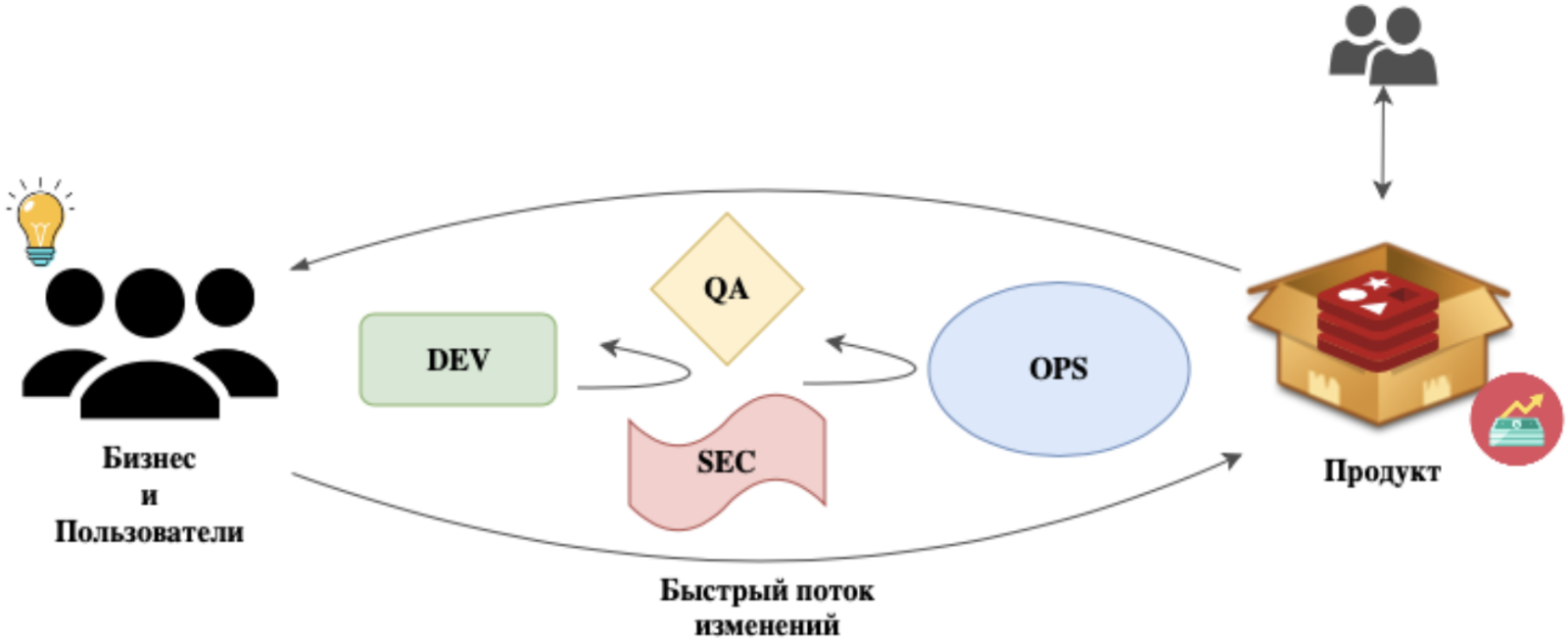
Kubernetes. Мониторинг. Логирование

План

- Какой должен быть мониторинг
- Мониторинг Kubernetes
  - Принципы мониторинга
  - Kubernetes Probes
  - Мониторинг Kubernetes
  - Prometheus Operator
  - Визуализация

Какой должен быть мониторинг

Обеспечить быструю обратную связь справа налево



Какой должен быть мониторинг

Плюсы быстрой обратной связи

- Повышает качество продукта
- Позволяет оперативно реагировать на значимые события/изменения
- Повышается скорость обучения внутри организации

Какой должен быть мониторинг

Примеры быстрой обратной связи

- Мониторинг окружений и работы приложения
- Мониторинг поведения пользователей
- Автоматизация сборки кода и прогона тестов
- ChatOps

Мониторинг должен

Должен удовлетворять потребности своих клиентов:

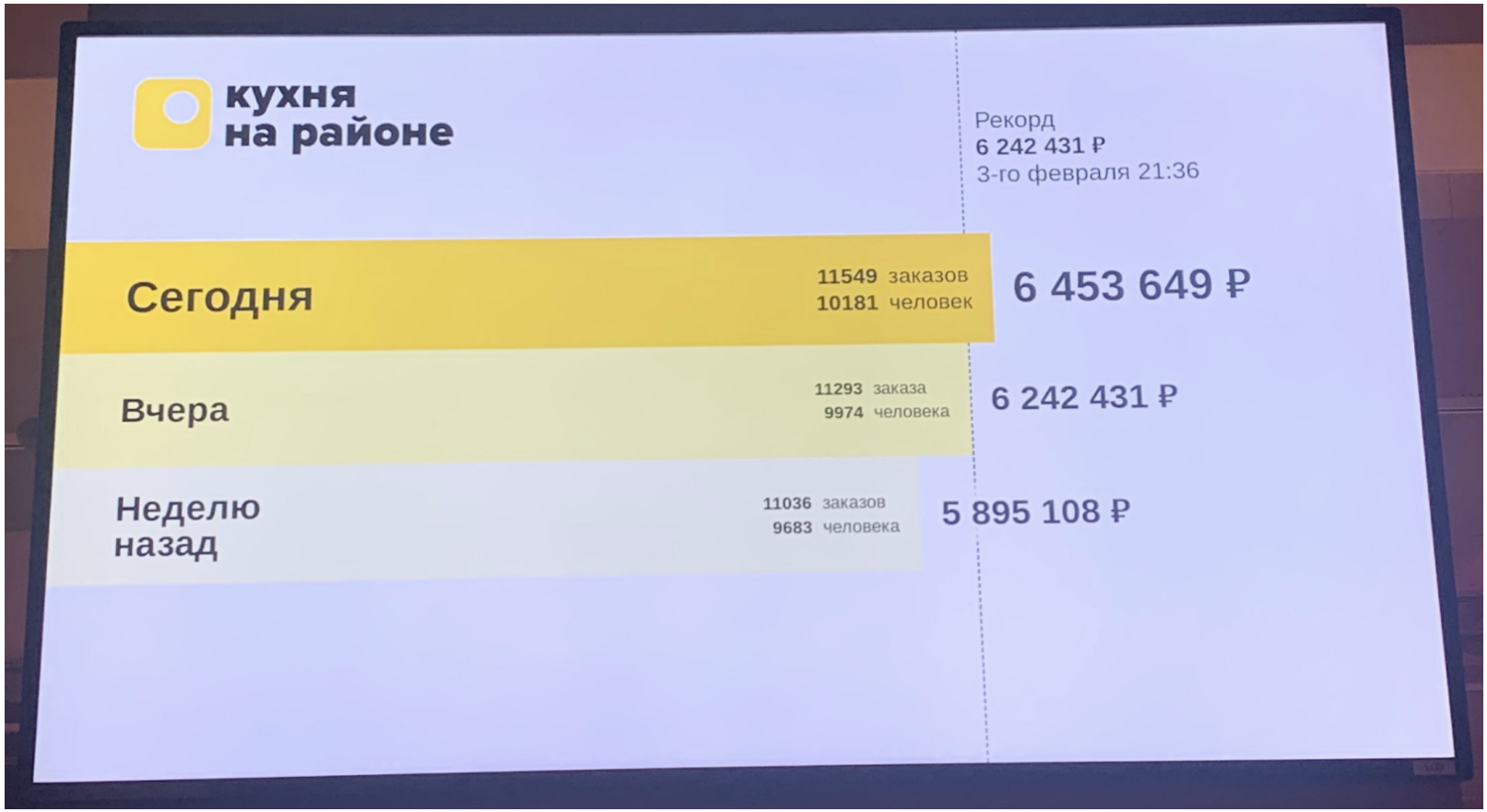
- Dev
- Ops
- Business

Какой должен быть мониторинг

Мониторинг

Совокупность инструментов и практик, позволяющая осуществлять контроль над работой IT систем и давать оценку качества работы этих систем.

Какие метрики



Мониторинг как сервис

Может быть:

- собственный сервис (Zabbix, Prometheus, ...)
- SaaS (New Relic, Datadog, okmeter.io)

Модели мониторинга Blackbox мониторинг

- Мониторинг извне с точки зрения пользователя
- Не видим, как работает система внутри
- Примеры: проверка открытых портов, подсчет коннектов, наличие процесса

Модели мониторинга Whitebox мониторинг

- Дополняет модель blackbox
- Мониторинг на основе информации о внутренней работе системы
- Примеры: метрики приложений (время запроса к БД, количество пользователей и т.д.)

