Документация Deckhouse Prom++

Дата генерации: 24.07.2026

Документация

Концепции

Модель данных

Deckhouse Prom++ хранит все данные в виде временных рядов: потоков значений с временными метками, принадлежащих одной метрике и одному набору лейблов. Помимо хранимых временных рядов, Deckhouse Prom++ может генерировать производные временные ряды в результате выполнения запросов.

Имена метрик и лейблы

Каждый временной ряд однозначно идентифицируется своим именем метрики и необязательными парами «ключ-значение», называемыми лейблами.

Имена метрик

- **Описание**: Указывают общую характеристику системы, которая измеряется (например, `http_requests_total` — общее количество полученных HTTP-запросов).
- **Правила именования**:
  - Имена метрик могут содержать буквы ASCII, цифры, подчеркивания и двоеточия.
  - Должны соответствовать регулярному выражению: `[a-zA-Z_:][a-zA-Z0-9_:]*`.

> **Внимание.** Двоеточия зарезервированы для пользовательских правил записи. Они не должны использоваться в экспортерах или прямом инструментировании.

Лейблы

- **Описание**: Позволяют модели данных Deckhouse Prom++ идентифицировать любую комбинацию лейблов для одной метрики. Они определяют конкретную размерную реализацию этой метрики (например, все HTTP-запросы, которые использовали метод `POST` для обработчика `/api/tracks`). Язык запросов позволяет фильтровать и агрегировать данные на основе этих измерений.
- **Правила именования**:
  - Лейблы могут содержать буквы ASCII, цифры и подчеркивания. Они должны соответствовать регулярному выражению: `[a-zA-Z_][a-zA-Z0-9_]*`.
  - Имена лейблов, начинающиеся с `__` (двух подчеркиваний), зарезервированы для внутреннего использования.
  - Значения лейблов могут содержать любые символы Unicode.
  - Лейблы с пустым значением считаются эквивалентными отсутствию лейбла.
- **Изменение лейблов**: Любое изменение значения лейбла, включая добавление или удаление лейблов, создает новый временной ряд.

Samples

Samples (выборки) формируют фактические данные временных рядов. Каждая выборка включает:

- **значения** — число с плавающей точкой (`float64`);
- **временные метки (timestamps)** — точность до миллисекунды.

> Начиная с версии Deckhouse Prom++ v2.40, добавлена экспериментальная поддержка нативных гистограмм. Вместо простого `float64` значение выборки может теперь представлять собой полную гистограмму.

Формат представления метрик

Для идентификации временных рядов используется следующий формат:

```
<metric name>{<label name>=<label value>, ...}
```

Например, временной ряд с именем метрики `api_http_requests_total` и лейблами `method="POST"` и `handler="/messages"` может быть записан следующим образом:

```
api_http_requests_total{method="POST", handler="/messages"}
```

Служебные лейблы и метрики

В терминах Deckhouse Prom++, эндпоинт, с которого можно собирать метрики, называется экземпляром (instance) — как правило, это один рабочий процесс.
Набор экземпляров с одинаковым назначением — например, реплики одного и того же процесса, используемые для масштабирования или обеспечения отказоустойчивости — называется задачей (job).

Пример задачи сервера API с четырьмя репликами-экземплярами:

```
- job: `api-server`
  - instance 1: `1.2.3.4:5670`
  - instance 2: `1.2.3.4:5671`
  - instance 3: `5.6.7.8:5670`
  - instance 4: `5.6.7.8:5671`
```

Автоматически создаваемые лейблы и временные ряды

Когда Deckhouse Prom++ опрашивает целевой эндпоинт (target), он автоматически добавляет к собранным временным рядам несколько служебных лейблов, которые помогают идентифицировать источник данных:

- `job` — имя задачи, которой принадлежит целевая система согласно настройке;
- `instance` — часть URL-адреса целевого сервиса в формате `<host>:<port>`, с которого были собраны данные.

Если любой из этих лейблов (`job` или `instance`) уже присутствует в исходных данных, поведение определяется настройкой `honor_labels`. Дополнительные подробности можно найти в разделе настройки сбора данных (scraping).

При каждом сборе данных (scraping) экземпляра Deckhouse Prom++ создает несколько дополнительных временных рядов:

- `up{job="<job-name>", instance="<instance-id>"}` — равен `1`, если экземпляр доступен (scrape прошёл успешно), и `0`, если опрос не удался;
- `scrape_duration_seconds{job="<job-name>", instance="<instance-id>"}` — продолжительность опроса в секундах;
- `scrape_samples_post_metric_relabeling{job="<job-name>", instance="<instance-id>"}` — количество метрик после применения правил переименования метрик (metric relabeling);
- `scrape_samples_scraped{job="<job-name>", instance="<instance-id>"}` — общее количество метрик, полученных от целевой системы за опрос;
- `scrape_series_added{job="<job-name>", instance="<instance-id>"}` — приблизительное количество новых временных рядов, добавленных за опрос.

Временной ряд `up` широко используется для мониторинга доступности экземпляров.

Если активирован флаг `extra-scrape-metrics`, дополнительно доступны следующие метрики:

- `scrape_timeout_seconds{job="<job-name>", instance="<instance-id>"}` — настроенное значение `scrape_timeout` для целевой системы;
- `scrape_sample_limit{job="<job-name>", instance="<instance-id>"}` — настроенный лимит числа метрик (`sample_limit`) для опроса. Если лимит не задан, будет возвращено значение `0` ;
- `scrape_body_size_bytes{job="<job-name>", instance="<instance-id>"}` : Размер ответа последнего опроса (в байтах), если опрос прошёл успешно. При ошибке из-за превышения лимита `body_size_limit` будет возвращено `-1`, в остальных случаях неудачного опроса — `0`.

Типы метрик

Counter

Counter (счетчик) — это кумулятивная метрика, представляющая собой один равномерно возрастающий счетчик, значение которого может только увеличиваться или обнуляться при перезапуске. Например, вы можете использовать счетчик для представления количества обслуженных запросов, выполненных задач или ошибок.

Не используйте счетчик для представления значения, которое может уменьшаться. Например, не используйте счетчик для количества текущих запущенных процессов — вместо него используйте индикатор.

Gauge

Gauge (индикатор) — это метрика, представляющая собой одно числовое значение, которое может произвольно увеличиваться или уменьшаться. Обычно индикаторы используются для измерения таких величин, как температура или текущее использование памяти, а также для «подсчетов», которые могут увеличиваться и уменьшаться, например, количество одновременных запросов.

Histogram

Histogram (гистограмма) регистрирует отдельные измерения (например, длительность запросов или размер ответов) и распределяет их по настраиваемым диапазонам — бакетам. Она также хранит сумму всех зафиксированных значений.

Гистограмма с базовым именем метрики `<basename>` при сборе данных экспонирует несколько временных рядов:

- накопительные счётчики для каждого бакета в формате `<basename>_bucket{le="<верхняя граница включительно>"}`;
- суммарное значение всех зафиксированных измерений — `<basename>_sum`;
- общее количество зарегистрированных событий — `<basename>_count` (эквивалентно значению `<basename>_bucket{le="+Inf"}`, включающему все измерения).

Для расчёта процентилей на основе гистограмм можно использовать функцию `histogram_quantile()`.

Гистограммы также подходят для вычисления Apdex-индекса. При работе с бакетами помните, что значения гистограммы накапливаются (являются кумулятивными).

Summary

Аналогично гистограмме, summary (сводка) регистрирует отдельные измерения (например, длительность запросов или размер ответов).

Метрика сохраняет общее количество событий, сумму всех зафиксированных значений, а также рассчитывает настраиваемые процентильные значения в рамках «скользящего» временного окна.

Сводка с базовым именем метрики `<basename>` при сборе данных экспонирует несколько временных рядов:

- процентильные значения φ, вычисляемые на лету, в формате `<basename>{quantile="<φ>"}`;
- сумма всех зафиксированных измерений — `<basename>_sum`;
- общее число зарегистрированных событий — `<basename>_count`.

Сервер Deckhouse Prom++

Установка

Конвертация WAL перед установкой

Deckhouse Prom++ использует альтернативный формат WAL (Write-Ahead Log), но полностью совместим с историческими блоками.
Поскольку WAL содержит **последние 1,5 блока данных** (обычно около **3 часов**), если вы планируете использовать Deckhouse Prom++ в качестве замены Prometheus, необходимо переконвертировать WAL для предотвращения потери данных.

Обратитесь к руководству по миграции для подробных шагов по конвертации.

Предварительно скомпилированные бинарные файлы

1. Скачайте последний бинарный файл с учетом необходимой архитектуры:
    - amd64;
    - arm64.
2. Распакуйте его:

    ```
    tar zxf prompp-binaries-<amd64|arm64>.tar.gz
    ```

    Будет создана папка `prompp` с бинарным файлом `prompp` и конфигурационным файлом `prometheus.yml`.
3. Запустите его как замену Prometheus:

    ```
    cd prompp
    ./prompp --config.file=prometheus.yml --storage.tsdb.path=data/
    ```

Docker

Deckhouse Prom++ доступен в виде Docker-образа в следующих хранилищах образов контейнеров:

- `registry.deckhouse.ru/prompp/prompp`;
- `ghcr.io/deckhouse/prompp`.

Все доступные версии можно найти на странице релизов.

Чтобы быстро запустить контейнер, выполните следующую команду:

```
docker run --name prompp -d -p 127.0.0.1:9090:9090 registry.deckhouse.ru/prompp/
prompp:0.7.4
```

Либо используйте GitHub Container Registry:

```
docker run --name prompp -d -p 127.0.0.1:9090:9090 ghcr.io/deckhouse/prompp:0.7.4
```

После запуска Deckhouse Prom++ будет доступен по адресу http://localhost:9090/.

Также вы можете добавить собственную конфигурацию Prom++, передав параметр `--config.file`:

```
docker run --name prompp -v /path/on/host/prometheus.yml:/etc/prometheus.yml -d -p
127.0.0.1:9090:9090 registry.deckhouse.ru/prompp/prompp:0.7.4 --config.file=/etc/
prometheus.yml
```

Prometheus Operator

1. Создайте файл `prompp.yaml` со следующей конфигурацией (другие настройки могут зависеть от вашей установки):

   ```
   apiVersion: monitoring.coreos.com/v1
   kind: Prometheus
   metadata:
     name: example-prometheus
     namespace: monitoring
   spec:
     image: registry.deckhouse.ru/prompp/prompp:0.7.4  # Замените Prometheus на
   Deckhouse Prom++.
     securityContext:
       fsGroup: 64535
       runAsGroup: 64535
       runAsNonRoot: true
       runAsUser: 64535
     # Дополнительные параметры могут зависеть от вашей установки.
   ```

2. Примените обновленный ресурс:

   ```
   kubectl apply -f prompp.yaml
   ```

Миграция из Prometheus

Ручная конвертация WAL

Если миграция происходит вручную, используйте утилиту `prompptool`, включенную в релиз.

Конвертация WAL Prometheus в формат Deckhouse Prom++

Чтобы сконвертировать WAL Prometheus в формат Deckhouse Prom++, выполните следующую команду:

```
prompptool walvanilla --working-dir <path to prometheus data dir>
```

Конвертация WAL Deckhouse Prom++ обратно в формат Prometheus

Чтобы сконвертировать WAL Deckhouse Prom++ в формат Prometheus, выполните следующую команду:

```
prompptool walpp --working-dir <path to prometheus data dir>
```

Автоматическая конвертация WAL с помощью Prometheus Operator

Конвертация WAL Prometheus в формат Deckhouse Prom++

1. Создайте файл `prompp-migration.yaml` со следующей конфигурацией (дополнительные параметры могут зависеть от вашей установки):

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: example-prometheus
  namespace: monitoring
spec:
  ...
  image: registry.deckhouse.ru/prompp/prompp:0.7.4
  securityContext:
    fsGroup: 64535
    runAsGroup: 64535
    runAsNonRoot: true
    runAsUser: 64535
  initContainers:
    - name: prompptool
      image: prompp/prompp:<tag>
      command:
        - /bin/prompptool
        - "--working-dir=/prometheus"
        - "walvanilla"
      volumeMounts:
        - name: prometheus-main-db
          mountPath: /prometheus
          subPath: prometheus-db
      resources:
        requests:
          cpu: "100m"
          memory: "128Mi"
  # Дополнительные параметры могут зависеть от вашей установки.
```

2. Примените обновленный ресурс:

```
kubectl apply -f prompp-migration.yaml
```

Конвертация WAL Deckhouse Prom++ обратно в формат Prometheus

1. Измените `initContainer` в вашем файле `prompp-migration.yaml`:

```
command:
  - /bin/prompptool
  - "--working-dir=/prometheus"
  - "--verbose"
  - "walpp"
```

2. Примените изменения снова:

```
kubectl apply -f prompp-migration.yaml
```

Начало работы

Это руководство в стиле «Hello, World!», которое демонстрирует, как установить, настроить и использовать простой экземпляр Deckhouse Prom++. Вы загрузите и запустите Deckhouse Prom++ локально, настроите его для сбора собственных метрик и метрик тестового приложения, а затем поработаете с запросами, правилами и графиками для работы с собранными временными рядами.

Загрузка и запуск Deckhouse Prom++

Установите последнюю версию Deckhouse Prom++ для вашей платформы.

Затем настройте Deckhouse Prom++ перед запуском.

Настройка Deckhouse Prom++ для мониторинга своего состояния

Deckhouse Prom++ собирает метрики с целевых объектов, поочередно опрашивая их HTTP-эндпоинты с метриками. Поскольку Deckhouse Prom++ предоставляет данные о своем состоянии аналогичным образом, он может также отслеживать и свое собственное состояние. Хотя сервер Deckhouse Prom++, который собирает данные только о себе, не несет большой пользы, он может послужить хорошим стартовым примером.

Сохраните следующую базовую конфигурацию в файле под именем `prometheus.yml`:

```
global:
  scrape_interval: 15s
# По умолчанию опрашивает целевые объекты каждые 15 секунд. Эти лейблы будут
прикрепляться ко всем временным рядам и алертам при взаимодействии с внешними
системами (федерация, удаленное хранилище, Alertmanager).
  external_labels:
    monitor: 'codelab-monitor'
```

Конфигурация опроса с одним эндпоинтом

В данном случае это сам Deckhouse Prom++.

