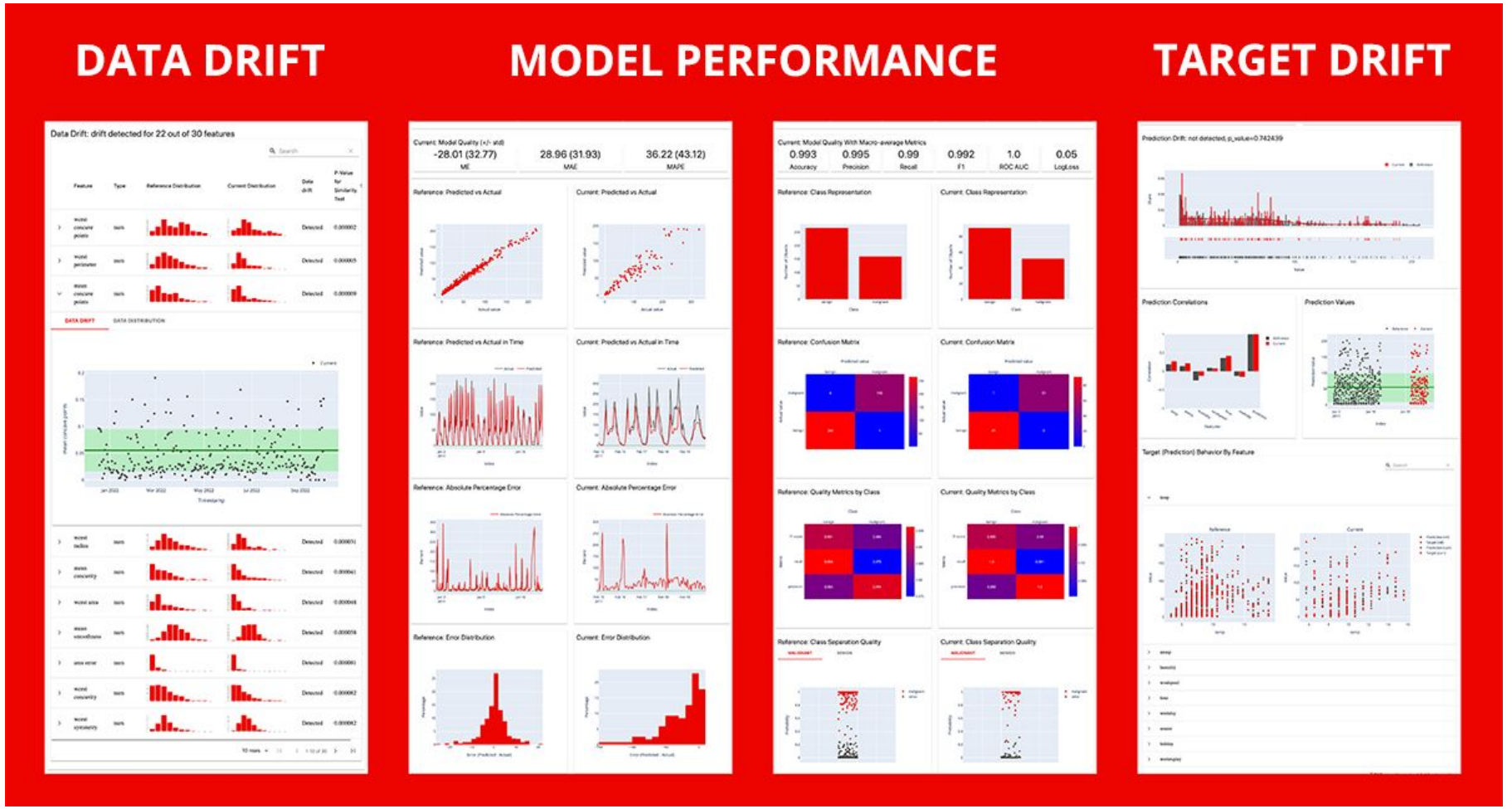
Great Expectations is NOT a data versioning tool.

Great Expectations does not store data itself. Instead, it deals in metadata about data: Expectations, validation results, etc. If you want to bring your data itself under version control, check out tools like: DVC and Quilt.

Great Expectations currently works best in a Python environment.

Great Expectations is Python-based. You can invoke it from the command line without using a Python programming environment, but if you're working in another ecosystem, other tools might be a better choice. If you're running in a pure R environment, you might consider assertR as an alternative. Within the TensorFlow ecosystem, TFDV fulfills a similar function as Great Expectations.

Evidently.AI



https://evidentlyai.com/

DeepChecks

Why Deepchecks?

Deepchecks is a comprehensive MLOps solution for continuous validation of ML systems. It's designed to help various stakeholders gain control of their ML systems, including data science leaders, ML engineers, analysts, software engineers, and more.

https://deepchecks.com/

Итоги

Валидация данных — это не менее важно, чем мониторинг метрик сервиса

Понимание этого факта — первично

Инструменты в этой области — это НОТ

академия больших данных

mail.ru group

Мониторинг, постепенные выкатки

Михаил Марюфич, MLE

https://forms.gle/iNY9UoexWReefuaH6