Мониторинг Kubernetes

Мониторинг Kubernetes

Описание всех компонентов мониторинга, и как его использовать описаны в документе

“Kubernetes monitoring architecture”

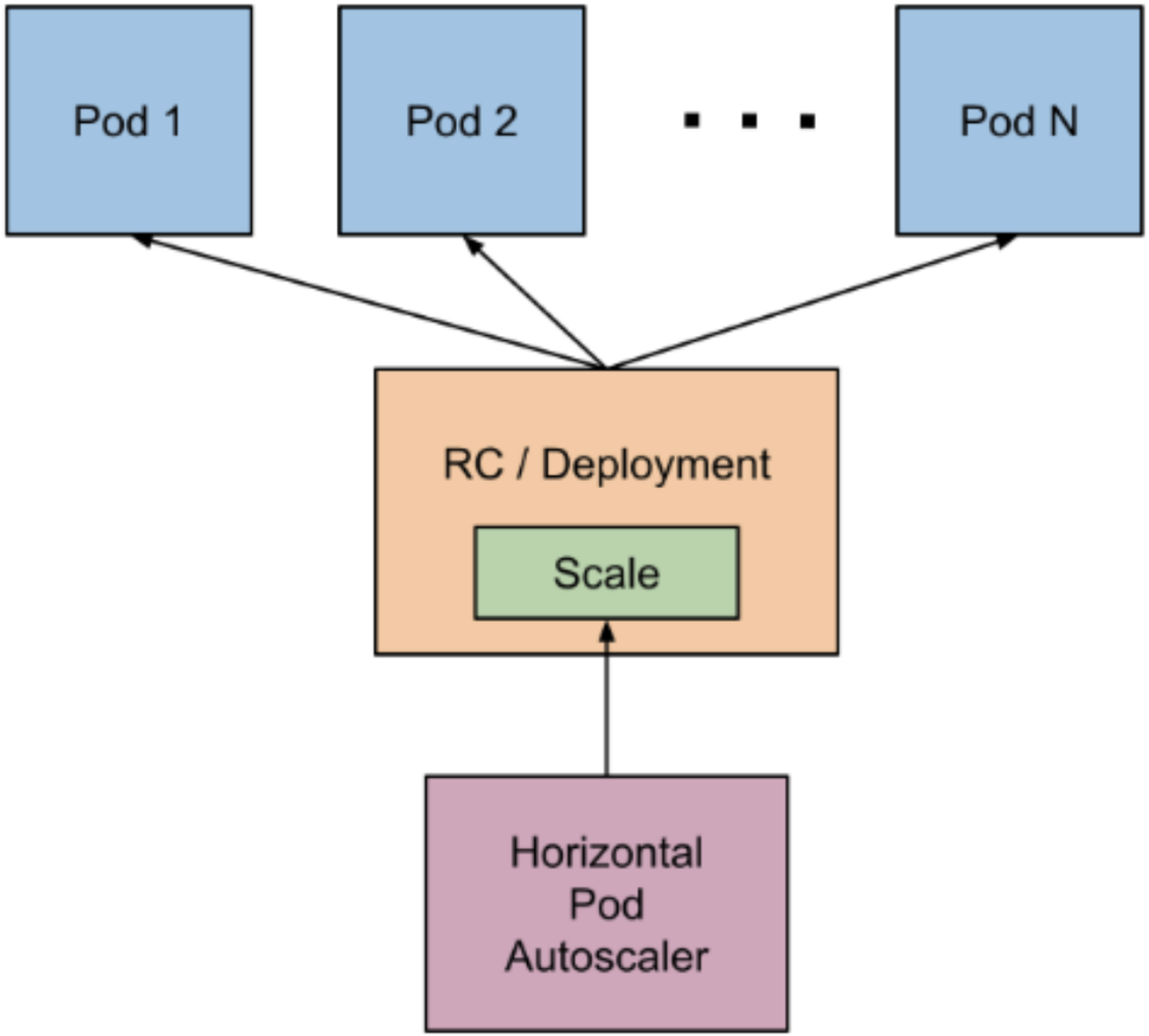
Мониторинг Kubernetes

- Pipelines
  - Core metrics pipeline
    - Kubelet
    - Resource estimator
    - Metrics-server
    - API server
  - Monitoring pipeline
    - Expose metrics to end-users

Мониторинг Kubernetes

- Metric-server предоставляет данные для HPA через адаптер
- Использование “Infrastore” в качестве хранилища данных
- Пользовательские системы мониторинга не взаимодействуют напрямую с metric-server
- cAdvisor должен быть источником метрик контейнеров
- Все компоненты k8s должны предоставлять (если должны) метрики в формате пригодном для Prometheus

Мониторинг Kubernetes



Мониторинг Kubernetes

Мониторинг нод кластера

- Core metrics
  - CPU
  - RAM
  - Disk
  - Network

Подходы к мониторингу

- The Four Golden Signals
- The USE Method
- The RED Method

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 USE Method

Подход озвучил Брендан Грэг

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

The RED Method

Подход озвучил Том Вилки

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

Rate

количество запросов в секунду

Errors

количество запросов завершившихся с ошибкой

Duration

количество времени которое занимает каждый запрос

Мониторинг | Контейнеры | CPU

```
1   sum(
2       rate(container_cpu_usage_seconds_total[5m]))
3   by (container_name)
```

Мониторинг | Контейнеры | CPU

- `container_cpu_cfs_throttled_seconds_total`

```
1   sum(
2       rate(container_cpu_cfs_throttled_seconds_total[5m]))
3   by (container_name)
```

Мониторинг времени когда наши контейнеры начинают “тротлить”

Мониторинг | Контейнеры | RAM

- `container_memory_working_set_bytes`

```
1   sum(container_memory_working_set_bytes{name!~"POD"})
2     by (name)
```

OOMkiller все видит ^_^

Мониторинг K8s API Server

- Request (Rates and Latencies)
- Performance of controller work queues
- Etcd helper cache work queues and cache performance
- General process status (File Descriptors/Memory/CPU Seconds)
- Golang status (GC/Memory/Threads)

Мониторинг K8s API Server

- Запросы: WATCH, PUT, POST, PATCH, LIST, GET, DELETE, and CONNECT

```
sum(rate(apiserver_request_count[5m])) by (resource, subresource, verb)
```

- Errors

```
rate(apiserver_request_count{code=~"^(?:5..)$"}[5m]) / rate(apiserver_request_count[5m])
```

- Duration

```
 histogram_quantile(0.9, sum(rate(apiserver_request_latencies_bucket[5m]))
 by (le, resource, subresource, verb) ) / 1e+06
```

Мониторинг ETCD

Доступно подробное описание всех метрик

```
- job_name: etcd
  static_configs:
    - targets: ['etcd-a.xxxx.xxxx:4001']
```

- Leader existence and leader change rate
- Proposals committed/applied/pending/failed
- Disk write performance
- Inbound gRPC stats
- Intra-cluster gRPC stats

Мониторинг Kube-state-metrics

- Значения для всех типов объектов
- Все метки и их значения связанные с каждым объектом
- Время создания для всех объектов
- Общая информация специфичная для каждого объекта
- Другие состояния специфичные для определенного объекта

```
count(kube_pod_status_phase{phase="Running"})
count(kube_pod_status_phase{phase="Failed"})
```

Мониторинг сервисов

Probes

Периодические проверки pod’а на жизнеспособность.

Как узнать, что сервис “жив” и готов к работе?

- **ExecAction** - выполнить команду и ждать exit code 0
- **TCPSocketAction** - проверить, что TCP-порт открыт
- **HTTPGetAction** - отправить HTTP GET-запрос

Мониторинг сервисов

Liveness probes

- Проверяет что приложение запущено и “живо”
- Если probe неуспешна - перезапуск контейнера*

```
1   apiVersion: v1
2   kind: Pod
3   ...
4   spec:
5     containers:
6     - name: liveness
7       image: liveness
8       livenessProbe:
9         tcpSocket:
10          port: 8080
11        initialDelaySeconds: 5
12        periodSeconds: 10
```

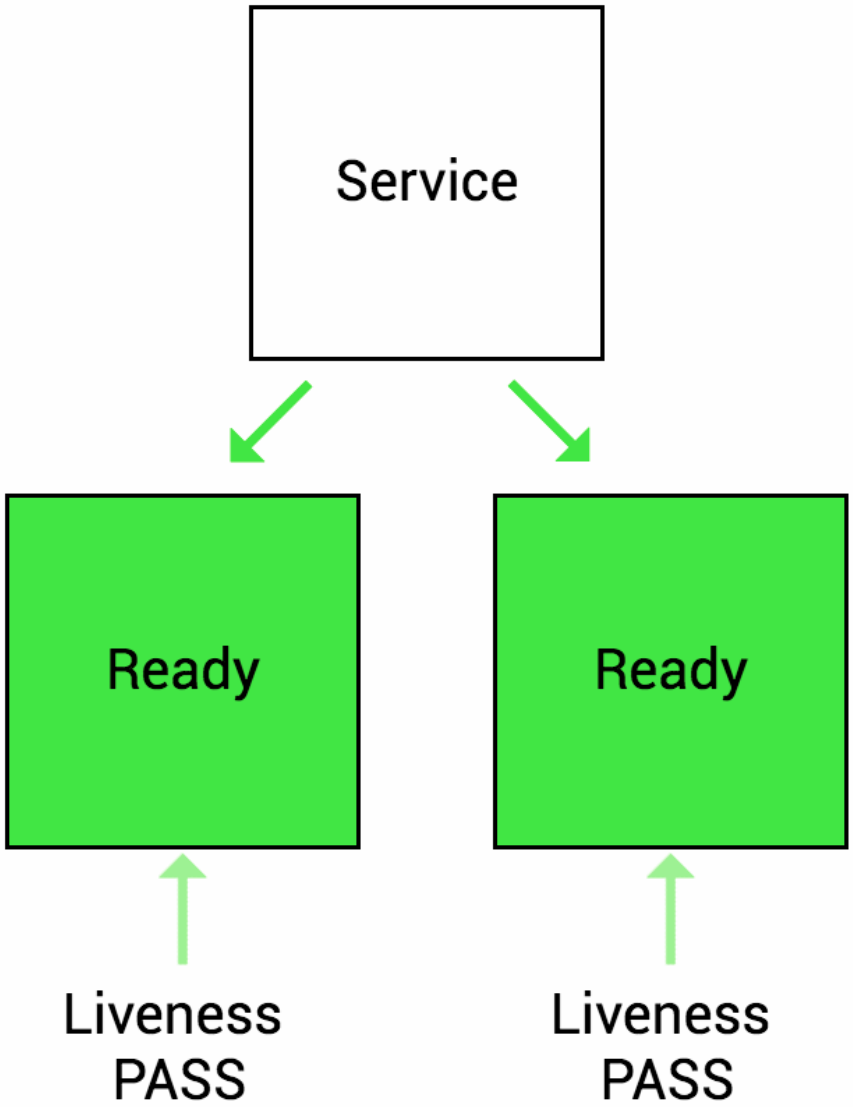
Мониторинг сервисов

Startup probes

- Проверяет что приложение готово обслуживать запросы
- После того как `startupProbe` успешна - управление передается `livenessProbe`

```yaml
livenessProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 1
  periodSeconds: 1

startupProbe:
  httpGet:
    path: /healthz
    port: liveness-port
  failureThreshold: 30
  periodSeconds: 10
```