```
scrape_configs:
  # Название задачи добавляется в лейбл `job=<job_name>` для всех временных рядов из
данной задачи.
  - job_name: 'promplusplus'

  # Переопределение глобального интервала опроса — для этой задачи он составит 5
секунд.
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:9090']
```

Запуск Deckhouse Prom++

Чтобы запустить Deckhouse Prom++ с новым конфигурационным файлом, перейдите в каталог с бинарным файлом Deckhouse Prom++ и выполните:

```
# По умолчанию база данных Prom++ хранится в ./data (опция --storage.tsdb.path).
./prompp --config.file=prometheus.yml
```

Deckhouse Prom++ должен успешно запуститься.

Откройте браузер и перейдите на страницу статуса Deckhouse Prom++ по адресу http://localhost:9090. Немного подождите, чтобы Prom++ мог собрать данные о своем состоянии через собственный HTTP-эндпоинт для метрик.

Чтобы убедиться в том, что Deckhouse Prom++ публикует собственные метрики, перейдите по адресу http://localhost:9090/metrics.

Использование браузера выражений

Чтобы изучить данные, которые Deckhouse Prom++ собрал о своем состоянии, перейдите во встроенный браузер выражений по адресу http://localhost:9090/graph → вкладка «Graph» → вид «Table».

Как можно увидеть на странице http://localhost:9090/metrics, одной из собственных метрик, которые экспортирует Deckhouse Prom++, является `prometheus_target_interval_length_seconds` (фактический интервал между опросами целевых объектов). Введите следующее выражение в консоли запросов и нажмите «Execute»:

```
prometheus_target_interval_length_seconds
```

Это вернет несколько временных рядов (с последними сохраненными значениями), в каждом из которых будет указана метрика `prometheus_target_interval_length_seconds`, но с разными лейблами. Лейблы обозначают различные процентильные задержки и интервалы группировки целевых объектов.

Если вас интересуют только задержки 99-го процентиля, используйте следующий запрос:

```
prometheus_target_interval_length_seconds{quantile="0.99"}
```

Чтобы посчитать количество возвращаемых временных рядов, используйте следующий запрос:

```
count(prometheus_target_interval_length_seconds)
```

Использование интерфейса графиков

Чтобы построить график по выражениям, перейдите по адресу http://localhost:9090/graph → вкладка «Graph».

Например, чтобы построить график скорости создания новых чанков в самосканируемом экземпляре Deckhouse Prom++, используйте следующий запрос:

```
rate(prometheus_tsdb_head_chunks_created_total[1m])
```

Попробуйте поэкспериментировать с параметрами диапазона и другими настройками.

Перезагрузка конфигурации без перезапуска

Deckhouse Prom++ позволяет применять новую конфигурацию без перезапуска, отправляя процессу сигнал `SIGHUP`:

```
kill -s SIGHUP <PID>
```

Корректное завершение Deckhouse Prom++

Для корректного завершения работы Deckhouse Prom++ используйте сигнал `SIGTERM`:

```
kill -s SIGTERM <PID>
```

Конфигурация

Конфигурация

Deckhouse Prom++ настраивается с помощью параметров командной строки и конфигурационного файла. Параметры командной строки задают неизменяемые системные характеристики (такие как расположение хранилища, объем данных, хранимых на диске и в памяти, и т.д.), а конфигурационный файл описывает все, что связано с опросом jobs и их экземпляров, а также определяет, какие файлы правил необходимо загрузить.

Чтобы посмотреть список всех доступных флагов, выполните `./prompp -h`.

Deckhouse Prom++ может перезагружать свою конфигурацию на лету. Если новая конфигурация некорректно сформирована, изменения не будут применены. Перезагрузка конфигурации запускается отправкой сигнала `SIGHUP` процессу Deckhouse Prom++ или отправкой HTTP POST запроса на эндпоинт `/-/reload` (когда включен флаг `--web.enable-lifecycle`). Это также перезагрузит все настроенные файлы правил.

Файл конфигурации

Чтобы явно указать, какой файл конфигурации загрузить, используйте флаг `--config.file`.

Файл пишется в формате YAML по схеме, описанной ниже. Квадратные скобки обозначают необязательные параметры. Для параметров-скаляров при пропуске берётся значение по умолчанию.

Обобщённые плейсхолдеры определены следующим образом:

- `<boolean>`: логическое значение `true` или `false`
- `<duration>`: продолжительность, соответствующая регулярному выражению `((([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?|0)`, например `1d`, `1h30m`, `5m`, `10s`
- `<filename>`: корректный путь в текущей рабочей директории
- `<float>`: число с плавающей запятой
- `<host>`: корректная строка, состоящая из имени хоста или IP-адреса с опциональным номером порта
- `<int>`: целое число
- `<labelname>`: строка, соответствующая регулярному выражению `[a-zA-Z_][a-zA-Z0-9_]*`. Любые другие неподдерживаемые символы в исходной метке должны быть преобразованы в подчёркивание. Например, лейбл `app.kubernetes.io/name` должна быть записана как `app_kubernetes_io_name`
- `<labelvalue>`: строка Unicode-символов
- `<path>`: корректный URL-путь
- `<scheme>`: строка, принимающая значения `http` или `https`
- `<secret>`: обычная строка, содержащая секрет, например пароль
- `<string>`: обычная строка
- `<size>`: размер в байтах, например `512MB`. Единица измерения обязательна. Поддерживаемые единицы: B, KB, MB, GB, TB, PB, EB
- `<tmpl_string>`: строка, которая шаблонизируется перед использованием

Остальные плейсхолдеры описаны отдельно.

Ознакомьтесь с примером корректного файла.

Глобальная конфигурация задаёт параметры, действующие во всех остальных контекстах конфигурации. Они также служат значениями по умолчанию для других разделов конфигурации.

```
global:
  # Как часто по умолчанию опрашивать цели.
  [ scrape_interval: <duration> | default = 1m ]

  # Время до истечения тайм-аута запроса опроса.
  # Оно не может превышать интервал опроса.
  [ scrape_timeout: <duration> | default = 10s ]

  # Протоколы, которые необходимо согласовать во время опроса с клиентом.
  # Поддерживаемые значения (с учетом регистра): PrometheusProto,
OpenMetricsText0.0.1,
  # OpenMetricsText1.0.0, PrometheusText0.0.4.
  # Значение по умолчанию изменяется на [ PrometheusProto, OpenMetricsText1.0.0,
OpenMetricsText0.0.1, PrometheusText0.0.4 ]
  # когда установлен флаг функции native_histogram.
  [ scrape_protocols: [<string>, ...] | default = [ OpenMetricsText1.0.0,
OpenMetricsText0.0.1, PrometheusText0.0.4 ] ]

  # Как часто оценивать правила.
  [ evaluation_interval: <duration> | default = 1m ]

  # Смещение временной метки оценки правил этой конкретной группы на
  # указанную продолжительность в прошлое, чтобы гарантировать, что
  # исходные метрики были получены. Задержки в доступности метрик
  # более вероятны, когда Deckhouse Prom++ работает как цель для удаленной записи,
  # но также могут возникать при аномалиях с опросом.
  [ rule_query_offset: <duration> | default = 0s ]

  # Метки, которые добавляются к любым временным рядам или оповещениям при
взаимодействии с
  # внешними системами (федерация, удаленное хранилище, Alertmanager).
  # Ссылки на переменные окружения `${var}` или `$var` заменяются в соответствии
  # со значениями текущих переменных окружения.
  # Ссылки на неопределенные переменные заменяются на пустую строку.
  # Символ `$` можно экранировать, используя `$$`.
  external_labels:
    [ <labelname>: <labelvalue> ... ]

  # Файл, в который записываются запросы PromQL.
  # Перезагрузка конфигурации приведет к повторному открытию файла.
  [ query_log_file: <string> ]

  # Файл, в который записываются ошибки опроса.
  # Перезагрузка конфигурации приведет к повторному открытию файла.
  [ scrape_failure_log_file: <string> ]

  # Некомпрессированное тело ответа, превышающее это количество байт, приведет к
  # сбою опроса. 0 означает отсутствие ограничения. Пример: 100MB.
```

```
  # Это экспериментальная функция, поведение может измениться или быть удалено в
будущем.
  [ body_size_limit: <size> | default = 0 ]

  # Лимит на количество образцов, принимаемых за один опрос.
  # Если после изменения лейблов количество образцов превышает это значение,
  # весь опрос будет считаться неудачным. 0 означает отсутствие ограничения.
  [ sample_limit: <int> | default = 0 ]

  # Лимит на количество лейблов, принимаемых за один образец. Если после изменения
лейблов

# количество лейблов превышает это значение, весь опрос будет считаться неудачным. 0
означает отсутствие ограничения.
  [ label_limit: <int> | default = 0 ]

  # Лимит на длину (в байтах) каждого отдельного имени лейбла. Если любое имя лейбла
  # в опросе длиннее этого значения после изменения лейблов, весь опрос будет
считаться неудачным.
  # Обратите внимание, что имена лейблов кодируются в UTF-8, и символы могут занимать
до 4 байт. 0 означает отсутствие ограничения.
  [ label_name_length_limit: <int> | default = 0 ]

  # Лимит на длину (в байтах) каждого отдельного значения лейбла. Если любое значение
лейбла
  # в опросе длиннее этого значения после изменения лейблов, весь опрос будет
считаться неудачным.
  # Обратите внимание, что значения лейблов кодируются в UTF-8, и символы могут
занимать до 4 байт. 0 означает отсутствие ограничения.
  [ label_value_length_limit: <int> | default = 0 ]

  # Лимит на количество уникальных целей для каждой конфигурации опроса,
  # которые будут приняты. Если после изменения лейблов количество целей
  # превышает это значение, Deckhouse Prom++ пометит цели как неудачные
  # без их опроса. 0 означает отсутствие ограничения. Это экспериментальная
  # функция, поведение может измениться в будущем.
  [ target_limit: <int> | default = 0 ]

  # Лимит на количество целей, отброшенных при изменении лейблов,
  # которые будут сохранены в памяти для каждой конфигурации опроса.
  # 0 означает отсутствие ограничения.
  [ keep_dropped_targets: <int> | default = 0 ]

  # Указывает схему валидации для имен метрик и лейблов. Либо пусто,
  # либо "utf8" для полной поддержки UTF-8, либо "legacy" для букв,
  # цифр, двоеточий и подчеркиваний.
  [ metric_name_validation_scheme <string> | default "utf8" ]

  # Указывает, следует ли преобразовывать все собранные классические
```

```
  # гистограммы в нативные гистограммы с пользовательскими корзинами.
  [ convert_classic_histograms_to_nhcb <bool> | default = false]

runtime:
  # Настройка параметра GOGC для сборщика мусора Go
  # См. https://tip.golang.org/doc/gc-guide#GOGC
  # Уменьшение этого числа увеличивает использование CPU.
  [ gogc: <int> | default = 75 ]

# Файлы правил указывают список шаблонов. Правила и оповещения читаются
# из всех соответствующих файлов.
rule_files:
  [ - <filepath_glob> ... ]

# Файлы конфигурации опроса указывают список шаблонов. Конфигурации опроса
# читаются из всех соответствующих файлов и добавляются в список конфигураций опроса.
scrape_config_files:
  [ - <filepath_glob> ... ]

# Список конфигураций опроса.
scrape_configs:
  [ - <scrape_config> ... ]

# Оповещения указывают настройки, связанные с Alertmanager.
alerting:
  alert_relabel_configs:
    [ - <relabel_config> ... ]
  alertmanagers:
    [ - <alertmanager_config> ... ]

# Настройки, связанные с функцией удаленной записи.
remote_write:
  [ - <remote_write> ... ]

# Настройки, связанные с функцией приема OTLP.
# См. https://prometheus.io/docs/guides/opentelemetry/ для лучших практик.
otlp:
  [ promote_resource_attributes: [<string>, ...] | default = [ ] ]
  # Настройка перевода метрик OTLP при получении через конечную точку метрик OTLP.
  # Доступные значения:
  # - "UnderscoreEscapingWithSuffixes" относится к общепринятой нормализации,
  #   используемой OpenTelemetry в https://github.com/open-telemetry/opentelemetry-
collector-contrib/tree/main/pkg/translator/prometheus
  # - "NoUTF8EscapingWithSuffixes" — это режим, который полагается на поддержку UTF-8
в Deckhouse Prom++.
  #   Он сохраняет все специальные символы, такие как точки, но все же добавляет
необходимые суффиксы
  #   для имен метрик для единиц и _total, как это делает
UnderscoreEscapingWithSuffixes.
```

```
  # - (ЭКСПЕРИМЕНТАЛЬНО) "NoTranslation" — это режим, который полагается на поддержку
UTF-8 в Deckhouse Prom++.
  #   Он сохраняет все специальные символы, такие как точки, и не будет добавлять
специальные суффиксы
  #   для единиц и типов метрик.
  #
  #   ПРЕДУПРЕЖДЕНИЕ: Настройка "NoTranslation" имеет значительные известные риски и
ограничения
  #   (см. https://prometheus.io/docs/practices/naming/ для деталей):
  #       * Ухудшение UX при использовании PromQL в простом YAML (например,
оповещения, правила, панели, конфигурация автоскейлинга).
  #       * Коллизии серий, которые в лучшем случае могут привести к ошибкам OOO, в
худшем случае к тихо поврежденной
  #         временной серии. Например, вы можете оказаться в ситуации, когда
загружаете серию `foo.bar` с единицей
  #         `seconds` и отдельную серию `foo.bar` с единицей `milliseconds`.
  [ translation_strategy: <string> | default = "UnderscoreEscapingWithSuffixes" ]
  # Включает добавление атрибутов ресурсов "service.name", "service.namespace" и
"service.instance.id"
  # к метрике "target_info", помимо их преобразования в лейблы "instance" и "job".
  [ keep_identifying_resource_attributes: <boolean> | default = false]
  # Настройка опционального перевода явных гистограмм с корзинами OTLP в нативные
гистограммы с пользовательскими корзинами.
  [ convert_histograms_to_nhcb: <boolean> | default = false]

# Настройки, связанные с функцией удаленного чтения.
remote_read:
  [ - <remote_read> ... ]

# Настройки, связанные с хранилищем, которые могут быть перезагружены во время
выполнения.
storage:
  [ tsdb: <tsdb> ]
  [ exemplars: <exemplars> ]

# Настройка экспорта трассировок.
tracing:
  [ <tracing_config> ]
```

scrape_config

Секция `scrape_config` определяет набор целей и параметры, описывающие, как их опрашивать. В общем случае одна конфигурация опроса определяет одну задачу. В сложных конфигурациях это может измениться.

Цели могут быть статически настроены через параметр `static_configs` или динамически обнаружены с использованием одного из поддерживаемых механизмов обнаружения сервисов.

Additionally, `relabel_configs` allow advanced modifications to any target and its labels before scraping.

```
# Имя задания, присваиваемое собранным метрикам по умолчанию.
job_name: <job_name>

# Как часто опрашивать цели из этого задания.
[ scrape_interval: <duration> | default = <global_config.scrape_interval> ]

# Тайм-аут для каждого опроса при сборе данных для этого задания.
# Он не может быть больше, чем интервал опроса.
[ scrape_timeout: <duration> | default = <global_config.scrape_timeout> ]

# Протоколы, которые необходимо согласовать во время опроса с клиентом.
# Поддерживаемые значения (с учетом регистра): PrometheusProto, OpenMetricsText0.0.1,
# OpenMetricsText1.0.0, PrometheusText0.0.4, PrometheusText1.0.0.
[ scrape_protocols: [<string>, ...] | default = <global_config.scrape_protocols> ]

# Резервный протокол, который будет использоваться, если опрос возвращает пустой,
# неразборчивый или иным образом недопустимый Content-Type.
# Поддерживаемые значения (с учетом регистра): PrometheusProto, OpenMetricsText0.0.1,
# OpenMetricsText1.0.0, PrometheusText0.0.4, PrometheusText1.0.0.
[ fallback_scrape_protocol: <string> ]

# Следует ли собирать классическую гистограмму, даже если она также представлена как
# нативная гистограмма (не имеет эффекта без --enable-feature=native-histograms).
[ always_scrape_classic_histograms: <boolean> | default = false ]

# HTTP-ресурсный путь, по которому извлекаются метрики из целей.
[ metrics_path: <path> | default = /metrics ]

# honor_labels управляет тем, как Deckhouse Prom++ обрабатывает конфликты между
лейблами,
# которые уже присутствуют в собранных данных, и лейблами, которые Deckhouse Prom++
добавил бы
# на стороне сервера ("job" и "instance" лейблы, вручную настроенные лейблы целей,
# и лейблы, сгенерированные реализациями обнаружения сервисов).
#
# Если honor_labels установлено в "true", конфликты лейблов разрешаются путем
сохранения
# значений лейблов из собранных данных и игнорирования конфликтующих лейблов на
стороне сервера.
#
# Если honor_labels установлено в "false", конфликты лейблов разрешаются путем
переименования
# конфликтующих лейблов в собранных данных в "exported_<original-label>" (например,
"exported_instance",
# "exported_job"), а затем добавления лейблов на стороне сервера.
#
# Установка honor_labels в "true" полезна для таких случаев использования, как
федерация и
```

```
# сбор данных с Pushgateway, где все лейблы, указанные в цели, должны быть сохранены.
#
# Обратите внимание, что любые глобально настроенные "external_labels" не
затрагиваются этой
# настройкой. В общении с внешними системами они всегда применяются только тогда,
когда временной
# ряд еще не имеет данного лейбла, и игнорируются в противном случае.
[ honor_labels: <boolean> | default = false ]

# honor_timestamps управляет тем, уважает ли Deckhouse Prom++ временные метки,
# присутствующие в собранных данных.
#
# Если honor_timestamps установлено в "true", временные метки метрик, предоставленных
# целью, будут использоваться.
#
# Если honor_timestamps установлено в "false", временные метки метрик, предоставленных
# целью, будут игнорироваться.
[ honor_timestamps: <boolean> | default = true ]

# track_timestamps_staleness управляет тем, отслеживает ли Deckhouse Prom++
устаревание
# метрик, которые имеют явные временные метки в собранных данных.
#
# Если track_timestamps_staleness установлено в "true", маркер устаревания будет
# вставлен в TSDB, когда метрика больше не присутствует или цель недоступна.
[ track_timestamps_staleness: <boolean> | default = false ]

# Настраивает схему протокола, используемую для запросов.
[ scheme: <scheme> | default = http ]

# Необязательные параметры URL HTTP.
params:
  [ <string>: [<string>, ...] ]

# Если enable_compression установлено в "false", Deckhouse Prom++ запросит
# не сжатый ответ от собранной цели.
[ enable_compression: <boolean> | default = true ]

# Файл, в который записываются ошибки опроса.
# Перезагрузка конфигурации откроет файл заново.
[ scrape_failure_log_file: <string> ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]

# Список конфигураций обнаружения сервисов Azure.
```

```
azure_sd_configs:
  [ - <azure_sd_config> ... ]

# Список конфигураций обнаружения сервисов Consul.
consul_sd_configs:
  [ - <consul_sd_config> ... ]

# Список конфигураций обнаружения сервисов DigitalOcean.
digitalocean_sd_configs:
  [ - <digitalocean_sd_config> ... ]

# Список конфигураций обнаружения сервисов Docker.
docker_sd_configs:
  [ - <docker_sd_config> ... ]

# Список конфигураций обнаружения сервисов Docker Swarm.
dockerswarm_sd_configs:
  [ - <dockerswarm_sd_config> ... ]

# Список конфигураций обнаружения сервисов DNS.
dns_sd_configs:
  [ - <dns_sd_config> ... ]

# Список конфигураций обнаружения сервисов EC2.
ec2_sd_configs:
  [ - <ec2_sd_config> ... ]

# Список конфигураций обнаружения сервисов Eureka.
eureka_sd_configs:
  [ - <eureka_sd_config> ... ]

# Список конфигураций обнаружения сервисов файлов.
file_sd_configs:
  [ - <file_sd_config> ... ]

# Список конфигураций обнаружения сервисов GCE.
gce_sd_configs:
  [ - <gce_sd_config> ... ]

# Список конфигураций обнаружения сервисов Hetzner.
hetzner_sd_configs:
  [ - <hetzner_sd_config> ... ]

# Список конфигураций обнаружения сервисов HTTP.
http_sd_configs:
  [ - <http_sd_config> ... ]

# Список конфигураций обнаружения сервисов IONOS.
ionos_sd_configs:
```

```
  [ - <ionos_sd_config> ... ]

# Список конфигураций обнаружения сервисов Kubernetes.
kubernetes_sd_configs:
  [ - <kubernetes_sd_config> ... ]

# Список конфигураций обнаружения сервисов Kuma.
kuma_sd_configs:
  [ - <kuma_sd_config> ... ]

# Список конфигураций обнаружения сервисов Lightsail.
lightsail_sd_configs:
  [ - <lightsail_sd_config> ... ]

# Список конфигураций обнаружения сервисов Linode.
linode_sd_configs:
  [ - <linode_sd_config> ... ]

# Список конфигураций обнаружения сервисов Marathon.
marathon_sd_configs:
  [ - <marathon_sd_config> ... ]

# Список конфигураций обнаружения сервисов AirBnB's Nerve.
nerve_sd_configs:
  [ - <nerve_sd_config> ... ]

# Список конфигураций обнаружения сервисов Nomad.
nomad_sd_configs:
  [ - <nomad_sd_config> ... ]

# Список конфигураций обнаружения сервисов OpenStack.
openstack_sd_configs:
  [ - <openstack_sd_config> ... ]

# Список конфигураций обнаружения сервисов OVHcloud.
ovhcloud_sd_configs:
  [ - <ovhcloud_sd_config> ... ]

# Список конфигураций обнаружения сервисов PuppetDB.
puppetdb_sd_configs:
  [ - <puppetdb_sd_config> ... ]

# Список конфигураций обнаружения сервисов Scaleway.
scaleway_sd_configs:
  [ - <scaleway_sd_config> ... ]

# Список конфигураций обнаружения сервисов Zookeeper Serverset.
serverset_sd_configs:
  [ - <serverset_sd_config> ... ]
```

```
# Список конфигураций обнаружения сервисов Triton.
triton_sd_configs:
  [ - <triton_sd_config> ... ]

# Список конфигураций обнаружения сервисов Uyuni.
uyuni_sd_configs:
  [ - <uyuni_sd_config> ... ]

# Список статически настроенных целей с лейблами для этого задания.
static_configs:
  [ - <static_config> ... ]

# Список конфигураций переназначения целей.
relabel_configs:
  [ - <relabel_config> ... ]

# Список конфигураций переназначения метрик.
metric_relabel_configs:
  [ - <relabel_config> ... ]

# Несжатое тело ответа, превышающее это количество байт, приведет к
# сбою опроса. 0 означает отсутствие ограничения. Пример: 100MB.
# Это экспериментальная функция, это поведение может
# измениться или быть удалено в будущем.
[ body_size_limit: <size> | default = 0 ]

# Лимит на количество собранных образцов, которые будут приняты за один опрос.
# Если после переназначения метрик присутствует больше этого количества образцов,
# весь опрос будет считаться неудачным. 0 означает отсутствие ограничения.
[ sample_limit: <int> | default = 0 ]

# Лимит на количество лейблов, которые будут приняты на один образец. Если после
# переназначения метрик на любом образце присутствует больше этого количества лейблов,
# весь опрос будет считаться неудачным. 0 означает отсутствие ограничения.
[ label_limit: <int> | default = 0 ]

# Лимит на длину (в байтах) каждого отдельного имени лейбла. Если любое имя лейбла
# в опросе длиннее этого числа после переназначения метрик, весь опрос будет
# считаться неудачным. Обратите внимание, что имена лейблов кодируются в UTF-8,
# и символы могут занимать до 4 байт. 0 означает отсутствие ограничения.
[ label_name_length_limit: <int> | default = 0 ]

# Лимит на длину (в байтах) каждого отдельного значения лейбла. Если любое значение
# лейбла в опросе длиннее этого числа после переназначения метрик, весь опрос будет
# считаться неудачным. Обратите внимание, что значения лейблов кодируются в UTF-8,
# и символы могут занимать до 4 байт. 0 означает отсутствие ограничения.
[ label_value_length_limit: <int> | default = 0 ]
```

```
# Лимит на количество уникальных целей, которые будут приняты за одну конфигурацию
# опроса. Если после переназначения целей присутствует больше этого количества целей,
# Deckhouse Prom++ отметит цели как неудачные без их опроса. 0 означает отсутствие
# ограничения. Это экспериментальная функция, это поведение может измениться в
будущем.
[ target_limit: <int> | default = 0 ]

# Лимит на количество целей, отброшенных переназначением, которые будут храниться в
# памяти за одну конфигурацию опроса. 0 означает отсутствие ограничения.
[ keep_dropped_targets: <int> | default = 0 ]

# Указывает схему валидации для имен метрик и лейблов. Либо пусто, либо
# "utf8" для полной поддержки UTF-8, либо "legacy" для букв, цифр, двоеточий и
# подчеркиваний.
[ metric_name_validation_scheme <string> | default "utf8" ]

# Указывает схему экранирования символов, которая будет запрашиваться при сборе
# метрик и имен лейблов, которые не соответствуют набору символов legacy Deckhouse
Prom++.
# Доступные варианты:
#   * `allow-utf-8`: Полная поддержка UTF-8, экранирование не требуется.
#   * `underscores`: Экранирует все недопустимые символы legacy в подчеркивания.
#   * `dots`: Экранирует точки в `_dot_`, подчеркивания в `__`, и все другие
#     недопустимые символы legacy в подчеркивания.
#   * `values`: Добавляет к имени префикс `U__` и заменяет все недопустимые
#     символы их значением в юникоде, окруженным подчеркиваниями. Одинарные
#     подчеркивания заменяются на двойные подчеркивания.
#     например, "U__my_2e_dotted_2e_name".
# Если это значение оставлено пустым, Deckhouse Prom++ по умолчанию будет использовать
# `allow-utf-8`, если схема валидации для текущей конфигурации опроса установлена в
utf8,
# или `underscores`, если схема валидации установлена в `legacy`.
[ metric_name_validation_scheme <string> | default "utf8" ]

# Лимит на общее количество положительных и отрицательных корзин, разрешенных в одной
# нативной гистограмме. Разрешение гистограммы с большим количеством корзин будет
# уменьшено до тех пор, пока количество корзин не будет в пределах лимита. Если лимит
# не может быть достигнут, опрос завершится неудачей.
# 0 означает отсутствие ограничения.
[ native_histogram_bucket_limit: <int> | default = 0 ]

# Нижний предел для коэффициента роста от одной корзины к следующей в каждой нативной
# гистограмме. Разрешение гистограммы с более низким коэффициентом роста будет
# уменьшено настолько, насколько это возможно, пока оно не будет в пределах лимита.
# Чтобы установить верхний предел для схемы (эквивалентно "scale" в экспоненциальных
# гистограммах OTel), используйте следующие пределы коэффициентов:
#
# +----------------------------+----------------------------+
# |        коэффициент роста   | результирующая схема       |
```

```
# +----------------------------+----------------------------+
# |          65536             |             -4             |
# +----------------------------+----------------------------+
# |            256             |             -3             |
# +----------------------------+----------------------------+
# |             16             |             -2             |
# +----------------------------+----------------------------+
# |              4             |             -1             |
# +----------------------------+----------------------------+
# |              2             |              0             |
# +----------------------------+----------------------------+
# |              1.4           |              1             |
# +----------------------------+----------------------------+
# |              1.1           |              2             |
# +----------------------------+----------------------------+
# |              1.09          |              3             |
# +----------------------------+----------------------------+
# |              1.04          |              4             |
# +----------------------------+----------------------------+
# |              1.02          |              5             |
# +----------------------------+----------------------------+
# |              1.01          |              6             |
# +----------------------------+----------------------------+
# |              1.005         |              7             |
# +----------------------------+----------------------------+
# |              1.002         |              8             |
# +----------------------------+----------------------------+
#
# 0 приводит к наименьшему поддерживаемому коэффициенту (который в настоящее время
~1.0027 или
# схема 8, но может измениться в будущем).
[ native_histogram_min_bucket_factor: <float> | default = 0 ]

# Указывает, следует ли преобразовывать классические гистограммы в нативные
гистограммы с
# пользовательскими корзинами (не имеет эффекта без --enable-feature=native-
histograms).
[ convert_classic_histograms_to_nhcb <bool> | default =
<global.convert_classic_histograms_to_nhcb>]
```

Где `<job_name>` должен быть уникальным для всех конфигураций опроса.

http_config

`http_config` позволяет настраивать HTTP-запросы.

```
# Устанавливает заголовок `Authorization` в каждом запросе с
# настроенным именем пользователя и паролем.
# username и username_file являются взаимоисключающими.
# password и password_file являются взаимоисключающими.
basic_auth:
  [ username: <string> ]
  [ username_file: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# Устанавливает заголовок `Authorization` в каждом запросе с
# настроенными учетными данными.
authorization:
  # Устанавливает тип аутентификации запроса.
  [ type: <string> | default: Bearer ]
  # Устанавливает учетные данные запроса. Это взаимоисключается с
  # `credentials_file`.
  [ credentials: <secret> ]
  # Устанавливает учетные данные запроса с учетными данными, прочитанными из
  # настроенного файла. Это взаимоисключается с `credentials`.
  [ credentials_file: <filename> ]

# Опциональная конфигурация OAuth 2.0.
# Не может использоваться одновременно с basic_auth или authorization.
oauth2:
  [ <oauth2> ]

# Настройка, следует ли запросам следовать за HTTP 3xx редиректами.
[ follow_redirects: <boolean> | default = true ]

# Включение HTTP2.
[ enable_http2: <boolean> | default: true ]

# Настройка параметров TLS для запроса.
tls_config:
  [ <tls_config> ]

# Опциональный URL прокси.
[ proxy_url: <string> ]
# Строка, разделенная запятыми, которая может содержать IP-адреса, нотацию CIDR,
доменные имена,
# которые должны быть исключены из проксирования. IP-адреса и доменные имена могут
# содержать номера портов.
[ no_proxy: <string> ]
# Использовать URL прокси, указанный переменными окружения (HTTP_PROXY, https_proxy,
HTTPs_PROXY, https_proxy и no_proxy)
[ proxy_from_environment: <boolean> | default: false ]
# Указывает заголовки, которые следует отправлять прокси во время CONNECT-запросов.
```

```
[ proxy_connect_header:
  [ <string>: [<secret>, ...] ] ]

# Пользовательские HTTP-заголовки, которые будут отправлены вместе с каждым запросом.
# Заголовки, которые устанавливаются самим Deckhouse Prom++, не могут быть
перезаписаны.
http_headers:
  # Имя заголовка.
  [ <string>:
    # Значения заголовка.
    [ values: [<string>, ...] ]
    # Значения заголовков. Скрыто на странице конфигурации.
    [ secrets: [<secret>, ...] ]
    # Файлы для чтения значений заголовков.
    [ files: [<string>, ...] ] ]
```

tls_config

`tls_config` позволяет настраивать TLS-соединения.

```
# Сертификат CA для проверки сертификата API сервера. Разрешено использовать не более
одного из ca и ca_file.
[ ca: <string> ]
[ ca_file: <filename> ]

# Сертификат и ключ для аутентификации клиента на сервере.
# Разрешено использовать не более одного из cert и cert_file.
# Разрешено использовать не более одного из key и key_file.
[ cert: <string> ]
[ cert_file: <filename> ]
[ key: <secret> ]
[ key_file: <filename> ]

# Расширение ServerName для указания имени сервера.
# https://tools.ietf.org/html/rfc4366#section-3.1
[ server_name: <string> ]

# Отключить проверку сертификата сервера.
[ insecure_skip_verify: <boolean> ]

# Минимально допустимая версия TLS. Допустимые значения: TLS10 (TLS 1.0), TLS11 (TLS
1.1), TLS12 (TLS 1.2), TLS13 (TLS 1.3).
# Если не установлено, Deckhouse Prom++ будет использовать минимальную версию по
умолчанию Go, которая равна TLS 1.2.
# См. MinVersion в https://pkg.go.dev/crypto/tls#Config.
[ min_version: <string> ]
# Максимально допустимая версия TLS. Допустимые значения: TLS10 (TLS 1.0), TLS11 (TLS
1.1), TLS12 (TLS 1.2), TLS13 (TLS 1.3).
# Если не установлено, Deckhouse Prom++ будет использовать максимальную версию по
умолчанию Go, которая равна TLS 1.3.
# См. MaxVersion в https://pkg.go.dev/crypto/tls#Config.
[ max_version: <string> ]
```

oauth2

Аутентификация OAuth 2.0 с использованием клиентских учетных данных или типа предоставления пароля. Deckhouse Prom++ получает токен доступа с указанной конечной точки, используя предоставленные клиентские ключи доступа и секретные ключи.