Liveness Probe



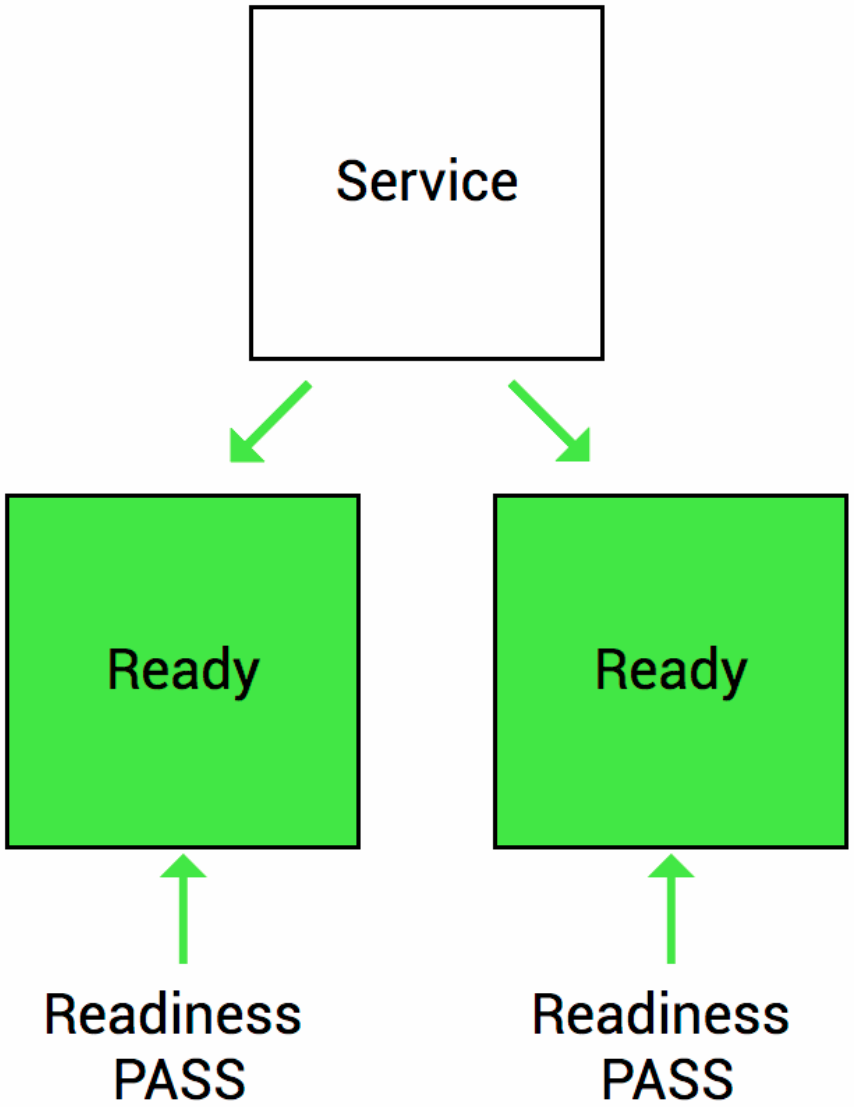
Мониторинг сервисов

Readiness probes

- Проверяет что приложение готово обслуживать запросы
- Если `readinessProbe` неуспешна - происходит удаление pod из балансировки

```
1   apiVersion: v1
2   kind: Pod
3   ...
4   spec:
5     containers:
6     - name: readiness
7       image: readiness
8       readinessProbe:
9         httpGet:
10          path: /healthz
11          port: 8080
12        initialDelaySeconds: 3
13        periodSeconds: 3
```

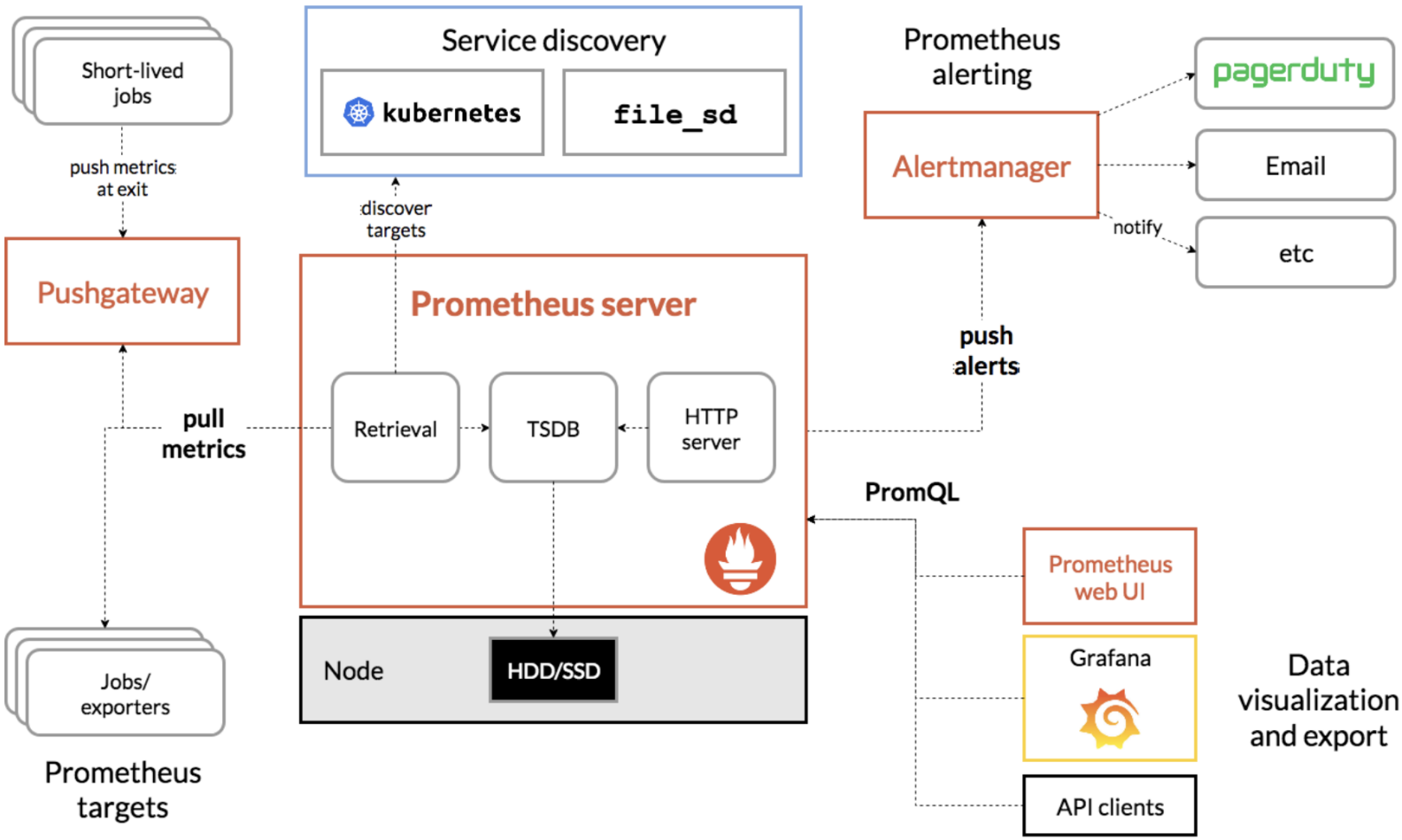
Redness Probe



Prometheus

- Компоненты
  - Pushgateway
  - Prometheus server
  - Alertmanagers
  - Web-UI

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

Виды федерации:

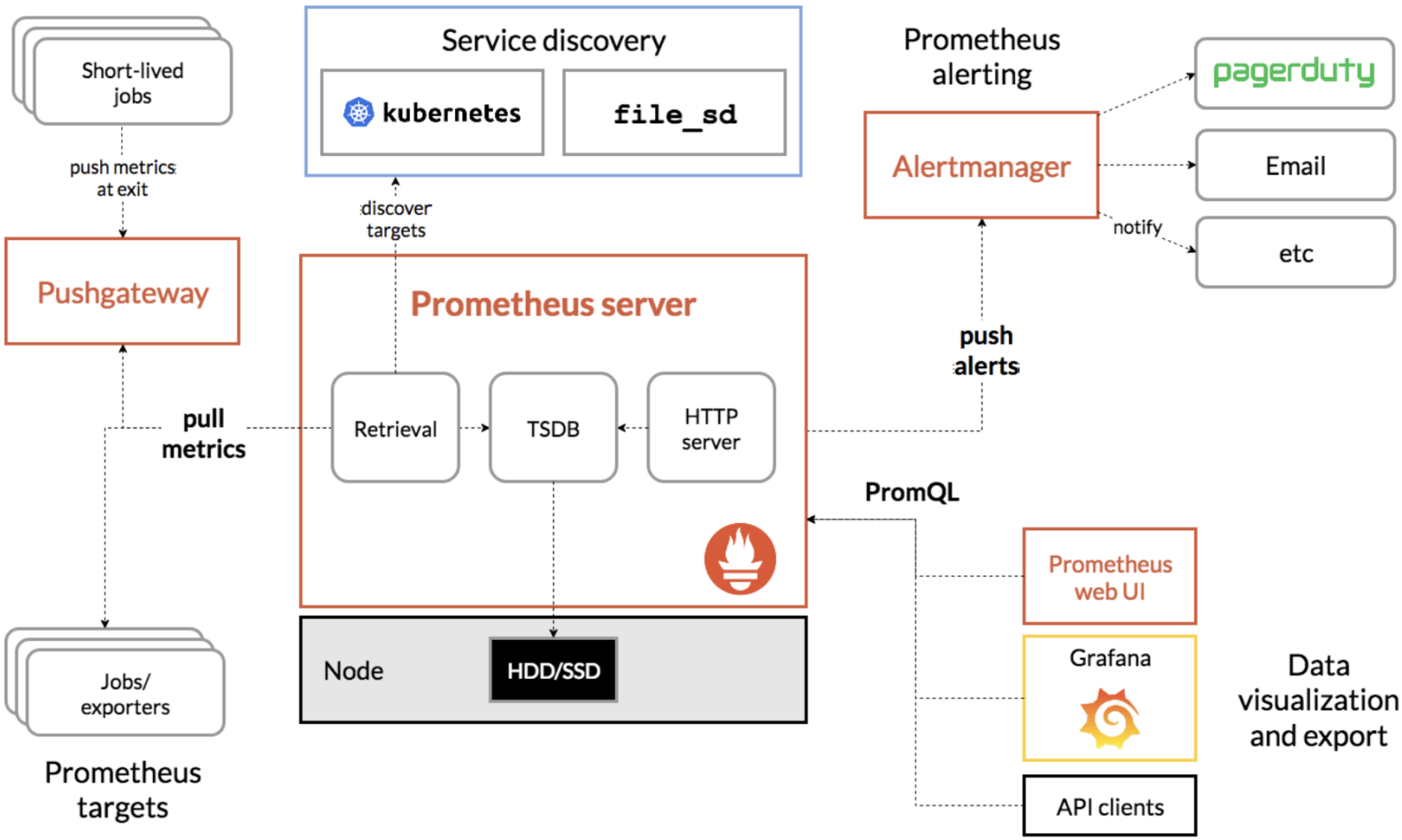
- Hierarchical federation
- Cross-service federation

Prometheus хранение метрик

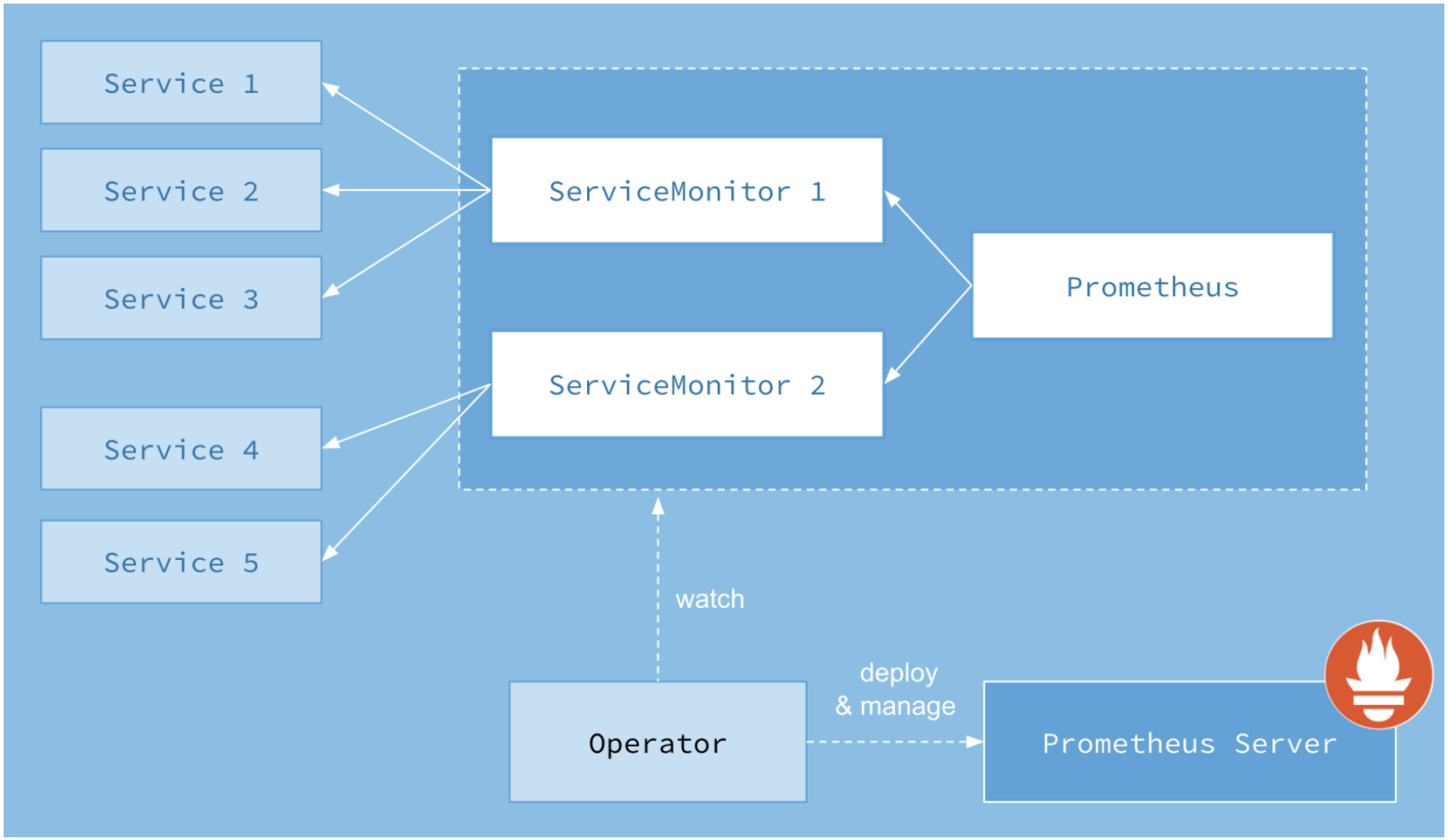
Отдельные хранилища для исторических данных:

- Victoria Metrics
- Cortex
- Thanos
- Clickhouse ^_^

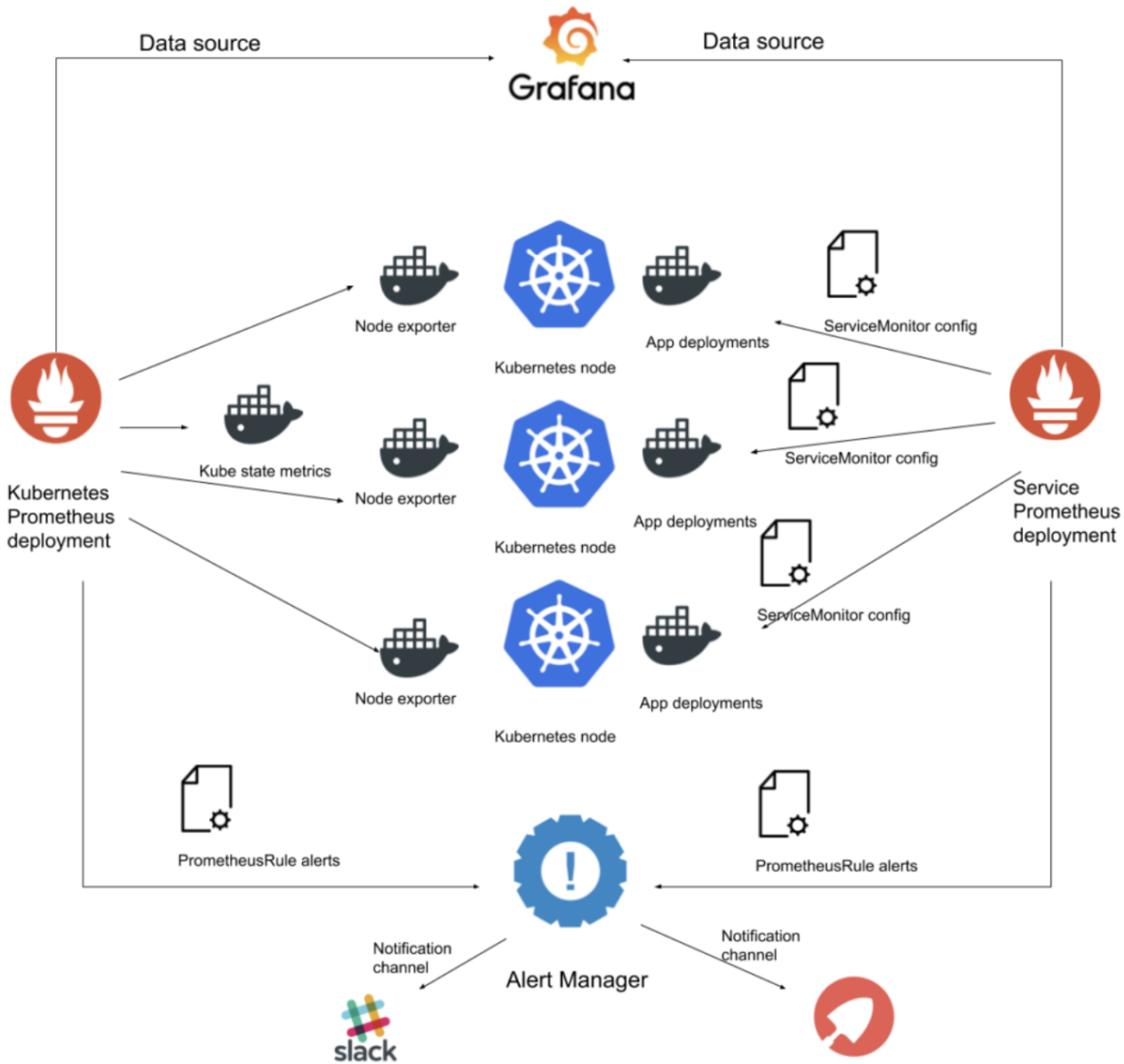
Prometheus operator



Prometheus operator



Prometheus operator | Service monitor



Prometheus operator

Варианты установки

- Helm chart
- Official repo
- Вкатить его руками ^_^

Prometheus operator values.yaml

```yaml
prometheus:
  prometheusSpec:
    retention: 3d
    ### Check this labels: kubectl get prometheus -o yaml -n monitoring
    serviceMonitorNamespaceSelector: {} ### Namespace for ServiceMonitors select
    serviceMonitorSelectorNilUsesHelmValues: false
    serviceMonitorSelector: {}   ### matchLabels for ServiceMonitors select
    # serviceMonitorSelector:
    #   matchLabels:
    #     prometheus: kube-prometheus
    #     release: prometheus-cluster-monitoring
grafana:
  adminPassword: XXXXXXXXXXXXXXXXXXX
```

Prometheus operator

Service for service monitor

```yaml
kind: Service
apiVersion: v1
metadata:
  name: example-app
  labels:
    app: example-app
spec:
  selector:
    app: example-app
  ports:
  - name: web
    port: 8080
```

> ! Обращаем внимание, что мы создаем **named** порт

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
```

Prometheus operator

Загрузка шаблонов

- При помощи configmap (sidecar container)
- Gerrit API
- Руками нуждающихся ^_^

Загрузка правил

- Через специальный объект PrometheusRule

Grafana

- Open source инструмент для построения дашбордов систем мониторинга
- Поддерживает получение данных из Graphite, Elasticsearch, OpenTSDB, Prometheus и InfluxDB и баз SQL
- Плагины для интеграции с другими системами мониторинга
- Hub с готовыми дошбордами

Дашборды

- Дашборды в Grafana хранятся в `.json`
- Есть возможность их импортировать/экспортировать
- Начиная с версии 4.0 поддерживается версионирование дашбордов при изменении с возможностью отката
- При желании, свои дашборды можно опубликовать в маркетплейсе https://grafana.com/dashboards

Hub для дашбордов

На сайте Grafana доступны сотни готовых дашбордов. Доступен поиск и фильтрация по различным критериям.

Promql

Язык запросов PromQL

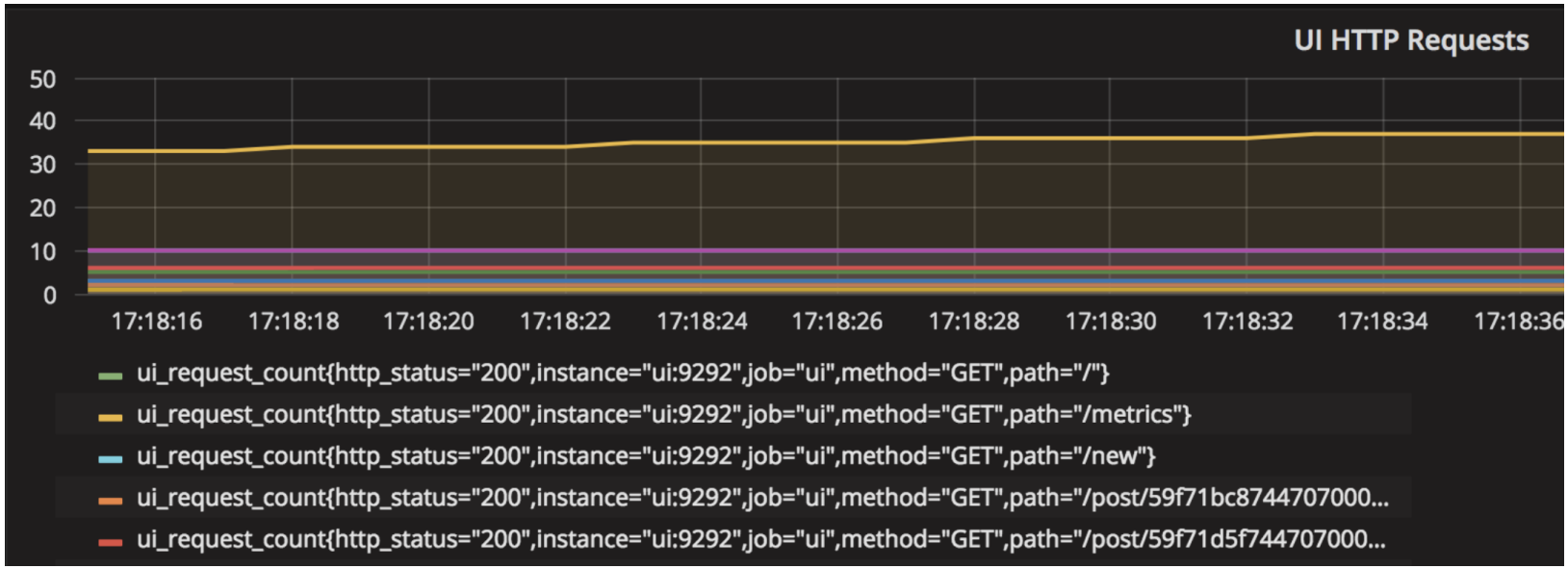
PromQL - язык запроса данных, реализованный в Prometheus. Запросы данных в PromQL состоят из:

- **Literals** (литералы) - числа, строки
- **Time series Selectors** - выборка вектора значений метрики из временного ряда за определенный момент или за промежуток времени. Например: `node_cpu`, `node_cpu{mode='idle'}`, `node_cpu{mode='idle'} offset 5m`, `node_cpu{mode='idle'}[1m]`
- **Operators** (операторы) - различные операторы: арифметические, сравнения, работы над векторами и агрегации
- **Functions** (функции) - большой список функций, которые можно применять к вектору временного ряда

Time series selector

Отобразим метрику:

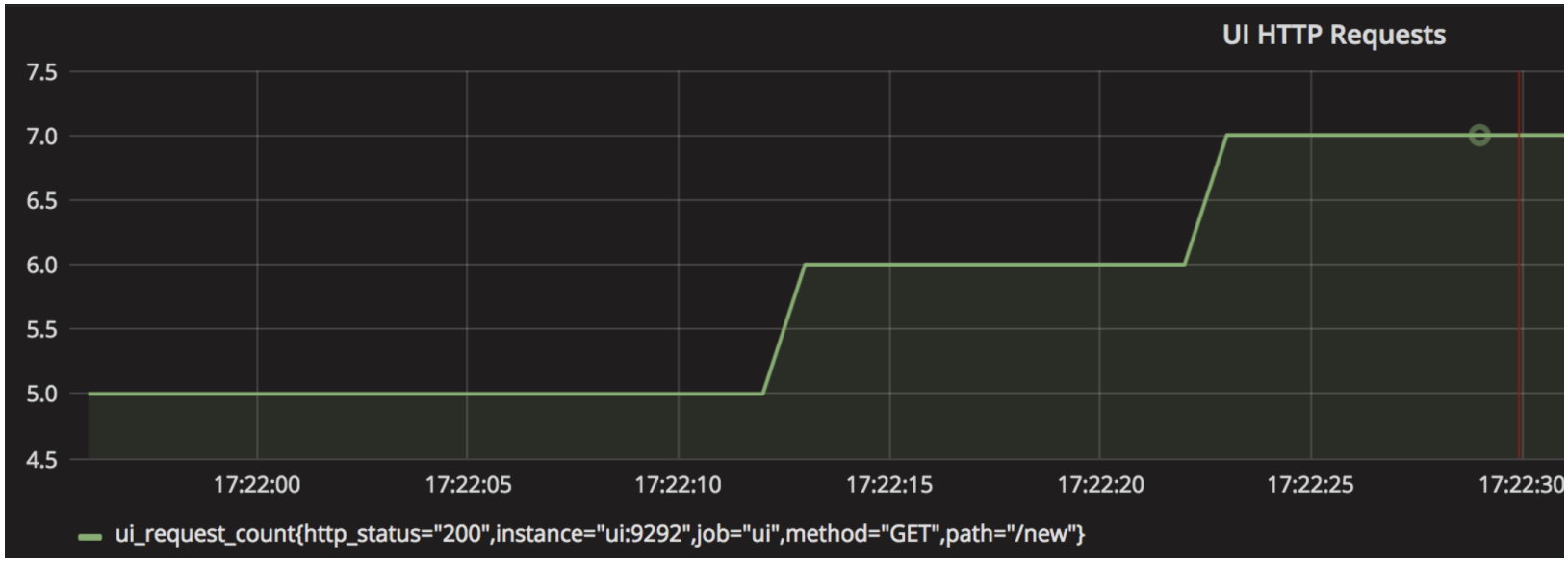
`ui_request_count`



Time series selector

Уточняем метрику, используя лейблы:

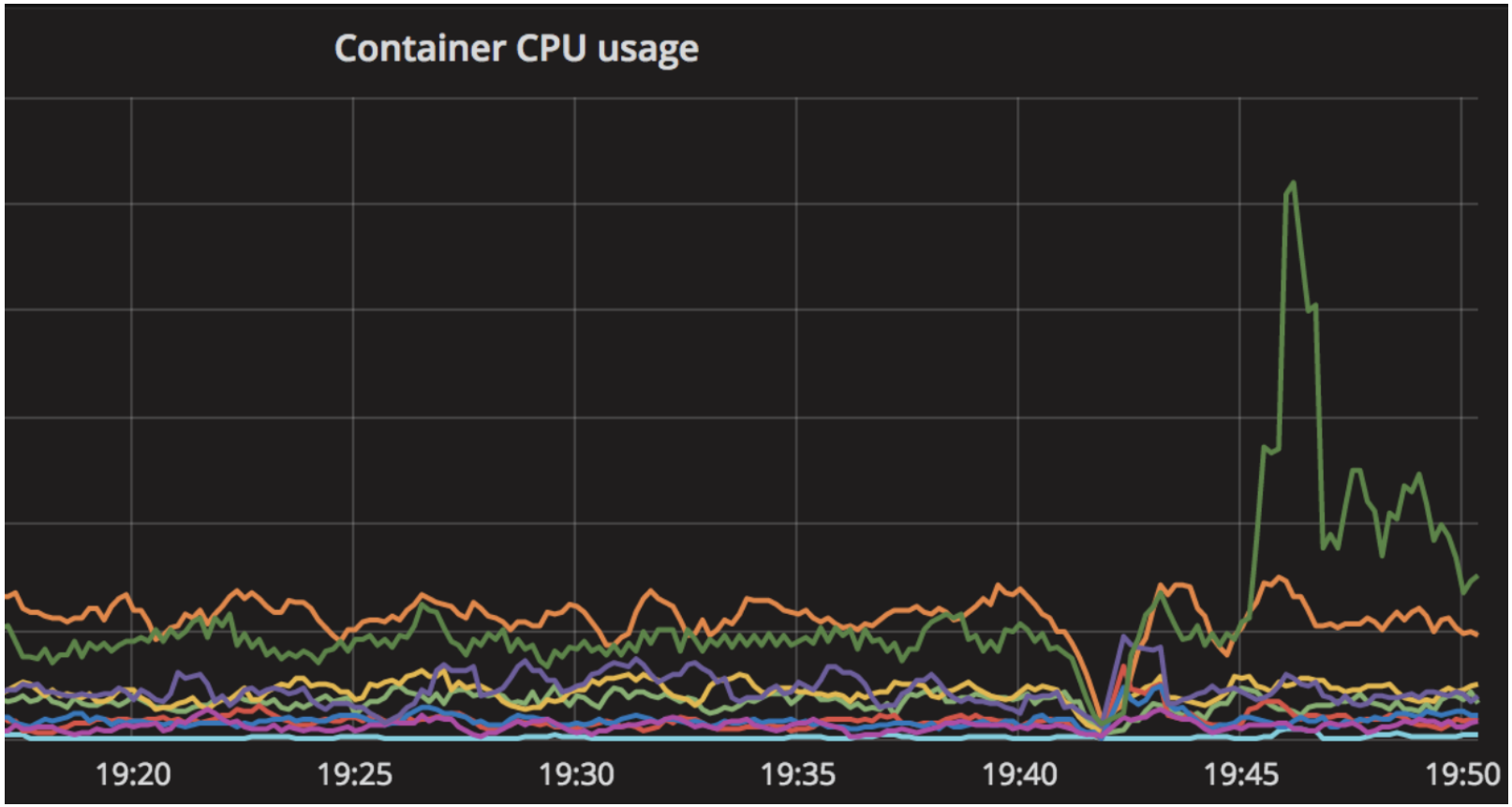
```
ui_request_count{method="GET",path="/new"}
```



Range vector selector

Результат запроса:

```
rate(container_cpu_user_seconds_total{image!=""}[1m])
```



Логирование

Логирование

Инструментарий

- ELK-stack\ELK-stack
- Loki

kubectl logs

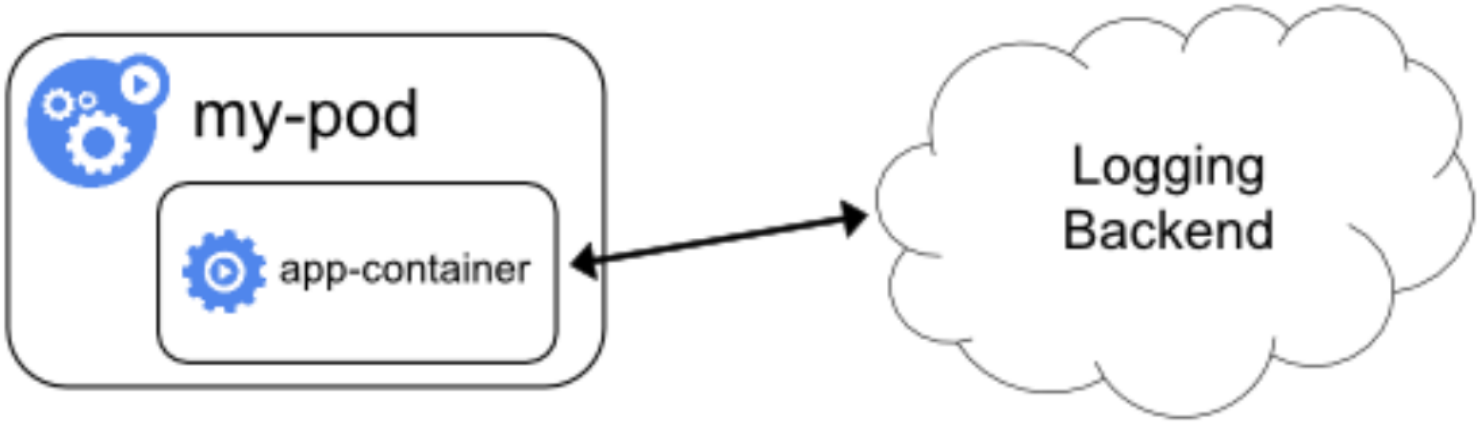
```
$ kubectl logs pod_name container_name --tail 10 -f

10.8.19.5 - - [04/Dec/2017 15:04:16] "GET /metrics HTTP/1.1" 200 -
10.8.19.5 - - [04/Dec/2017 15:05:16] "GET /metrics HTTP/1.1" 200 -
10.8.19.5 - - [04/Dec/2017 15:06:16] "GET /metrics HTTP/1.1" 200 -
10.8.19.5 - - [04/Dec/2017 15:07:16] "GET /metrics HTTP/1.1" 200 -
10.8.19.5 - - [04/Dec/2017 15:08:16] "GET /metrics HTTP/1.1" 200 -
10.8.19.5 - - [04/Dec/2017 15:09:16] "GET /metrics HTTP/1.1" 200 -
```

> У Kubernetes нет встроенных механизмов для отправки логов (таких как logging drivers в Docker)

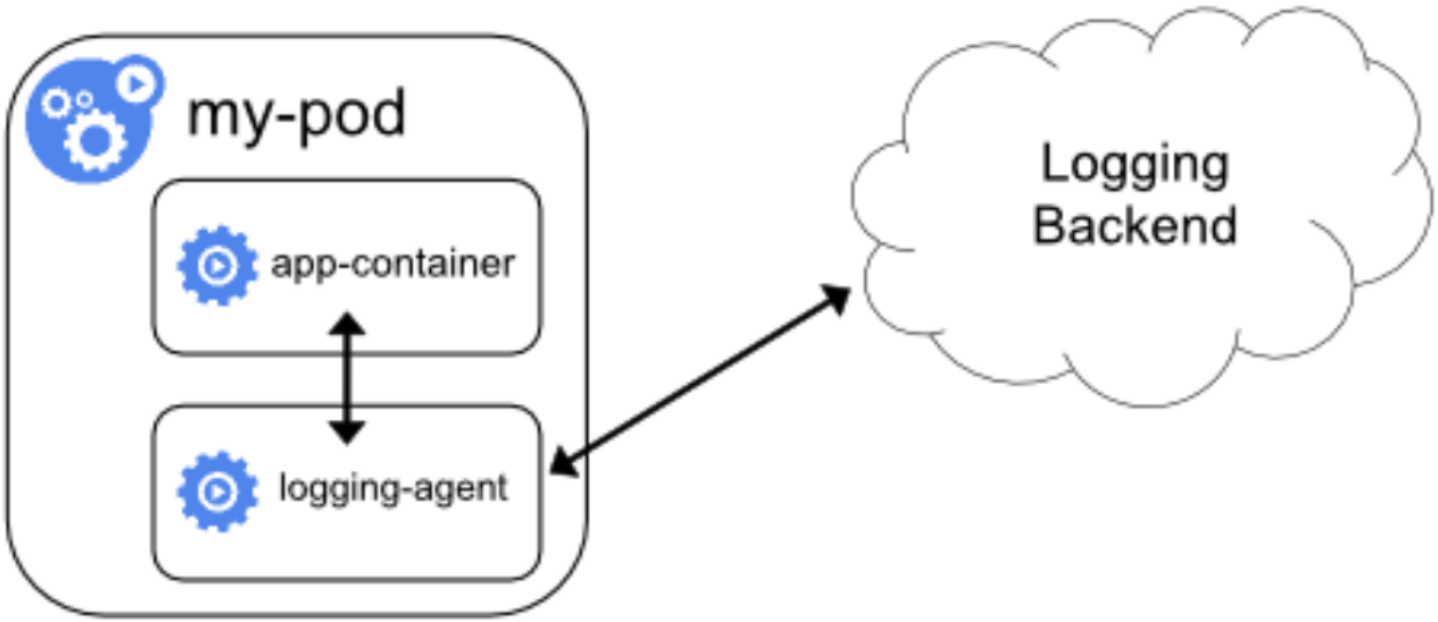
Логирование

- В Kubernetes мы не можем хранить логи локально
- Инстансы могут быть ограничены жизненным циклом
- Командное взаимодействие при отслеживании проблемы
- Ротация логов

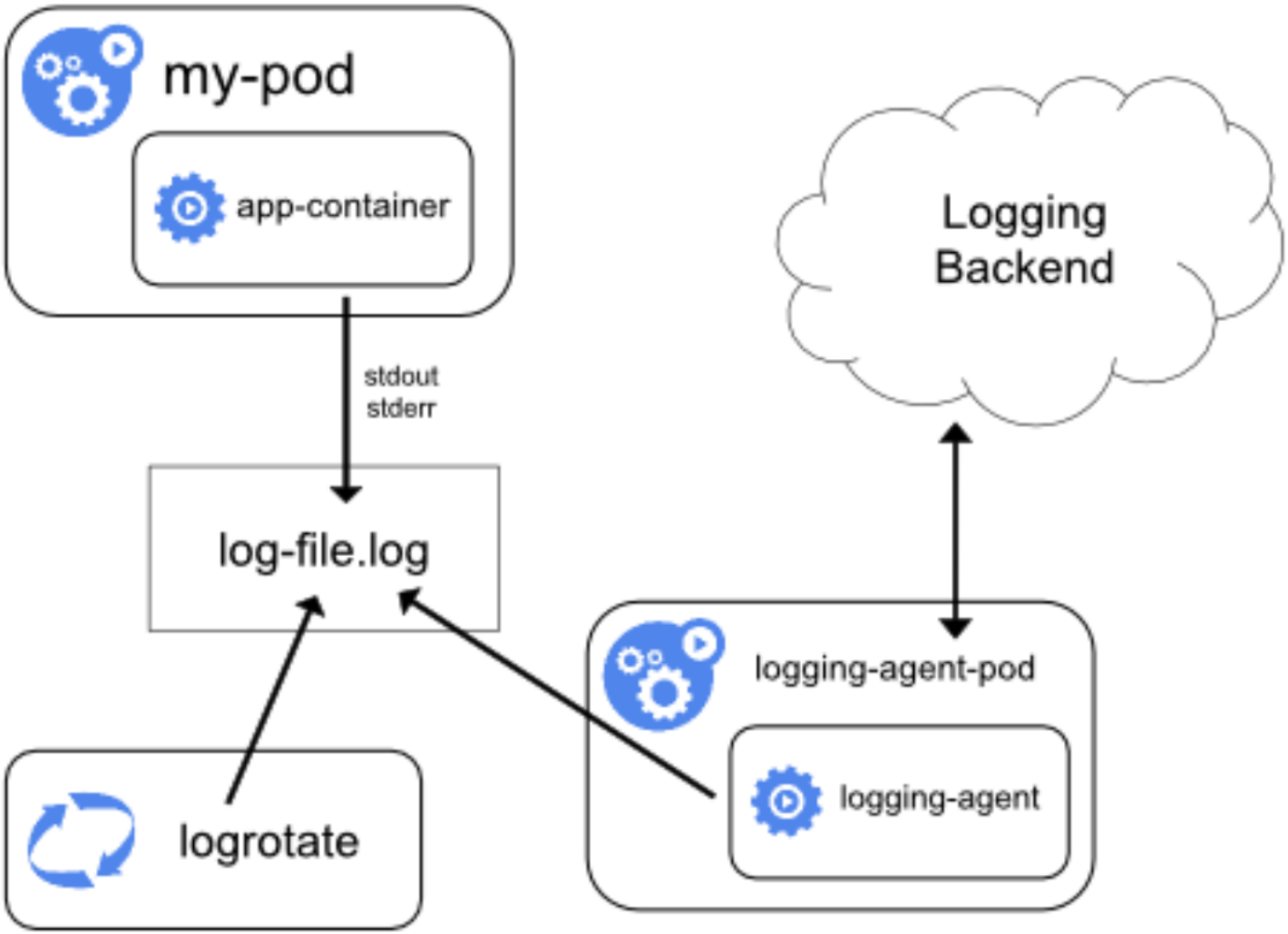
Логирование



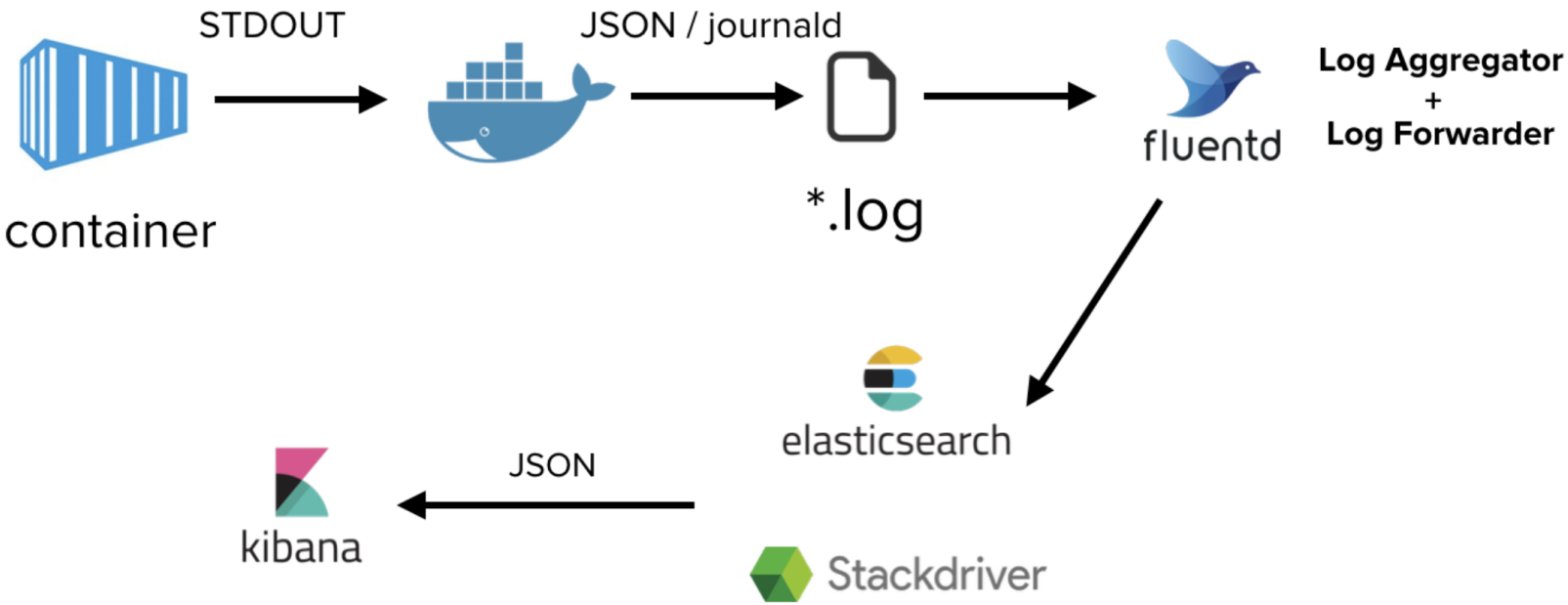
Логирование



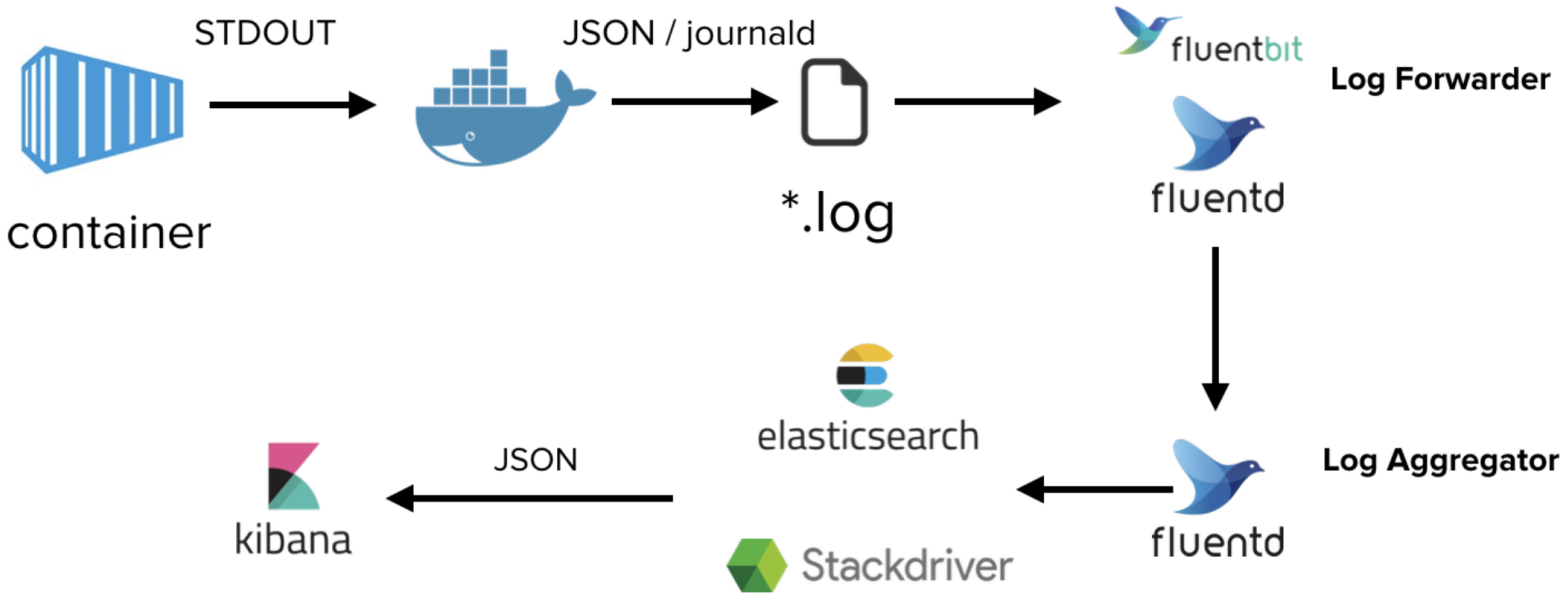
Логирование