```
client_id: <string>
[ client_secret: <secret> ]

# Чтение клиентского секрета из файла.
# Это взаимоисключается с `client_secret`.
[ client_secret_file: <filename> ]

# Области для запроса токена.
scopes:
  [ - <string> ... ]

# URL для получения токена.
token_url: <string>

# Необязательные параметры для добавления к URL токена.
# Чтобы установить тип предоставления 'password', добавьте его в параметры:
# endpoint_params:
#   grant_type: 'password'
#   username: 'username@example.com'
#   password: 'strongpassword'
endpoint_params:
  [ <string>: <string> ... ]

# Настройка параметров TLS для запроса токена.
tls_config:
  [ <tls_config> ]

# Необязательный URL прокси.
[ proxy_url: <string> ]
# Строка, разделенная запятыми, которая может содержать IP-адреса, нотацию CIDR,
доменные имена,
# которые должны быть исключены из проксирования. IP и доменные имена могут
# содержать номера портов.
[ no_proxy: <string> ]
# Использовать URL прокси, указанный переменными окружения (HTTP_PROXY, https_proxy,
HTTPs_PROXY, https_proxy и no_proxy)
[ proxy_from_environment: <boolean> | default: false ]
# Указывает заголовки, которые нужно отправлять прокси во время CONNECT-запросов.
[ proxy_connect_header:
  [ <string>: [<secret>, ...] ] ]

# Пользовательские HTTP-заголовки, которые отправляются вместе с каждым запросом.
# Заголовки, установленные самим Deckhouse Prom++, не могут быть перезаписаны.
http_headers:
  # Имя заголовка.
  [ <string>:
    # Значения заголовка.
    [ values: [<string>, ...] ]
```

```
    # Значения заголовков. Скрыто на странице конфигурации.
    [ secrets: [<secret>, ...] ]
    # Файлы для чтения значений заголовков.
    [ files: [<string>, ...] ] ]
```

azure_sd_config

Конфигурации Azure SD позволяют получать цели для опроса с виртуальных машин Azure.

Для обнаружения требуется как минимум следующие разрешения:

- `Microsoft.Compute/virtualMachines/read` : Требуется для обнаружения виртуальных машин
- `Microsoft.Network/networkInterfaces/read` : Требуется для обнаружения виртуальных машин
- `Microsoft.Compute/virtualMachineScaleSets/virtualMachines/read` : Требуется для обнаружения наборов масштабирования (VMSS)
- `Microsoft.Compute/virtualMachineScaleSets/virtualMachines/networkInterfaces/read` : Требуется для обнаружения наборов масштабирования (VMSS)

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_azure_machine_id` : ID машины
- `__meta_azure_machine_location` : местоположение, в котором работает машина
- `__meta_azure_machine_name` : имя машины
- `__meta_azure_machine_computer_name` : имя компьютера машины
- `__meta_azure_machine_os_type` : операционная система машины
- `__meta_azure_machine_private_ip` : частный IP машины
- `__meta_azure_machine_public_ip` : публичный IP машины, если он существует
- `__meta_azure_machine_resource_group` : группа ресурсов машины
- `__meta_azure_machine_tag_<tagname>` : значение каждого тега машины
- `__meta_azure_machine_scale_set` : имя набора масштабирования, частью которого является виртуальная машина (это значение устанавливается только если вы используете набор масштабирования)
- `__meta_azure_machine_size` : размер машины
- `__meta_azure_subscription_id` : ID подписки
- `__meta_azure_tenant_id` : ID арендатора

См. ниже для вариантов конфигурации для обнаружения Azure:

```
# Информация для доступа к Azure API.
# Среда Azure.
[ environment: <string> | default = AzurePublicCloud ]

# Метод аутентификации: OAuth, ManagedIdentity или SDK.
# См. https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-
azure-resources/overview
# Метод аутентификации SDK по умолчанию использует переменные окружения.
# См. https://learn.microsoft.com/en-us/azure/developer/go/azure-sdk-authentication
[ authentication_method: <string> | default = OAuth]
# ID подписки. Всегда требуется.
subscription_id: <string>
# Необязательный ID арендатора. Требуется только при использовании
authentication_method OAuth.
[ tenant_id: <string> ]
# Необязательный ID клиента. Требуется только при использовании authentication_method
OAuth.
[ client_id: <string> ]
# Необязательный секрет клиента. Требуется только при использовании
authentication_method OAuth.
[ client_secret: <secret> ]

# Необязательное имя группы ресурсов. Ограничивает обнаружение этой группой ресурсов.
[ resource_group: <string> ]

# Интервал обновления для повторного чтения списка экземпляров.
[ refresh_interval: <duration> | default = 300s ]

# Порт для опроса метрик. Если используется публичный IP-адрес, это должно
# быть указано в правиле переназначения.
[ port: <int> | default = 80 ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]
```

consul_sd_config

Конфигурации Consul SD позволяют получать цели (targets) для опроса из Catalog API Consul.

Следующие мета лейблы (labels) доступны для целей (targets) во время переназначения:

- `__meta_consul_address`: адрес цели (target)
- `__meta_consul_dc`: имя датацентра для цели (target)
- `__meta_consul_health`: статус здоровья сервиса
- `__meta_consul_partition`: имя административного раздела, где зарегистрирован сервис

- `__meta_consul_metadata_<key>` : значение каждого ключа метаданных узла для цели (target)
- `__meta_consul_node` : имя узла, определенное для цели (target)
- `__meta_consul_service_address` : адрес сервиса для цели (target)
- `__meta_consul_service_id` : ID сервиса для цели (target)
- `__meta_consul_service_metadata_<key>` : значение каждого ключа метаданных сервиса для цели (target)
- `__meta_consul_service_port` : порт сервиса для цели (target)
- `__meta_consul_service` : имя сервиса, к которому принадлежит цель (target)
- `__meta_consul_tagged_address_<key>` : значение каждого ключа адреса с тегом узла для цели (target)
- `__meta_consul_tags` : список тегов цели (target), объединенных разделителем тегов

```
# Информация для доступа к Consul API. Она должна быть определена
# в соответствии с требованиями документации Consul.
[ server: <host> | default = "localhost:8500" ]
# Префикс для URI, когда Consul находится за API-шлюзом (обратным прокси).
[ path_prefix: <string> ]
[ token: <secret> ]
[ datacenter: <string> ]
# Пространства имен поддерживаются только в Consul Enterprise.
[ namespace: <string> ]
# Административные разделы поддерживаются только в Consul Enterprise.
[ partition: <string> ]
[ scheme: <string> | default = "http" ]
# Поля username и password устарели в пользу конфигурации basic_auth.
[ username: <string> ]
[ password: <secret> ]

# Список сервисов, для которых извлекаются цели (target). Если не указано,
опрашиваются все сервисы.
services:
  [ - <string> ]

# Выражение фильтра Consul, используемое для фильтрации результатов каталога.
# См. https://www.consul.io/api-docs/catalog#list-services для получения информации
# о выражениях фильтров, которые можно использовать.
[ filter: <string> ]

# Поля `tags` и `node_meta` устарели в Consul в пользу `filter`.
# Необязательный список тегов, используемых для фильтрации узлов для данного сервиса.
Сервисы должны содержать все теги в списке.
tags:
  [ - <string> ]

# Пары ключ/значение метаданных узла для фильтрации узлов для данного сервиса. Начиная
с Consul 1.14, используйте `filter`.
[ node_meta:
  [ <string>: <string> ... ] ]

# Строка, по которой теги Consul объединяются в лейбл (label) тегов.
[ tag_separator: <string> | default = , ]

# Разрешить устаревшие результаты Consul (см. https://www.consul.io/api/features/
consistency.html). Это уменьшит нагрузку на Consul.
[ allow_stale: <boolean> | default = true ]

# Время, после которого предоставленные имена обновляются.
# В больших установках может быть полезно увеличить это значение, так как каталог
будет постоянно изменяться.
[ refresh_interval: <duration> | default = 30s ]
```

```
# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]
```

Обратите внимание, что IP-адрес и порт, используемые для опроса целей, собираются как `<__meta_consul_address>:<__meta_consul_service_port>`. Однако в некоторых настройках Consul, соответствующий адрес находится в `__meta_consul_service_address`. В таких случаях вы можете использовать функцию переназначения для замены специального лейбла `__address__`.

Фаза переназначения является предпочтительным и более мощным способом фильтрации сервисов или узлов для сервиса на основе произвольных лейблов. Для пользователей с тысячами сервисов может быть более эффективно использовать API Consul напрямую, который имеет базовую поддержку фильтрации узлов (в настоящее время по метаданным узлов и одному тегу).

digitalocean_sd_config

Конфигурации SD для DigitalOcean позволяют получать цели для опроса из API Droplets DigitalOcean. Это обнаружение сервисов по умолчанию использует публичный IPv4-адрес, но это можно изменить с помощью переназначения, как показано в файле конфигурации digitalocean-sd для Deckhouse Prom++.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_digitalocean_droplet_id`: идентификатор капли
- `__meta_digitalocean_droplet_name`: имя капли
- `__meta_digitalocean_image`: слаг образа капли
- `__meta_digitalocean_image_name`: отображаемое имя образа капли
- `__meta_digitalocean_private_ipv4`: частный IPv4 капли
- `__meta_digitalocean_public_ipv4`: публичный IPv4 капли
- `__meta_digitalocean_public_ipv6`: публичный IPv6 капли
- `__meta_digitalocean_region`: регион капли
- `__meta_digitalocean_size`: размер капли
- `__meta_digitalocean_status`: статус капли
- `__meta_digitalocean_features`: список функций капли, разделенных запятыми
- `__meta_digitalocean_tags`: список тегов капли, разделенных запятыми
- `__meta_digitalocean_vpc`: идентификатор VPC капли

```
# Порт, с которого собираются метрики.
[ port: <int> | default = 80 ]

# Время, после которого капли обновляются.
[ refresh_interval: <duration> | default = 60s ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]
```

docker_sd_config

Конфигурации SD для Docker позволяют получать цели для опроса с хостов Docker Engine.

Этот SD обнаруживает “контейнеры” и создаст цель для каждого сетевого IP и порта, которые контейнер настроен для экспонирования.

Доступные мета лейблы:

- `__meta_docker_container_id` : идентификатор контейнера
- `__meta_docker_container_name` : имя контейнера
- `__meta_docker_container_network_mode` : сетевой режим контейнера
- `__meta_docker_container_label_<labelname>` : каждый лейбл контейнера, с любыми неподдерживаемыми символами, замененными на подчеркивание
- `__meta_docker_network_id` : идентификатор сети
- `__meta_docker_network_name` : имя сети
- `__meta_docker_network_ingress` : является ли сеть входящей
- `__meta_docker_network_internal` : является ли сеть внутренней
- `__meta_docker_network_label_<labelname>` : каждый лейбл сети, с любыми неподдерживаемыми символами, замененными на подчеркивание
- `__meta_docker_network_scope` : область действия сети
- `__meta_docker_network_ip` : IP контейнера в этой сети
- `__meta_docker_port_private` : порт на контейнере
- `__meta_docker_port_public` : внешний порт, если существует сопоставление портов
- `__meta_docker_port_public_ip` : публичный IP, если существует сопоставление портов

Смотрите ниже для опций конфигурации для обнаружения Docker:

```
# Адрес демона Docker.
host: <string>

# Порт, с которого собираются метрики, когда `role` — это узлы, а также для
обнаруженных
# задач и сервисов, у которых нет опубликованных портов.
[ port: <int> | default = 80 ]

# Хост, который используется, если контейнер находится в режиме сетевого хостинга.
[ host_networking_host: <string> | default = "localhost" ]

# Сортировать все ненулевые сети в порядке возрастания на основе имени сети и
# выбрать первую сеть, если у контейнера определено несколько сетей,
# таким образом избегая сбора дублирующихся целей.
[ match_first_network: <boolean> | default = true ]

# Необязательные фильтры для ограничения процесса обнаружения до подмножества
доступных
# ресурсов.
# Доступные фильтры перечислены в документации:
# https://docs.docker.com/engine/api/v1.40/#operation/ContainerList
[ filters:
  [ - name: <string>
      values: <string>, [...] ]

# Время, после которого контейнеры обновляются.
[ refresh_interval: <duration> | default = 60s ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]
```

Фаза [перелейблинга] является предпочтительным и более мощным способом фильтрации контейнеров. Для пользователей с тысячами контейнеров может быть более эффективно использовать Docker API напрямую, который имеет базовую поддержку фильтрации контейнеров (с использованием `filters`).

Смотрите [этот пример конфигурационного файла Deckhouse Prom++] для детального примера настройки Deckhouse Prom++ для Docker Engine.

dockerswarm_sd_config

Конфигурации SD для Docker Swarm позволяют получать цели для опроса с движка [Docker Swarm].

Можно настроить одну из следующих ролей для обнаружения целей:

`services`

Роль `services` обнаруживает все сервисы Swarm и экспонирует их порты как цели. Для каждого опубликованного порта сервиса создается одна цель. Если у сервиса нет опубликованных портов, создается одна цель на сервис, используя параметр `port`, определенный в конфигурации SD.

Доступные мета лейблы:

- `__meta_dockerswarm_service_id`: идентификатор сервиса
- `__meta_dockerswarm_service_name`: имя сервиса
- `__meta_dockerswarm_service_mode`: режим сервиса
- `__meta_dockerswarm_service_endpoint_port_name`: имя порта конечной точки, если доступно
- `__meta_dockerswarm_service_endpoint_port_publish_mode`: режим публикации порта конечной точки
- `__meta_dockerswarm_service_label_<labelname>`: каждый лейбл сервиса, с любыми неподдерживаемыми символами, замененными на подчеркивание
- `__meta_dockerswarm_service_task_container_hostname`: имя хоста контейнера цели, если доступно
- `__meta_dockerswarm_service_task_container_image`: образ контейнера цели
- `__meta_dockerswarm_service_updating_status`: статус сервиса, если доступно
- `__meta_dockerswarm_network_id`: идентификатор сети
- `__meta_dockerswarm_network_name`: имя сети
- `__meta_dockerswarm_network_ingress`: является ли сеть входящей
- `__meta_dockerswarm_network_internal`: является ли сеть внутренней
- `__meta_dockerswarm_network_label_<labelname>`: каждый лейбл сети, с любыми неподдерживаемыми символами, замененными на подчеркивание
- `__meta_dockerswarm_network_scope`: область действия сети

`tasks`

Роль `tasks` обнаруживает все задачи Swarm и экспонирует их порты как цели. Для каждого опубликованного порта задачи создается одна Цель. Если у задачи нет опубликованных портов, создается одна цель на задачу, используя параметр `port`, определенный в конфигурации SD.

Доступные мета лейблы:

- `__meta_dockerswarm_container_label_<labelname>`: каждый лейбл контейнера, с любыми неподдерживаемыми символами, замененными на подчеркивание
- `__meta_dockerswarm_task_id`: идентификатор задачи
- `__meta_dockerswarm_task_container_id`: идентификатор контейнера задачи
- `__meta_dockerswarm_task_desired_state`: желаемое состояние задачи
- `__meta_dockerswarm_task_slot`: слот задачи
- `__meta_dockerswarm_task_state`: состояние задачи
- `__meta_dockerswarm_task_port_publish_mode`: режим публикации порта задачи

- `__meta_dockerswarm_service_id` : идентификатор сервиса
- `__meta_dockerswarm_service_name` : имя сервиса
- `__meta_dockerswarm_service_mode` : режим сервиса
- `__meta_dockerswarm_service_label_<labelname>` : каждый лейбл сервиса, с любыми неподдерживаемыми символами, замененными на подчеркивание
- `__meta_dockerswarm_network_id` : идентификатор сети
- `__meta_dockerswarm_network_name` : имя сети
- `__meta_dockerswarm_network_ingress` : является ли сеть входящей
- `__meta_dockerswarm_network_internal` : является ли сеть внутренней
- `__meta_dockerswarm_network_label_<labelname>` : каждый лейбл сети, с любыми неподдерживаемыми символами, замененными на подчеркивание
- `__meta_dockerswarm_network_label` : каждый лейбл сети, с любыми неподдерживаемыми символами, замененными на подчеркивание
- `__meta_dockerswarm_network_scope` : область действия сети
- `__meta_dockerswarm_node_id` : идентификатор узла
- `__meta_dockerswarm_node_hostname` : имя хоста узла
- `__meta_dockerswarm_node_address` : адрес узла
- `__meta_dockerswarm_node_availability` : доступность узла
- `__meta_dockerswarm_node_label_<labelname>` : каждый лейбл узла, с любыми неподдерживаемыми символами, замененными на подчеркивание
- `__meta_dockerswarm_node_platform_architecture` : архитектура узла
- `__meta_dockerswarm_node_platform_os` : операционная система узла
- `__meta_dockerswarm_node_role` : роль узла
- `__meta_dockerswarm_node_status` : статус узла

Мета лейблы `__meta_dockerswarm_network_*` не заполняются для портов, которые опубликованы с `mode=host`.

`nodes`

Роль `nodes` используется для обнаружения узлов Swarm.

Доступные мета лейблы:

- `__meta_dockerswarm_node_address` : адрес узла
- `__meta_dockerswarm_node_availability` : доступность узла
- `__meta_dockerswarm_node_engine_version` : версия движка узла
- `__meta_dockerswarm_node_hostname` : имя хоста узла
- `__meta_dockerswarm_node_id` : идентификатор узла
- `__meta_dockerswarm_node_label_<labelname>` : каждый лейбл узла, с любыми неподдерживаемыми символами, замененными на подчеркивание
- `__meta_dockerswarm_node_manager_address` : адрес управляющего компонента узла
- `__meta_dockerswarm_node_manager_leader` : статус лидерства управляющего компонента узла (true или false)

- `__meta_dockerswarm_node_manager_reachability` : доступность управляющего компонента узла
- `__meta_dockerswarm_node_platform_architecture` : архитектура узла
- `__meta_dockerswarm_node_platform_os` : операционная система узла
- `__meta_dockerswarm_node_role` : роль узла
- `__meta_dockerswarm_node_status` : статус узла

Смотрите ниже для опций конфигурации обнаружения Docker Swarm:

```
# Адрес демона Docker.
host: <string>

# Роль целей для извлечения. Должна быть `services`, `tasks` или `nodes`.
role: <string>

# Порт для сбора метрик, когда `role` — это nodes, а также для обнаруженных
# задач и сервисов, у которых нет опубликованных портов.
[ port: <int> | default = 80 ]

# Необязательные фильтры для ограничения процесса обнаружения до подмножества
доступных
# ресурсов. Доступные фильтры перечислены в документации:
# Сервисы: https://docs.docker.com/engine/api/v1.40/#operation/ServiceList
# Задачи: https://docs.docker.com/engine/api/v1.40/#operation/TaskList
# Узлы: https://docs.docker.com/engine/api/v1.40/#operation/NodeList
[ filters:
  [ - name: <string>
      values: <string>, [...] ]

# Время, после которого данные об обнаружении сервисов обновляются.
[ refresh_interval: <duration> | default = 60s ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]
```

Фаза релейблинга является предпочтительным и более мощным способом фильтрации задач, сервисов или узлов. Для пользователей с тысячами задач может быть более эффективно использовать API Swarm напрямую, который имеет базовую поддержку фильтрации узлов (с использованием `filters`).

Смотрите этот пример конфигурационного файла Deckhouse Prom++ для детального примера настройки Deckhouse Prom++ для Docker Swarm.

dns_sd_config

Конфигурация обнаружения сервисов на основе DNS позволяет указать набор DNS доменных имен, которые периодически запрашиваются для обнаружения списка целей. DNS-серверы, к которым нужно обращаться, читаются из `/etc/resolv.conf`.

Этот метод обнаружения сервисов поддерживает только базовые DNS-запросы A, AAAA, MX, NS и SRV, но не поддерживает продвинутый подход DNS-SD, указанный в RFC6763.

Следующие мета лейблы доступны на целях во время релейблинга:

- `__meta_dns_name`: имя записи, которое привело к обнаруженной цели.
- `__meta_dns_srv_record_target`: поле цели SRV-записи
- `__meta_dns_srv_record_port`: поле порта SRV-записи
- `__meta_dns_mx_record_target`: поле цели MX-записи
- `__meta_dns_ns_record_target`: поле цели NS-записи

```
# Список DNS-доменных имен для запроса.
names:
  [ - <string> ]

# Тип DNS-запроса для выполнения. Один из SRV, A, AAAA, MX или NS.
[ type: <string> | default = 'SRV' ]

# Номер порта, используемый, если тип запроса не SRV.
[ port: <int>]

# Время, после которого предоставленные имена обновляются.
[ refresh_interval: <duration> | default = 30s ]
```

ec2_sd_config

Конфигурации EC2 SD позволяют извлекать цели сбора метрик из экземпляров AWS EC2. По умолчанию используется частный IP-адрес, но его можно изменить на публичный IP-адрес с помощью релейблинга.

Используемые IAM-учетные данные должны иметь разрешение `ec2:DescribeInstances` для обнаружения целей сбора метрик, и могут дополнительно иметь разрешение `ec2:DescribeAvailabilityZones`, если вы хотите, чтобы ID зоны доступности был доступен как лейбл (см. ниже).

Следующие мета лейблы доступны на целях во время релейблинга:

- `__meta_ec2_ami`: EC2 Amazon Machine Image
- `__meta_ec2_architecture`: архитектура экземпляра
- `__meta_ec2_availability_zone`: зона доступности, в которой работает экземпляр
- `__meta_ec2_availability_zone_id`: ID зоны доступности, в которой работает экземпляр (требуется `ec2:DescribeAvailabilityZones`)

- `__meta_ec2_instance_id` : ID экземпляра EC2
- `__meta_ec2_instance_lifecycle` : жизненный цикл экземпляра EC2, устанавливается только для ‘spot’ или ‘scheduled’ экземпляров, отсутствует в противном случае
- `__meta_ec2_instance_state` : состояние экземпляра EC2
- `__meta_ec2_instance_type` : тип экземпляра EC2
- `__meta_ec2_ipv6_addresses` : список IPv6-адресов, назначенных сетевым интерфейсам экземпляра, если они присутствуют
- `__meta_ec2_owner_id` : ID учетной записи AWS, которой принадлежит экземпляр EC2
- `__meta_ec2_platform` : платформа операционной системы, установлена на ‘windows’ на серверах Windows, отсутствует в противном случае
- `__meta_ec2_primary_ipv6_addresses` : список основных IPv6-адресов экземпляра, если они присутствуют. Список упорядочен на основе позиции каждого соответствующего сетевого интерфейса в порядке присоединения.
- `__meta_ec2_primary_subnet_id` : ID подсети основного сетевого интерфейса, если доступно
- `__meta_ec2_private_dns_name` : частное DNS-имя экземпляра, если доступно
- `__meta_ec2_private_ip` : частный IP-адрес экземпляра, если присутствует
- `__meta_ec2_public_dns_name` : публичное DNS-имя экземпляра, если доступно
- `__meta_ec2_public_ip` : публичный IP-адрес экземпляра, если доступен
- `__meta_ec2_region` : регион экземпляра
- `__meta_ec2_subnet_id` : список ID подсетей, в которых работает экземпляр, если доступно
- `__meta_ec2_tag_<tagkey>` : каждое значение тега экземпляра
- `__meta_ec2_vpc_id` : ID VPC, в которой работает экземпляр, если доступно

Смотрите ниже для опций конфигурации обнаружения EC2:

```
# Информация для доступа к API EC2.

# Регион AWS. Если пусто, используется регион из метаданных экземпляра.
[ region: <string> ]

# Пользовательская конечная точка для использования.
[ endpoint: <string> ]

# Ключи API AWS. Если пусто, используются переменные окружения `AWS_ACCESS_KEY_ID`
# и `AWS_SECRET_ACCESS_KEY`.
[ access_key: <string> ]
[ secret_key: <secret> ]
# Имя профиля AWS, используемого для подключения к API.
[ profile: <string> ]

# AWS Role ARN, альтернатива использованию ключей API AWS.
[ role_arn: <string> ]

# Интервал обновления для повторного чтения списка экземпляров.
[ refresh_interval: <duration> | default = 60s ]

# Порт для сбора метрик. Если используется публичный IP-адрес, это должно
# быть указано в правиле релейблинга.
[ port: <int> | default = 80 ]

# Фильтры могут использоваться опционально для фильтрации списка экземпляров по другим
критериям.
# Доступные критерии фильтрации можно найти здесь:
# https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_DescribeInstances.html
# Документация по API фильтров: https://docs.aws.amazon.com/AWSEC2/latest/
APIReference/API_Filter.html
filters:
  [ - name: <string>
      values: <string>, [...] ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]
```

Фаза релейблинга является предпочтительным и более мощным способом фильтрации целей на основе произвольных лейблов. Для пользователей с тысячами экземпляров может быть более эффективно использовать API EC2 напрямую, который поддерживает фильтрацию экземпляров.

openstack_sd_config

Конфигурации OpenStack SD позволяют извлекать цели сбора метрик из экземпляров OpenStack Nova.

Один из следующих типов `<openstack_role>` может быть настроен для обнаружения целей:

`hypervisor`

Роль `hypervisor` обнаруживает одну цель на узел гипервизора Nova. Адрес цели по умолчанию соответствует атрибуту `host_ip` гипервизора.

Следующие мета лейблы доступны на целях во время релейблинга:

- `__meta_openstack_hypervisor_host_ip` : IP-адрес узла гипервизора.
- `__meta_openstack_hypervisor_hostname` : имя узла гипервизора.
- `__meta_openstack_hypervisor_id` : ID узла гипервизора.
- `__meta_openstack_hypervisor_state` : состояние узла гипервизора.
- `__meta_openstack_hypervisor_status` : статус узла гипервизора.
- `__meta_openstack_hypervisor_type` : тип узла гипервизора.

`instance`

Роль `instance` обнаруживает одну цель на каждый сетевой интерфейс экземпляра Nova. Адрес цели по умолчанию соответствует частному IP-адресу сетевого интерфейса.

Следующие мета лейблы доступны на целях во время релейблинга:

- `__meta_openstack_address_pool` : пул частного IP.
- `__meta_openstack_instance_flavor` : имя flavor экземпляра OpenStack или ID flavor, если имя недоступно.
- `__meta_openstack_instance_id` : ID экземпляра OpenStack.
- `__meta_openstack_instance_image` : ID образа, который использует экземпляр OpenStack.
- `__meta_openstack_instance_name` : имя экземпляра OpenStack.
- `__meta_openstack_instance_status` : статус экземпляра OpenStack.
- `__meta_openstack_private_ip` : частный IP экземпляра OpenStack.
- `__meta_openstack_project_id` : проект (арендатор), которому принадлежит этот экземпляр.
- `__meta_openstack_public_ip` : публичный IP экземпляра OpenStack.
- `__meta_openstack_tag_<key>` : каждый элемент метаданных экземпляра, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_openstack_user_id` : учетная запись пользователя, владеющего арендатором.

`loadbalancer`

Роль `loadbalancer` обнаруживает одну цель на каждый балансировщик нагрузки Octavia с `PROMETHEUS` слушателем. Адрес цели по умолчанию соответствует VIP-адресу балансировщика нагрузки.

Следующие мета лейблы доступны на целях во время релейблинга:

- `__meta_openstack_loadbalancer_availability_zone` : зона доступности балансировщика нагрузки OpenStack.
- `__meta_openstack_loadbalancer_floating_ip` : плавающий IP балансировщика нагрузки OpenStack.
- `__meta_openstack_loadbalancer_id` : ID балансировщика нагрузки OpenStack.
- `__meta_openstack_loadbalancer_name` : имя балансировщика нагрузки OpenStack.
- `__meta_openstack_loadbalancer_provider` : провайдер Octavia балансировщика нагрузки OpenStack.
- `__meta_openstack_loadbalancer_operating_status` : операционный статус балансировщика нагрузки OpenStack.
- `__meta_openstack_loadbalancer_provisioning_status` : статус подготовки балансировщика нагрузки OpenStack.
- `__meta_openstack_loadbalancer_tags` : список тегов балансировщика нагрузки OpenStack, разделенных запятыми.
- `__meta_openstack_loadbalancer_vip` : VIP балансировщика нагрузки OpenStack.
- `__meta_openstack_project_id` : проект (арендатор), которому принадлежит этот балансировщик нагрузки.

См. ниже для параметров конфигурации для обнаружения OpenStack:

```
# Информация для доступа к API OpenStack.

# Роль OpenStack для сущностей, которые должны быть обнаружены.
role: <openstack_role>

# Регион OpenStack.
region: <string>

# identity_endpoint указывает HTTP-эндпоинт, который необходим для работы с
# API идентификации соответствующей версии. Хотя он в конечном итоге нужен всем
# сервисам идентификации, он часто заполняется функцией на уровне провайдера.
[ identity_endpoint: <string> ]

# username требуется, если используется API Identity V2. Обратитесь к панели
управления вашего провайдера, чтобы узнать имя пользователя вашей учетной записи. В
Identity V3 требуется либо userid, либо комбинация username и domain_id или
domain_name.
[ username: <string> ]
[ userid: <string> ]

# password для API Identity V2 и V3. Обратитесь к панели управления вашего провайдера,
чтобы узнать предпочтительный метод аутентификации вашей учетной записи.
[ password: <secret> ]

# Не более одного из domain_id и domain_name должно быть предоставлено, если
используется username с Identity V3. В противном случае оба являются необязательными.
[ domain_name: <string> ]
[ domain_id: <string> ]

# Поля project_id и project_name являются необязательными для API Identity V2.
Некоторые провайдеры позволяют указать project_name вместо project_id. Некоторые
требуют оба. Политики аутентификации вашего провайдера определят, как эти поля влияют
на аутентификацию.
[ project_name: <string> ]
[ project_id: <string> ]

# Поля application_credential_id или application_credential_name требуются, если
используется учетные данные приложения для аутентификации. Некоторые провайдеры
позволяют создать учетные данные приложения для аутентификации вместо пароля.
[ application_credential_name: <string> ]
[ application_credential_id: <string> ]

# Поле application_credential_secret требуется, если используется учетные данные
приложения для аутентификации.
[ application_credential_secret: <secret> ]

# Должно ли обнаружение сервисов перечислять все экземпляры для всех проектов.
# Это актуально только для роли 'instance' и обычно требует прав администратора.
```

```
[ all_tenants: <boolean> | default: false ]

# Интервал обновления для повторного чтения списка экземпляров.
[ refresh_interval: <duration> | default = 60s ]

# Порт для сбора метрик. Если используется публичный IP-адрес, это должно
# быть указано в правиле релейблинга.
[ port: <int> | default = 80 ]

# Доступность конечной точки для подключения. Должна быть одной из public, admin или
internal.
[ availability: <string> | default = "public" ]

# Настройки TLS.
tls_config:
  [ <tls_config> ]
```

ovhcloud_sd_config

Конфигурации OVHcloud SD позволяют извлекать цели сбора метрик с выделенных серверов и VPS OVHcloud, используя их API. Deckhouse Prom++ будет периодически проверять REST-эндпоинт и создавать цель для каждого обнаруженного сервера. Роль будет пытаться использовать публичный IPv4-адрес в качестве адреса по умолчанию, если его нет, она попытается использовать IPv6. Это может быть изменено с помощью релейблинга. Для публичных облачных экземпляров OVHcloud вы можете использовать openstack_sd_config.

VPS

- `__meta_ovhcloud_vps_cluster` : кластер сервера
- `__meta_ovhcloud_vps_datacenter` : датацентр сервера
- `__meta_ovhcloud_vps_disk` : диск сервера
- `__meta_ovhcloud_vps_display_name` : отображаемое имя сервера
- `__meta_ovhcloud_vps_ipv4` : IPv4 сервера
- `__meta_ovhcloud_vps_ipv6` : IPv6 сервера
- `__meta_ovhcloud_vps_keymap` : раскладка клавиатуры KVM сервера
- `__meta_ovhcloud_vps_maximum_additional_ip` : максимальное количество дополнительных IP сервера
- `__meta_ovhcloud_vps_memory_limit` : лимит памяти сервера
- `__meta_ovhcloud_vps_memory` : память сервера
- `__meta_ovhcloud_vps_monitoring_ip_blocks` : блоки IP для мониторинга сервера
- `__meta_ovhcloud_vps_name` : имя сервера
- `__meta_ovhcloud_vps_netboot_mode` : режим сетевой загрузки сервера
- `__meta_ovhcloud_vps_offer_type` : тип предложения сервера
- `__meta_ovhcloud_vps_offer` : предложение сервера
- `__meta_ovhcloud_vps_state` : состояние сервера
- `__meta_ovhcloud_vps_vcore` : количество виртуальных ядер сервера

- `__meta_ovhcloud_vps_version` : версия сервера
- `__meta_ovhcloud_vps_zone` : зона сервера

Выделенные серверы

- `__meta_ovhcloud_dedicated_server_commercial_range` : коммерческий диапазон сервера
- `__meta_ovhcloud_dedicated_server_datacenter` : датацентр сервера
- `__meta_ovhcloud_dedicated_server_ipv4` : IPv4 сервера
- `__meta_ovhcloud_dedicated_server_ipv6` : IPv6 сервера
- `__meta_ovhcloud_dedicated_server_link_speed` : скорость соединения сервера
- `__meta_ovhcloud_dedicated_server_name` : имя сервера
- `__meta_ovhcloud_dedicated_server_no_intervention` : отключено ли вмешательство датацентра для сервера
- `__meta_ovhcloud_dedicated_server_os` : операционная система сервера
- `__meta_ovhcloud_dedicated_server_rack` : стойка сервера
- `__meta_ovhcloud_dedicated_server_reverse` : обратное DNS-имя сервера
- `__meta_ovhcloud_dedicated_server_server_id` : ID сервера
- `__meta_ovhcloud_dedicated_server_state` : состояние сервера
- `__meta_ovhcloud_dedicated_server_support_level` : уровень поддержки сервера

См. ниже для параметров конфигурации для обнаружения OVHcloud:

```
# Ключ доступа для использования. https://api.ovh.com
application_key: <string>
application_secret: <secret>
consumer_key: <secret>
# Сервис целей для извлечения. Должен быть `vps` или `dedicated_server`.
service: <string>
# API-эндпоинт. https://github.com/ovh/go-ovh#supported-apis
[ endpoint: <string> | default = "ovh-eu" ]
# Интервал обновления для повторного чтения списка ресурсов.
[ refresh_interval: <duration> | default = 60s ]
```

puppetdb_sd_config

Конфигурации PuppetDB SD позволяют извлекать цели сбора метрик из ресурсов PuppetDB.

Это SD обнаруживает ресурсы и создаст цель для каждого ресурса, возвращенного API.

Адрес ресурса — это `certname` ресурса и может быть изменен во время релейблинга.

Следующие мета лейблы доступны на целях во время релейблинга:

- `__meta_puppetdb_query` : запрос на языке Puppet Query Language (PQL)
- `__meta_puppetdb_certname` : имя узла, связанного с ресурсом
- `__meta_puppetdb_resource` : SHA-1 хеш типа ресурса, заголовка и параметров для идентификации

- `__meta_puppetdb_type`: тип ресурса
- `__meta_puppetdb_title`: заголовок ресурса
- `__meta_puppetdb_exported`: экспортирован ли ресурс (`"true"` или `"false"`)
- `__meta_puppetdb_tags`: список тегов ресурса, разделенных запятыми
- `__meta_puppetdb_file`: файл манифеста, в котором был объявлен ресурс
- `__meta_puppetdb_environment`: окружение узла, связанного с ресурсом
- `__meta_puppetdb_parameter_<parametername>`: параметры ресурса

См. ниже для параметров конфигурации для обнаружения PuppetDB:

```
# URL корневой точки запроса PuppetDB.
url: <string>

# Запрос на языке Puppet Query Language (PQL). Поддерживаются только ресурсы.
# https://puppet.com/docs/puppetdb/latest/api/query/v4/pql.html
query: <string>

# Включать ли параметры как мета лейблы.
# Из-за различий между типами параметров и лейблами Deckhouse Prom++,
# некоторые параметры могут не отображаться. Формат параметров также может
# измениться в будущих выпусках.
#
# Примечание: Включение этого параметра раскрывает параметры в интерфейсе и API
Deckhouse Prom++. Убедитесь, что у вас нет секретов, раскрытых в качестве параметров,
если вы включаете это.
[ include_parameters: <boolean> | default = false ]

# Интервал обновления для повторного чтения списка ресурсов.
[ refresh_interval: <duration> | default = 60s ]

# Порт для сбора метрик.
[ port: <int> | default = 80 ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и авторизация), конфигурации прокси, параметры TLS, пользовательские
HTTP-заголовки и т.д.
[ <http_config> ]
```

file_sd_config

Обнаружение сервисов на основе файлов предоставляет более универсальный способ настройки статических целей и служит интерфейсом для подключения пользовательских механизмов обнаружения сервисов.

Он читает набор файлов, содержащих список из нуля или более `<static_config>`. Изменения во всех определенных файлах обнаруживаются с помощью наблюдения за диском и применяются немедленно.

Пока эти отдельные файлы отслеживаются на предмет изменений, родительская директория также отслеживается неявно. Это необходимо для эффективной обработки атомарного переименования и для обнаружения новых файлов, которые соответствуют настроенным шаблонам. Это может вызвать проблемы, если родительская директория содержит большое количество других файлов, так как каждый из этих файлов также будет отслеживаться, даже если события, связанные с ними, не имеют значения.

Файлы могут быть предоставлены в формате YAML или JSON. Применяются только изменения, приводящие к корректно сформированным группам целей.

Файлы должны содержать список статических конфигураций, используя следующие форматы:

- JSON

```
[
  {
    "targets": [ "<host>", ... ],
    "labels": {
      "<labelname>": "<labelvalue>", ...
    }
  },
  ...
]
```

- YAML

```
- targets:
  [ - '<host>' ]
  labels:
    [ <labelname>: <labelvalue> ... ]
```

В качестве резервного варианта содержимое файла также перечитывается периодически с указанным интервалом обновления.

Каждая цель имеет мета лейбл `__meta_filepath` во время фазы релейблинга. Его значение устанавливается в путь к файлу, из которого была извлечена цель.

Существует список интеграций с этим механизмом обнаружения.

```
# Шаблоны для файлов, из которых извлекаются группы целей.
files:
  [ - <filename_pattern> ... ]

# Интервал обновления для повторного чтения файлов.
[ refresh_interval: <duration> | default = 5m ]
```

Где `<filename_pattern>` может быть путем, оканчивающимся на `.json`, `.yml` или `.yaml`. Последний сегмент пути может содержать один `*`, который соответствует любой последовательности символов, например, `my/path/tg_*.json`.

gce_sd_config

Конфигурации GCE SD позволяют извлекать цели сбора метрик из экземпляров GCP GCE. По умолчанию используется частный IP-адрес, но его можно изменить на публичный IP-адрес с помощью релейблинга.

Следующие мета лейблы доступны на целях во время релейблинга:

- `__meta_gce_instance_id` : числовой идентификатор экземпляра
- `__meta_gce_instance_name` : имя экземпляра
- `__meta_gce_label_<labelname>` : каждый лейбл GCE экземпляра, с любыми неподдерживаемыми символами, преобразованными в подчеркивание
- `__meta_gce_machine_type` : полный или частичный URL типа машины экземпляра
- `__meta_gce_metadata_<name>` : каждый элемент метаданных экземпляра
- `__meta_gce_network` : URL сети экземпляра
- `__meta_gce_private_ip` : частный IP-адрес экземпляра
- `__meta_gce_interface_ipv4_<name>` : IPv4-адрес каждого именованного интерфейса
- `__meta_gce_project` : проект GCP, в котором работает экземпляр
- `__meta_gce_public_ip` : публичный IP-адрес экземпляра, если он присутствует
- `__meta_gce_subnetwork` : URL подсети экземпляра
- `__meta_gce_tags` : список тегов экземпляра, разделенных запятыми
- `__meta_gce_zone` : URL зоны GCE, в которой работает экземпляр

См. ниже параметры конфигурации для обнаружения GCE:

```
# Информация для доступа к API GCE.

# Проект GCP
project: <string>

# Зона целей сбора метрик. Если вам нужно несколько зон, используйте несколько
gce_sd_configs.
zone: <string>

# Фильтр может быть использован для фильтрации списка экземпляров по другим критериям.
Синтаксис этой строки фильтра описан здесь в разделе параметра запроса фильтра:
# https://cloud.google.com/compute/docs/reference/latest/instances/list
[ filter: <string> ]

# Интервал обновления для повторного чтения списка экземпляров
[ refresh_interval: <duration> | default = 60s ]

# Порт для сбора метрик. Если используется публичный IP-адрес, это должно быть указано
в правиле релейблинга.
[ port: <int> | default = 80 ]

# Разделитель тегов используется для разделения тегов при конкатенации
[ tag_separator: <string> | default = , ]
```

Учетные данные обнаруживаются клиентом по умолчанию Google Cloud SDK, который ищет их в следующих местах, предпочитая первое найденное местоположение:

1. JSON-файл, указанный переменной окружения `GOOGLE_APPLICATION_CREDENTIALS`
2. JSON-файл в известном пути `$HOME/.config/gcloud/application_default_credentials.json`
3. полученные с сервера метаданных GCE

Если Deckhouse Prom++ работает внутри GCE, учетная запись сервиса, связанная с экземпляром, на котором он работает, должна иметь как минимум права только для чтения к вычислительным ресурсам. Если работает вне GCE, убедитесь, что создана соответствующая учетная запись сервиса и файл учетных данных размещен в одном из ожидаемых мест.

hetzner_sd_config

Конфигурации Hetzner SD позволяют извлекать цели сбора метрик из API Hetzner Cloud и API Robot. Это обнаружение сервисов по умолчанию использует публичный IPv4-адрес, но это можно изменить с помощью релейблинга, как показано в файле конфигурации Deckhouse Prom++ hetzner-sd.

Следующие мета лейблы доступны на всех целях во время релейблинга:

- `__meta_hetzner_server_id` : идентификатор сервера
- `__meta_hetzner_server_name` : имя сервера

- `__meta_hetzner_server_status` : статус сервера
- `__meta_hetzner_public_ipv4` : публичный IPv4-адрес сервера
- `__meta_hetzner_public_ipv6_network` : публичная IPv6-сеть (/64) сервера
- `__meta_hetzner_datacenter` : датацентр сервера

лейблы ниже доступны только для целей с `role` , установленным в `hcloud` :

- `__meta_hetzner_hcloud_image_name` : имя образа сервера
- `__meta_hetzner_hcloud_image_description` : описание образа сервера
- `__meta_hetzner_hcloud_image_os_flavor` : ОС образа сервера
- `__meta_hetzner_hcloud_image_os_version` : версия ОС образа сервера
- `__meta_hetzner_hcloud_datacenter_location` : местоположение сервера
- `__meta_hetzner_hcloud_datacenter_location_network_zone` : зона сети сервера
- `__meta_hetzner_hcloud_server_type` : тип сервера
- `__meta_hetzner_hcloud_cpu_cores` : количество ядер процессора сервера
- `__meta_hetzner_hcloud_cpu_type` : тип процессора сервера (общий или выделенный)
- `__meta_hetzner_hcloud_memory_size_gb` : объем памяти сервера (в ГБ)
- `__meta_hetzner_hcloud_disk_size_gb` : размер диска сервера (в ГБ)
- `__meta_hetzner_hcloud_private_ipv4_<networkname>` : частный IPv4-адрес сервера в заданной сети
- `__meta_hetzner_hcloud_label_<labelname>` : каждый лейбл сервера, с любыми неподдерживаемыми символами, преобразованными в подчеркивание
- `__meta_hetzner_hcloud_labelpresent_<labelname>` : `true` для каждого лейбла сервера, с любыми неподдерживаемыми символами, преобразованными в подчеркивание

лейблы ниже доступны только для целей с `role` , установленным в `robot` :

- `__meta_hetzner_robot_product` : продукт сервера
- `__meta_hetzner_robot_cancelled` : статус отмены сервера

```
# Роль Hetzner для сущностей, которые должны быть обнаружены.
# Одна из robot или hcloud.
role: <string>

# Порт для сбора метрик.
[ port: <int> | default = 80 ]

# Время, после которого серверы обновляются.
[ refresh_interval: <duration> | default = 60s ]

# Селектор лейблов, используемый для фильтрации серверов при их получении из API.
Подробнее см. https://docs.hetzner.cloud/#label-selector.
# Используется только, когда роль - hcloud.
[ label_selector: <string> ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и авторизация), конфигурации прокси, параметры TLS, пользовательские
HTTP-заголовки и т.д.
[ <http_config> ]
```

http_sd_config

Обнаружение сервисов на основе HTTP предоставляет более универсальный способ настройки статических целей и служит интерфейсом для подключения пользовательских механизмов обнаружения сервисов.

Он извлекает цели из HTTP-эндпоинта, содержащего список из нуля или более `<static_config>`. цель должна отвечать с HTTP-ответом 200. HTTP-заголовок `Content-Type` должен быть `application/json`, а тело должно быть корректным JSON.

Пример тела ответа:

```
[
  {
    "targets": [ "<host>", ... ],
    "labels": {
      "<labelname>": "<labelvalue>", ...
    }
  },
  ...
]
```

Эндпоинт запрашивается периодически с указанным интервалом обновления. Метрика счетчика `prometheus_sd_http_failures_total` отслеживает количество неудачных обновлений.

Каждая цель имеет мета лейбл `__meta_url` во время фазы релейблинга. Его значение устанавливается в URL, из которого была извлечена цель.

```
# URL, из которого извлекаются цели.
url: <string>

# Интервал обновления для повторного запроса эндпоинта.
[ refresh_interval: <duration> | default = 60s ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и авторизация), конфигурации прокси, параметры TLS, пользовательские
HTTP-заголовки и т.д.
[ <http_config> ]
```

ionos_sd_config

Конфигурации IONOS SD позволяют извлекать цели сбора метрик из API IONOS Cloud. Это обнаружение сервисов по умолчанию использует IP-адрес первого NIC, но это можно изменить с помощью релейблинга. Следующие мета лейблы доступны на всех целях во время релейблинга:

- `__meta_ionos_server_availability_zone` : зона доступности сервера
- `__meta_ionos_server_boot_cdrom_id` : идентификатор CD-ROM, с которого загружен сервер
- `__meta_ionos_server_boot_image_id` : идентификатор загрузочного образа или снимка, с которого загружен сервер
- `__meta_ionos_server_boot_volume_id` : идентификатор загрузочного тома
- `__meta_ionos_server_cpu_family` : семейство процессоров сервера
- `__meta_ionos_server_id` : идентификатор сервера
- `__meta_ionos_server_ip` : список всех IP-адресов, назначенных серверу, разделенных запятыми
- `__meta_ionos_server_lifecycle` : состояние жизненного цикла ресурса сервера
- `__meta_ionos_server_name` : имя сервера
- `__meta_ionos_server_nic_ip_<nic_name>` : список IP-адресов, сгруппированных по имени каждого NIC, подключенного к серверу
- `__meta_ionos_server_servers_id` : идентификатор серверов, к которым принадлежит сервер
- `__meta_ionos_server_state` : состояние выполнения сервера
- `__meta_ionos_server_type` : тип сервера