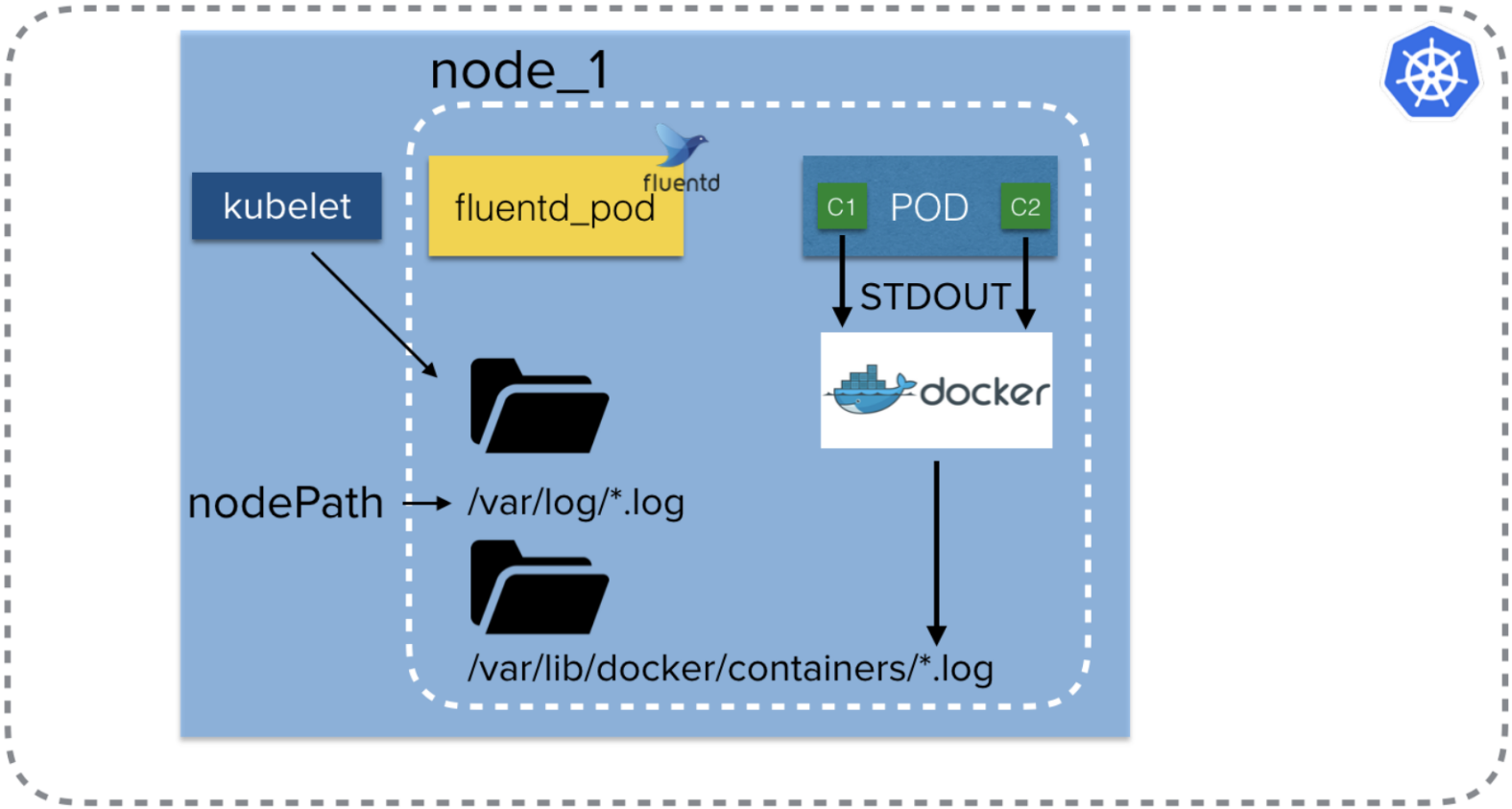
Логирование



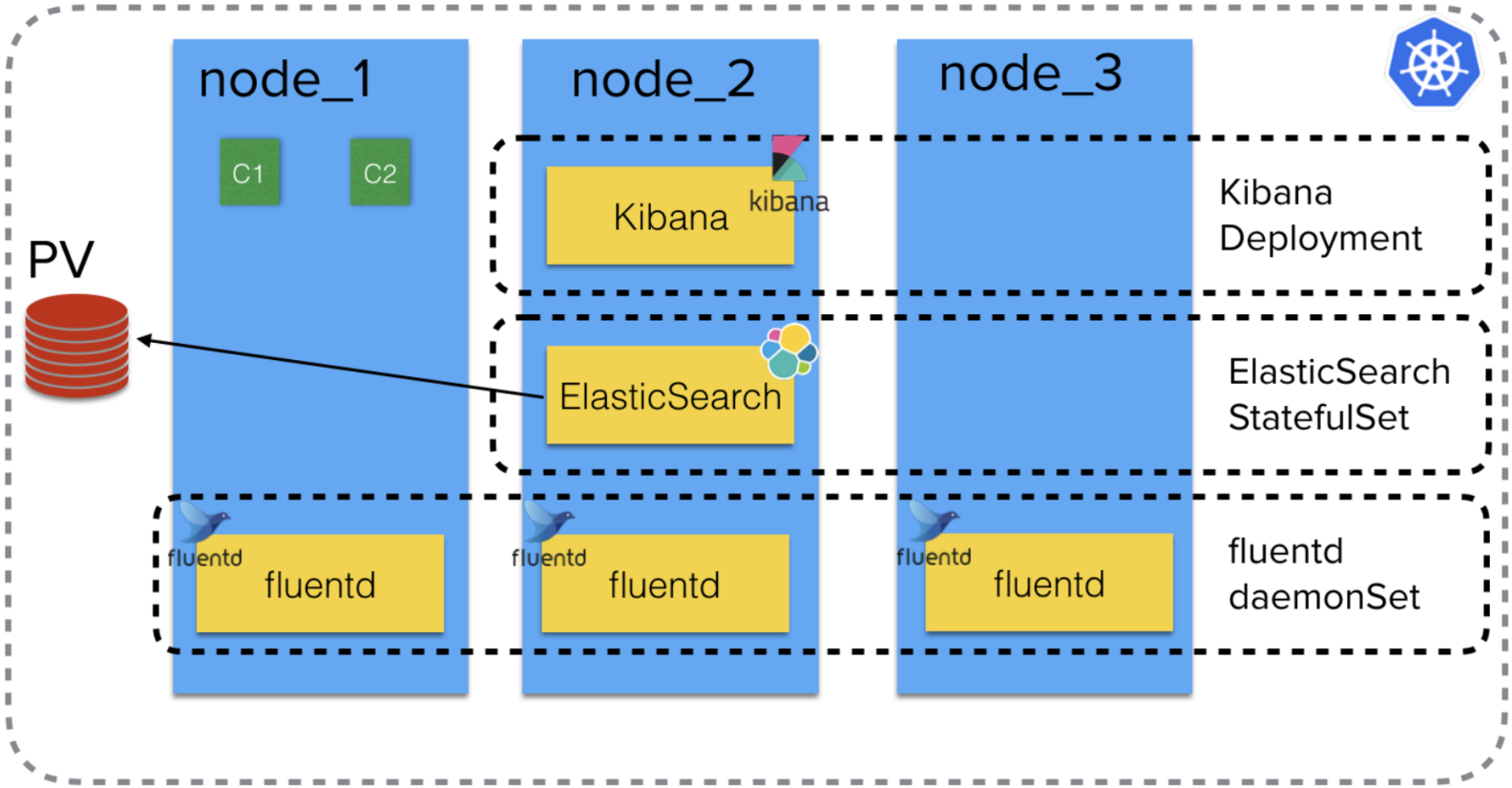
Логирование



Логирование



Логирование



Полезные ссылки

- Operating within Normal Parameters: Monitoring Kubernetes (SREcon 2019)
- Масштабирование приложения в Kubernetes на основе метрик из Prometheus
- Kube eagle
- Долгосрочное хранение метрик Prometheus
- 12 factor app

Полезные ссылки

- Книга Брендана Грэга System Performance: Enterprise and the Cloud
- Заметки про USE и RED
- Статья про SRE Golden Signals
- Статья от DataDog Monitoring 101: Собираем правильные данные
- Алексей Иванов, Dropbox. Практический опыт мониторинга распределенных систем Слайды
- Владимир Пычев, Google Как я научился не волноваться и полюбил пейджер (Про концепцию SRE, SLA/SLO/SLI и др.) Слайды

Полезные ссылки

- Владимир Иванов, Booking Grapgire в booking.com
- Конференции и др.: Uptime community
- Примеры Runbook от Gitlab