```
# Уникальный идентификатор датацентра.
datacenter_id: <string>

# Порт для сбора метрик.
[ port: <int> | default = 80 ]

# Время, после которого серверы обновляются.
[ refresh_interval: <duration> | default = 60s ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и авторизация), конфигурации прокси, параметры TLS, пользовательские
HTTP-заголовки и т.д.
[ <http_config> ]
```

kubernetes_sd_config

Конфигурации Kubernetes SD позволяют извлекать цели сбора метрик из REST API Kubernetes и всегда оставаться синхронизированными с состоянием кластера.

Один из следующих типов `role` может быть настроен для обнаружения целей:

`node`

Роль `node` обнаруживает одну цель на каждый узел кластера с адресом, по умолчанию установленным на HTTP-порт Kubelet. Адрес цели по умолчанию устанавливается на первый существующий адрес объекта узла Kubernetes в порядке типов адресов `NodeInternalIP`, `NodeExternalIP`, `NodeLegacyHostIP` и `NodeHostName`.

Доступные мета лейблы:

- `__meta_kubernetes_node_name`: Имя объекта узла.
- `__meta_kubernetes_node_provider_id`: Имя облачного провайдера для объекта узла.
- `__meta_kubernetes_node_label_<labelname>`: Каждый лейбл из объекта узла, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_node_labelpresent_<labelname>`: `true` для каждого лейбла из объекта узла, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_node_annotation_<annotationname>`: Каждая аннотация из объекта узла.
- `__meta_kubernetes_node_annotationpresent_<annotationname>`: `true` для каждой аннотации из объекта узла.
- `__meta_kubernetes_node_address_<address_type>`: Первый адрес для каждого типа адреса узла, если он существует.

Кроме того, лейбл `instance` для узла будет установлен на имя узла, как оно получено с сервера API.

`service`

Роль `service` обнаруживает цель для каждого порта сервиса для каждого сервиса. Это обычно полезно для мониторинга сервиса в черном ящике. Адрес будет установлен на DNS-имя Kubernetes сервиса и соответствующий порт сервиса.

Доступные мета лейблы:

- `__meta_kubernetes_namespace` : Пространство имен объекта сервиса.
- `__meta_kubernetes_service_annotation_<annotationname>` : Каждая аннотация из объекта сервиса.
- `__meta_kubernetes_service_annotationpresent_<annotationname>` : “true” для каждой аннотации объекта сервиса.
- `__meta_kubernetes_service_cluster_ip` : IP-адрес кластера сервиса. (Не применяется к сервисам типа ExternalName)
- `__meta_kubernetes_service_loadbalancer_ip` : IP-адрес балансировщика нагрузки. (Применяется к сервисам типа LoadBalancer)
- `__meta_kubernetes_service_external_name` : DNS-имя сервиса. (Применяется к сервисам типа ExternalName)
- `__meta_kubernetes_service_label_<labelname>` : Каждый лейбл из объекта сервиса, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_service_labelpresent_<labelname>` : `true` для каждого лейбла объекта сервиса, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_service_name` : Имя объекта сервиса.
- `__meta_kubernetes_service_port_name` : Имя порта сервиса для цели.
- `__meta_kubernetes_service_port_number` : Номер порта сервиса для цели.
- `__meta_kubernetes_service_port_protocol` : Протокол порта сервиса для цели.
- `__meta_kubernetes_service_type` : Тип сервиса.

`pod`

Роль `pod` обнаруживает все поды и выставляет их контейнеры как цели. Для каждого объявленного порта контейнера создается одна цель. Если у контейнера нет указанных портов, создается цель без порта для каждого контейнера для ручного добавления порта через релейблинг.

Доступные мета лейблы:

- `__meta_kubernetes_namespace` : Пространство имен объекта пода.
- `__meta_kubernetes_pod_name` : Имя объекта пода.
- `__meta_kubernetes_pod_ip` : IP-адрес пода объекта пода.
- `__meta_kubernetes_pod_label_<labelname>` : Каждый лейбл из объекта пода, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_pod_labelpresent_<labelname>` : `true` для каждого лейбла из объекта пода, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_pod_annotation_<annotationname>` : Каждая аннотация из объекта пода.

- `__meta_kubernetes_pod_annotationpresent_<annotationname>`: `true` для каждой аннотации из объекта пода.
- `__meta_kubernetes_pod_container_init`: `true`, если контейнер является InitContainer
- `__meta_kubernetes_pod_container_name`: Имя контейнера, на который указывает адрес цели.
- `__meta_kubernetes_pod_container_id`: Идентификатор контейнера, на который указывает адрес цели. Идентификатор имеет форму `<type>://<container_id>`.
- `__meta_kubernetes_pod_container_image`: Образ, который использует контейнер.
- `__meta_kubernetes_pod_container_port_name`: Имя порта контейнера.
- `__meta_kubernetes_pod_container_port_number`: Номер порта контейнера.
- `__meta_kubernetes_pod_container_port_protocol`: Протокол порта контейнера.
- `__meta_kubernetes_pod_ready`: Установлено в `true` или `false` для состояния готовности пода.
- `__meta_kubernetes_pod_phase`: Установлено в `Pending`, `Running`, `Succeeded`, `Failed` или `Unknown` в жизненном цикле.
- `__meta_kubernetes_pod_node_name`: Имя узла, на который запланирован под.
- `__meta_kubernetes_pod_host_ip`: Текущий IP-адрес хоста объекта пода.
- `__meta_kubernetes_pod_uid`: UID объекта пода.
- `__meta_kubernetes_pod_controller_kind`: Тип объекта контроллера пода.
- `__meta_kubernetes_pod_controller_name`: Имя контроллера пода.

`endpoints`

Роль `endpoints` обнаруживает цели из перечисленных конечных точек сервиса. Для каждого адреса конечной точки обнаруживается одна цель на порт. Если конечная точка поддерживается подом, все дополнительные порты контейнера пода, не привязанные к порту конечной точки, также обнаруживаются как цели.

Доступные мета лейблы:

- `__meta_kubernetes_namespace`: Пространство имен объекта конечных точек.
- `__meta_kubernetes_endpoints_name`: Имена объекта конечных точек.
- `__meta_kubernetes_endpoints_label_<labelname>`: Каждый лейбл из объекта конечных точек, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_endpoints_labelpresent_<labelname>`: `true` для каждого лейбла из объекта конечных точек, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_endpoints_annotation_<annotationname>`: Каждая аннотация из объекта конечных точек.
- `__meta_kubernetes_endpoints_annotationpresent_<annotationname>`: `true` для каждой аннотации из объекта конечных точек.
- Для всех целей, обнаруженных непосредственно из списка конечных точек (тех, которые не дополнительно выведены из подов), прикреплены следующие лейблы:
  - `__meta_kubernetes_endpoint_hostname`: Имя хоста конечной точки.

    - `__meta_kubernetes_endpoint_node_name`: Имя узла, на котором размещена конечная точка.
    - `__meta_kubernetes_endpoint_ready`: Установлено в `true` или `false` для состояния готовности конечной точки.
    - `__meta_kubernetes_endpoint_port_name`: Имя порта конечной точки.
    - `__meta_kubernetes_endpoint_port_protocol`: Протокол порта конечной точки.
    - `__meta_kubernetes_endpoint_address_target_kind`: Тип целевого адреса конечной точки.
    - `__meta_kubernetes_endpoint_address_target_name`: Имя целевого адреса конечной точки.
- Если конечные точки принадлежат сервису, все лейблы обнаружения `role: service` прикреплены.
- Для всех целей, поддерживаемых подом, все лейблы обнаружения `role: pod` прикреплены.

`endpointslice`

Роль `endpointslice` обнаруживает цели из существующих срезов конечных точек. Для каждого адреса конечной точки, указанного в объекте среза конечных точек, обнаруживается одна цель. Если конечная точка поддерживается подом, все дополнительные порты контейнера пода, не привязанные к порту конечной точки, также обнаруживаются как цели.

Роль требует версии API `discovery.k8s.io/v1` (доступна с Kubernetes v1.21).

Доступные мета лейблы:

- `__meta_kubernetes_namespace`: Пространство имен объекта конечных точек.
- `__meta_kubernetes_endpointslice_name`: Имя объекта среза конечных точек.
- `__meta_kubernetes_endpointslice_label_<labelname>`: Каждый лейбл из объекта среза конечных точек, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_endpointslice_labelpresent_<labelname>`: `true` для каждого лейбла из объекта среза конечных точек, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_endpointslice_annotation_<annotationname>`: Каждая аннотация из объекта среза конечных точек.
- `__meta_kubernetes_endpointslice_annotationpresent_<annotationname>`: `true` для каждой аннотации из объекта среза конечных точек.
- Для всех целей, обнаруженных непосредственно из списка срезов конечных точек (тех, которые не дополнительно выведены из подов), прикреплены следующие лейблы:
    - `__meta_kubernetes_endpointslice_address_target_kind`: Тип указанного объекта.
    - `__meta_kubernetes_endpointslice_address_target_name`: Имя указанного объекта.
    - `__meta_kubernetes_endpointslice_address_type`: Семейство IP-протоколов адреса цели.

    - `__meta_kubernetes_endpointslice_endpoint_conditions_ready` : Установлено в `true` или `false` для состояния готовности указанной конечной точки.
    - `__meta_kubernetes_endpointslice_endpoint_conditions_serving` : Установлено в `true` или `false` для состояния обслуживания указанной конечной точки.
    - `__meta_kubernetes_endpointslice_endpoint_conditions_terminating` : Установлено в `true` или `false` для состояния завершения указанной конечной точки.
    - `__meta_kubernetes_endpointslice_endpoint_topology_kubernetes_io_hostname` : Имя узла, на котором размещена указанная конечная точка.
    - `__meta_kubernetes_endpointslice_endpoint_topology_present_kubernetes_io_hostname` : Флаг, показывающий, есть ли у указанного объекта аннотация kubernetes.io/hostname.
    - `__meta_kubernetes_endpointslice_endpoint_hostname` : Имя хоста указанной конечной точки.
    - `__meta_kubernetes_endpointslice_endpoint_node_name` : Имя узла, на котором размещена указанная конечная точка.
    - `__meta_kubernetes_endpointslice_endpoint_zone` : Зона, в которой существует указанная конечная точка.
    - `__meta_kubernetes_endpointslice_port` : Порт указанной конечной точки.
    - `__meta_kubernetes_endpointslice_port_name` : Имя порта указанной конечной точки.
    - `__meta_kubernetes_endpointslice_port_protocol` : Протокол порта указанной конечной точки.

- Если конечные точки принадлежат сервису, все лейблы обнаружения `role: service` прикреплены.
- Для всех целей, поддерживаемых подом, все лейблы обнаружения `role: pod` прикреплены.

`ingress`

Роль `ingress` обнаруживает цель для каждого пути каждого ingress. Это обычно полезно для мониторинга ingress в режиме черного ящика. Адрес будет установлен на хост, указанный в спецификации ingress.

Роль требует версии API `networking.k8s.io/v1` (доступна с Kubernetes v1.19).

Доступные мета лейблы:

- `__meta_kubernetes_namespace` : Пространство имен объекта ingress.
- `__meta_kubernetes_ingress_name` : Имя объекта ingress.
- `__meta_kubernetes_ingress_label_<labelname>` : Каждый лейбл из объекта ingress, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_ingress_labelpresent_<labelname>` : `true` для каждого лейбла из объекта ingress, с любыми неподдерживаемыми символами, преобразованными в подчеркивание.
- `__meta_kubernetes_ingress_annotation_<annotationname>` : Каждая аннотация из объекта ingress.

- `__meta_kubernetes_ingress_annotationpresent_<annotationname>`: `true` для каждой аннотации из объекта ingress.
- `__meta_kubernetes_ingress_class_name`: Имя класса из спецификации ingress, если присутствует.
- `__meta_kubernetes_ingress_scheme`: Протокольная схема ingress, `https`, если установлена конфигурация TLS. По умолчанию `http`.
- `__meta_kubernetes_ingress_path`: Путь из спецификации ingress. По умолчанию `/`.

См. ниже для параметров конфигурации обнаружения Kubernetes:

```
# Информация для доступа к API Kubernetes.

# Адреса API сервера. Если оставить пустым, предполагается, что Deckhouse Prom++
работает внутри
# кластера и автоматически обнаружит API серверы, используя CA сертификат и файл
токена
# аутентификации пода по пути /var/run/secrets/kubernetes.io/serviceaccount/.
[ api_server: <host> ]

# Роль Kubernetes для сущностей, которые должны быть обнаружены.
# Одна из: endpoints, endpointslice, service, pod, node или ingress.
role: <string>

# Необязательный путь к файлу kubeconfig.
# Обратите внимание, что api_server и kube_config взаимоисключающие.
[ kubeconfig_file: <filename> ]

# Необязательное обнаружение пространств имен. Если не указано, используются все
пространства имен.
namespaces:
  own_namespace: <boolean>
  names:
    [ - <string> ]

# Необязательные селекторы лейблов и полей для ограничения процесса обнаружения до
подмножества доступных ресурсов.
# См. https://kubernetes.io/docs/concepts/overview/working-with-objects/field-
selectors/
# и https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/ для
получения дополнительной информации
# о возможных фильтрах, которые можно использовать. Роль endpoints поддерживает
селекторы pod, service и endpoints.
# Роль pod поддерживает селекторы узлов при настройке с `attach_metadata: {node: true}
`.
# Другие роли поддерживают только селекторы, соответствующие самой роли (например,
роль node может содержать только селекторы узлов).

# Примечание: При принятии решения об использовании селекторов полей/лейблов
убедитесь, что это
# наилучший подход - это предотвратит повторное использование Deckhouse Prom++ единого
списка/наблюдения
# для всех конфигураций сбора. Это может привести к увеличению нагрузки на API
Kubernetes,
# так как для каждой комбинации селекторов будет дополнительный LIST/WATCH. С другой
стороны,
# если вы хотите мониторить небольшое подмножество подов в большом кластере,
рекомендуется использовать селекторы.
# Решение о том, следует ли использовать селекторы, зависит от конкретной ситуации.
```

```
[ selectors:
  [ - role: <string>
    [ label: <string> ]
    [ field: <string> ] ]]

# Необязательные метаданные для прикрепления к обнаруженным целям. Если не указано,
дополнительные метаданные не прикрепляются.
attach_metadata:
# Прикрепляет метаданные узла к обнаруженным целям. Действительно для ролей: pod,
endpoints, endpointslice.
# Если установлено в true, Deckhouse Prom++ должен иметь разрешения на получение
узлов.
  [ node: <boolean> | default = false ]

# Настройки HTTP клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP заголовки и
т.д.
[ <http_config> ]
```

См. этот пример конфигурационного файла Deckhouse Prom++ для подробного примера настройки Deckhouse Prom++ для Kubernetes.

Вы можете ознакомиться с 3-й стороной Prometheus Operator, который автоматизирует настройку Deckhouse Prom++ поверх Kubernetes.

kuma_sd_config

Конфигурации SD для Kuma позволяют получать цели сбора из контрольной плоскости Kuma.

Этот SD обнаруживает “задания мониторинга” на основе Dataplane Proxies Kuma, через MADS v1 (Monitoring Assignment Discovery Service) xDS API, и создаст цель для каждого прокси внутри сети, поддерживаемой Deckhouse Prom++.

Следующие мета лейблы доступны для каждой цели:

- `__meta_kuma_mesh` : имя сети прокси
- `__meta_kuma_dataplane` : имя прокси
- `__meta_kuma_service` : имя связанного сервиса прокси
- `__meta_kuma_label_<tagname>` : каждый тег прокси

См. ниже для параметров конфигурации обнаружения MonitoringAssignment для Kuma:

```
# Адрес MADS xDS сервера контрольной плоскости Kuma.
server: <string>

# Идентификатор клиента используется контрольной плоскостью Kuma для вычисления
задания мониторинга для конкретного бэкенда Deckhouse Prom++.
# Это полезно при миграции между несколькими бэкендами Deckhouse Prom++, или при
наличии отдельного бэкенда для каждой сети.
# Если не указано, будет использоваться системное имя хоста/FQDN, если доступно, в
противном случае будет использоваться `prometheus`.
[ client_id: <string> ]

# Время ожидания между запросами обновления.
[ refresh_interval: <duration> | default = 30s ]

# Время, после которого задания мониторинга обновляются.
[ fetch_timeout: <duration> | default = 2m ]

# Настройки HTTP клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP заголовки и
т.д.
[ <http_config> ]
```

Фаза переназначения является предпочтительным и более мощным способом фильтрации прокси и пользовательских тегов.

lightsail_sd_config

Конфигурации SD для Lightsail позволяют получать цели сбора с инстансов AWS Lightsail. По умолчанию используется частный IP-адрес, но он может быть изменен на публичный IP-адрес с помощью переназначения.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_lightsail_availability_zone` : зона доступности, в которой работает инстанс
- `__meta_lightsail_blueprint_id` : ID шаблона Lightsail
- `__meta_lightsail_bundle_id` : ID пакета Lightsail
- `__meta_lightsail_instance_name` : имя инстанса Lightsail
- `__meta_lightsail_instance_state` : состояние инстанса Lightsail
- `__meta_lightsail_instance_support_code` : код поддержки инстанса Lightsail
- `__meta_lightsail_ipv6_addresses` : список IPv6 адресов, назначенных сетевым интерфейсам инстанса, через запятую, если присутствует
- `__meta_lightsail_private_ip` : частный IP-адрес инстанса
- `__meta_lightsail_public_ip` : публичный IP-адрес инстанса, если доступен
- `__meta_lightsail_region` : регион инстанса
- `__meta_lightsail_tag_<tagkey>` : каждое значение тега инстанса

См. ниже для параметров конфигурации обнаружения Lightsail:

```
# Информация для доступа к API Lightsail.

# Регион AWS. Если пусто, используется регион из метаданных инстанса.
[ region: <string> ]

# Пользовательская конечная точка для использования.
[ endpoint: <string> ]

# Ключи API AWS. Если пусто, используются переменные окружения `AWS_ACCESS_KEY_ID`
# и `AWS_SECRET_ACCESS_KEY`.
[ access_key: <string> ]
[ secret_key: <secret> ]
# Имя профиля AWS, используемого для подключения к API.
[ profile: <string> ]

# ARN роли AWS, альтернатива использованию ключей API AWS.
[ role_arn: <string> ]

# Интервал обновления для повторного чтения списка инстансов.
[ refresh_interval: <duration> | default = 60s ]

# Порт для сбора метрик. Если используется публичный IP-адрес, это должно
# быть указано в правиле переназначения.
[ port: <int> | default = 80 ]

# Настройки HTTP клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP заголовки и
т.д.
[ <http_config> ]
```

linode_sd_config

Конфигурации SD для Linode позволяют получать цели сбора из Linode’s Linode APIv4. Эта служба обнаружения по умолчанию использует публичный IPv4 адрес, но это может быть изменено с помощью переназначения, как показано в конфигурационном файле linode-sd для Deckhouse Prom++.

Токен APIv4 Linode должен быть создан с правами: `linodes:read_only`, `ips:read_only`, и `events:read_only`.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_linode_instance_id`: ID инстанса linode
- `__meta_linode_instance_label`: лейбл инстанса linode
- `__meta_linode_image`: идентификатор образа инстанса linode

- `__meta_linode_private_ipv4` : частный IPv4 инстанса linode
- `__meta_linode_public_ipv4` : публичный IPv4 инстанса linode
- `__meta_linode_public_ipv6` : публичный IPv6 инстанса linode
- `__meta_linode_private_ipv4_rdns` : обратный DNS для первого частного IPv4 инстанса linode
- `__meta_linode_public_ipv4_rdns` : обратный DNS для первого публичного IPv4 инстанса linode
- `__meta_linode_public_ipv6_rdns` : обратный DNS для первого публичного IPv6 инстанса linode
- `__meta_linode_region` : регион инстанса linode
- `__meta_linode_type` : тип инстанса linode
- `__meta_linode_status` : статус инстанса linode
- `__meta_linode_tags` : список тегов инстанса linode, объединенных через разделитель тегов
- `__meta_linode_group` : группа отображения, членом которой является инстанс linode
- `__meta_linode_gpus` : количество GPU инстанса linode
- `__meta_linode_hypervisor` : программное обеспечение виртуализации, управляющее инстансом linode
- `__meta_linode_backups` : статус службы резервного копирования инстанса linode
- `__meta_linode_specs_disk_bytes` : объем дискового пространства, доступного инстансу linode
- `__meta_linode_specs_memory_bytes` : объем оперативной памяти, доступной инстансу linode
- `__meta_linode_specs_vcpus` : количество VCPU, доступных этому linode
- `__meta_linode_specs_transfer_bytes` : объем сетевого трафика, выделенного инстансу linode каждый месяц
- `__meta_linode_extra_ips` : список всех дополнительных IPv4 адресов, назначенных инстансу linode, объединенных через разделитель тегов
- `__meta_linode_ipv6_ranges` : список диапазонов IPv6 с маской, назначенных инстансу linode, объединенных через разделитель тегов

```
# Необязательный регион для фильтрации.
[ region: <string> ]

# Порт для сбора метрик.
[ port: <int> | default = 80 ]

# Строка, по которой теги инстанса Linode объединяются в лейбл тегов.
[ tag_separator: <string> | default = , ]

# Время, после которого инстансы linode обновляются.
[ refresh_interval: <duration> | default = 60s ]

# Настройки HTTP клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP заголовки и
т.д.
[ <http_config> ]
```

marathon_sd_config

Конфигурации SD для Marathon позволяют получать цели сбора с использованием REST API Marathon. Deckhouse Prom++ будет периодически проверять конечную точку REST для текущих задач и создавать группу целей для каждого приложения, у которого есть хотя бы одна здоровая задача.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_marathon_app` : имя приложения (с заменой слэшей на дефисы)
- `__meta_marathon_image` : имя используемого Docker образа (если доступно)
- `__meta_marathon_task` : ID задачи Mesos
- `__meta_marathon_app_label_<labelname>` : любые лейблы Marathon, прикрепленные к приложению, с любыми неподдерживаемыми символами, преобразованными в подчеркивание
- `__meta_marathon_port_definition_label_<labelname>` : лейблы определения порта, с любыми неподдерживаемыми символами, преобразованными в подчеркивание
- `__meta_marathon_port_mapping_label_<labelname>` : лейблы отображения порта, с любыми неподдерживаемыми символами, преобразованными в подчеркивание
- `__meta_marathon_port_index` : номер индекса порта (например, `1` для `PORT1`)

См. ниже для параметров конфигурации обнаружения Marathon:

```
# Список URL-адресов для связи с серверами Marathon.
# Необходимо указать хотя бы один URL сервера.
servers:
  - <string>

# Интервал опроса
[ refresh_interval: <duration> | default = 30s ]

# Необязательная информация для аутентификации на основе токенов
# https://docs.mesosphere.com/1.11/security/ent/iam-api/#passing-an-authentication-
token
# Взаимоисключается с `auth_token_file` и другими механизмами аутентификации.
[ auth_token: <secret> ]

# Необязательная информация для аутентификации на основе токенов
# https://docs.mesosphere.com/1.11/security/ent/iam-api/#passing-an-authentication-
token
# Взаимоисключается с `auth_token` и другими механизмами аутентификации.
[ auth_token_file: <filename> ]

# Настройки HTTP клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP заголовки и
т.д.
[ <http_config> ]
```

По умолчанию каждое приложение, перечисленное в Marathon, будет собираться Deckhouse Prom++. Если не все ваши сервисы предоставляют метрики Deckhouse Prom++, вы можете использовать лейбл Marathon и переназначение Deckhouse Prom++, чтобы контролировать, какие инстансы будут фактически собираться. См. конфигурационный файл marathon-sd для Deckhouse Prom++ для практического примера настройки вашего приложения Marathon и конфигурации Deckhouse Prom++.

По умолчанию все приложения будут отображаться как одна задача в Deckhouse Prom++ (та, которая указана в конфигурационном файле), что также можно изменить с помощью переназначения.

nerve_sd_config

Конфигурации SD для Nerve позволяют получать цели сбора из [Nerve от AirBnB] (https://github.com/airbnb/nerve), которые хранятся в Zookeeper.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_nerve_path` : полный путь к узлу конечной точки в Zookeeper
- `__meta_nerve_endpoint_host` : хост конечной точки
- `__meta_nerve_endpoint_port` : порт конечной точки

- `__meta_nerve_endpoint_name` : имя конечной точки

```
# Сервера Zookeeper.
servers:
  - <host>
# Пути могут указывать на одну службу или корень дерева служб.
paths:
  - <string>
[ timeout: <duration> | default = 10s ]
```

nomad_sd_config

Конфигурации SD для Nomad позволяют получать цели сбора из API сервиса Nomad.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_nomad_address` : адрес сервиса цели
- `__meta_nomad_dc` : имя датацентра для цели
- `__meta_nomad_namespace` : пространство имен цели
- `__meta_nomad_node_id` : имя узла, определенное для цели
- `__meta_nomad_service` : имя сервиса, к которому принадлежит цель
- `__meta_nomad_service_address` : адрес сервиса цели
- `__meta_nomad_service_id` : ID сервиса цели
- `__meta_nomad_service_port` : порт сервиса цели
- `__meta_nomad_tags` : список тегов цели, объединенных через разделитель тегов

```
# Информация для доступа к API Nomad. Она должна быть определена
# в соответствии с требованиями документации Nomad.
[ allow_stale: <boolean> | default = true ]
[ namespace: <string> | default = default ]
[ refresh_interval: <duration> | default = 60s ]
[ region: <string> | default = global ]
# URL для подключения к API.
[ server: <string> ]
[ tag_separator: <string> | default = ,]

# Настройки HTTP клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP заголовки и
т.д.
[ <http_config> ]
```

serverset_sd_config

Конфигурации SD для Serverset позволяют получать цели сбора из [Serversets] (https://github.com/twitter/finagle/tree/develop/finagle-serversets), которые хранятся в Zookeeper. Serversets обычно используются Finagle и Aurora.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_serverset_path`: полный путь к узлу члена serverset в Zookeeper
- `__meta_serverset_endpoint_host`: хост конечной точки по умолчанию
- `__meta_serverset_endpoint_port`: порт конечной точки по умолчанию
- `__meta_serverset_endpoint_host_<endpoint>`: хост указанной конечной точки
- `__meta_serverset_endpoint_port_<endpoint>`: порт указанной конечной точки
- `__meta_serverset_shard`: номер шардирования члена
- `__meta_serverset_status`: статус члена

```
# Сервера Zookeeper.
servers:
  - <host>
# Пути могут указывать на один serverset или корень дерева serverset.
paths:
  - <string>
[ timeout: <duration> | default = 10s ]
```

Данные serverset должны быть в формате JSON, формат Thrift в настоящее время не поддерживается.

triton_sd_config

Конфигурации SD для Triton позволяют получать цели сбора из конечных точек обнаружения Container Monitor.

Один из следующих типов `<triton_role>` может быть настроен для обнаружения целей:

`container`

Роль `container` обнаруживает одну цель на каждую “виртуальную машину”, принадлежащую `account`. Это зоны SmartOS или зоны с брендом lx/KVM/bhyve.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_triton_groups`: список групп, к которым принадлежит цель, объединенных через запятую
- `__meta_triton_machine_alias`: псевдоним контейнера цели
- `__meta_triton_machine_brand`: бренд контейнера цели
- `__meta_triton_machine_id`: UUID контейнера цели
- `__meta_triton_machine_image`: тип образа контейнера цели
- `__meta_triton_server_id`: UUID сервера, на котором работает контейнер цели

`cn`

Роль `cn` обнаруживает одну цель на каждый вычислительный узел (также известный как “сервер” или “глобальная зона”), составляющий инфраструктуру Triton. `account` должен быть оператором Triton и в настоящее время требуется владеть хотя бы одним `container`.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_triton_machine_alias` : имя хоста цели (требуется triton-cmon 1.7.0 или новее)
- `__meta_triton_machine_id` : UUID цели

См. ниже для параметров конфигурации обнаружения Triton:

```
# Информация для доступа к API обнаружения Triton.

# Учетная запись для использования при обнаружении новых целей.
account: <string>

# Тип целей для обнаружения, может быть установлен в:
# * "container" для обнаружения виртуальных машин (зоны SmartOS, зоны с брендом lx/
KVM/bhyve), работающих на Triton
# * "cn" для обнаружения вычислительных узлов (серверов/глобальных зон), составляющих
инфраструктуру Triton
[ role : <string> | default = "container" ]

# DNS-суффикс, который должен быть применен к цели.
dns_suffix: <string>

# Конечная точка обнаружения Triton (например, 'cmon.us-east-3b.triton.zone'). Это
# часто то же значение, что и dns_suffix.
endpoint: <string>

# Список групп, для которых извлекаются цели, поддерживается только при `role` ==
`container`.
# Если не указано, все контейнеры, принадлежащие запрашивающей учетной записи, будут
собраны.
groups:
  [ - <string> ... ]

# Порт для использования при обнаружении и сборе метрик.
[ port: <int> | default = 9163 ]

# Интервал, который должен использоваться для обновления целей.
[ refresh_interval: <duration> | default = 60s ]

# Версия API обнаружения Triton.
[ version: <int> | default = 1 ]

# Конфигурация TLS.
tls_config:
  [ <tls_config> ]
```

eureka_sd_config

Конфигурации Eureka SD позволяют получать цели для сбора данных с использованием Eureka REST API. Deckhouse Prom++ будет периодически проверять REST-эндпоинт и создавать цель для каждого экземпляра приложения.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_eureka_app_name` : имя приложения
- `__meta_eureka_app_instance_id` : ID экземпляра приложения
- `__meta_eureka_app_instance_hostname` : имя хоста экземпляра
- `__meta_eureka_app_instance_homepage_url` : URL домашней страницы экземпляра приложения
- `__meta_eureka_app_instance_statuspage_url` : URL страницы статуса экземпляра приложения
- `__meta_eureka_app_instance_healthcheck_url` : URL проверки состояния экземпляра приложения
- `__meta_eureka_app_instance_ip_addr` : IP-адрес экземпляра приложения
- `__meta_eureka_app_instance_vip_address` : VIP-адрес экземпляра приложения
- `__meta_eureka_app_instance_secure_vip_address` : защищенный VIP-адрес экземпляра приложения
- `__meta_eureka_app_instance_status` : статус экземпляра приложения
- `__meta_eureka_app_instance_port` : порт экземпляра приложения
- `__meta_eureka_app_instance_port_enabled` : порт включен для экземпляра приложения
- `__meta_eureka_app_instance_secure_port` : защищенный порт экземпляра приложения
- `__meta_eureka_app_instance_secure_port_enabled` : защищенный порт экземпляра приложения включен
- `__meta_eureka_app_instance_country_id` : ID страны экземпляра приложения
- `__meta_eureka_app_instance_metadata_<metadataname>` : метаданные экземпляра приложения
- `__meta_eureka_app_instance_datacenterinfo_name` : имя датацентра экземпляра приложения
- `__meta_eureka_app_instance_datacenterinfo_<metadataname>` : метаданные датацентра

См. ниже параметры конфигурации для обнаружения Eureka:

```
# URL для подключения к серверу Eureka.
server: <string>

# Интервал обновления для повторного чтения списка экземпляров приложения.
[ refresh_interval: <duration> | default = 30s ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]
```

См. файл конфигурации eureka-sd для Deckhouse Prom++ для практического примера настройки вашего приложения Eureka и конфигурации Deckhouse Prom++.

scaleway_sd_config

Конфигурации Scaleway SD позволяют получать цели для сбора данных с инстансов Scaleway и услуг baremetal.

Следующие мета лейблы доступны на целях во время переназначения:

Роль инстанса

- `__meta_scaleway_instance_boot_type` : тип загрузки сервера
- `__meta_scaleway_instance_hostname` : имя хоста сервера
- `__meta_scaleway_instance_id` : ID сервера
- `__meta_scaleway_instance_image_arch` : архитектура образа сервера
- `__meta_scaleway_instance_image_id` : ID образа сервера
- `__meta_scaleway_instance_image_name` : имя образа сервера
- `__meta_scaleway_instance_location_cluster_id` : ID кластера местоположения сервера
- `__meta_scaleway_instance_location_hypervisor_id` : ID гипервизора местоположения сервера
- `__meta_scaleway_instance_location_node_id` : ID узла местоположения сервера
- `__meta_scaleway_instance_name` : имя сервера
- `__meta_scaleway_instance_organization_id` : организация сервера
- `__meta_scaleway_instance_private_ipv4` : частный IPv4-адрес сервера
- `__meta_scaleway_instance_project_id` : ID проекта сервера
- `__meta_scaleway_instance_public_ipv4` : публичный IPv4-адрес сервера
- `__meta_scaleway_instance_public_ipv6` : публичный IPv6-адрес сервера
- `__meta_scaleway_instance_public_ipv4_addresses` : публичные IPv4-адреса сервера
- `__meta_scaleway_instance_public_ipv6_addresses` : публичные IPv6-адреса сервера
- `__meta_scaleway_instance_region` : регион сервера
- `__meta_scaleway_instance_security_group_id` : ID группы безопасности сервера
- `__meta_scaleway_instance_security_group_name` : имя группы безопасности сервера

- `__meta_scaleway_instance_status` : статус сервера
- `__meta_scaleway_instance_tags` : список тегов сервера, объединенных разделителем тегов
- `__meta_scaleway_instance_type` : коммерческий тип сервера
- `__meta_scaleway_instance_zone` : зона сервера (например, `fr-par-1` , полный список здесь)

Эта роль использует первый найденный адрес в следующем порядке: частный IPv4, публичный IPv4, публичный IPv6. Это можно изменить с помощью переназначения, как показано в файле конфигурации scaleway-sd для Deckhouse Prom++. Если у инстанса нет адреса до переназначения, он не будет добавлен в список целей, и вы не сможете его переназначить.

Роль baremetal

- `__meta_scaleway_baremetal_id` : ID сервера
- `__meta_scaleway_baremetal_public_ipv4` : публичный IPv4-адрес сервера
- `__meta_scaleway_baremetal_public_ipv6` : публичный IPv6-адрес сервера
- `__meta_scaleway_baremetal_name` : имя сервера
- `__meta_scaleway_baremetal_os_name` : имя операционной системы сервера
- `__meta_scaleway_baremetal_os_version` : версия операционной системы сервера
- `__meta_scaleway_baremetal_project_id` : ID проекта сервера
- `__meta_scaleway_baremetal_status` : статус сервера
- `__meta_scaleway_baremetal_tags` : список тегов сервера, объединенных разделителем тегов
- `__meta_scaleway_baremetal_type` : коммерческий тип сервера
- `__meta_scaleway_baremetal_zone` : зона сервера (например, `fr-par-1` , полный список здесь)

Эта роль по умолчанию использует публичный IPv4-адрес. Это можно изменить с помощью переназначения, как показано в файле конфигурации scaleway-sd для Deckhouse Prom++.

См. ниже параметры конфигурации для обнаружения Scaleway:

```
# Ключ доступа для использования. https://console.scaleway.com/project/credentials
access_key: <string>

# Секретный ключ для использования при перечислении целей. https://
console.scaleway.com/project/credentials
# Он взаимоисключается с `secret_key_file`.
[ secret_key: <secret> ]

# Устанавливает секретный ключ с учетными данными, прочитанными из настроенного файла.
# Он взаимоисключается с `secret_key`.
[ secret_key_file: <filename> ]

# ID проекта целей.
project_id: <string>

# Роль целей для получения. Должна быть `instance` или `baremetal`.
role: <string>

# Порт для сбора метрик.
[ port: <int> | default = 80 ]

# URL API для использования при выполнении запросов на перечисление серверов.
[ api_url: <string> | default = "https://api.scaleway.com" ]

# Зона - это зона доступности ваших целей (например, fr-par-1).
[ zone: <string> | default = fr-par-1 ]

# NameFilter указывает фильтр имени (работает как LIKE) для применения к запросу на
перечисление серверов.
[ name_filter: <string> ]

# TagsFilter указывает фильтр тегов (сервер должен иметь все определенные теги, чтобы
быть перечисленным) для применения к запросу на перечисление серверов.
tags_filter:
[ - <string> ]

# Интервал обновления для повторного чтения списка целей.
[ refresh_interval: <duration> | default = 60s ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]
```

uyuni_sd_config

Конфигурации Uyuni SD позволяют получать цели для сбора данных с управляемых систем через Uyuni API.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_uyuni_endpoint_name` : имя конечной точки приложения
- `__meta_uyuni_exporter` : экспортер, предоставляющий метрики для цели
- `__meta_uyuni_groups` : системные группы цели
- `__meta_uyuni_metrics_path` : путь метрик для цели
- `__meta_uyuni_minion_hostname` : имя хоста клиента Uyuni
- `__meta_uyuni_primary_fqdn` : основной FQDN клиента Uyuni
- `__meta_uyuni_proxy_module` : имя модуля, если настроен прокси *Exporter Exporter* для цели
- `__meta_uyuni_scheme` : протокольная схема, используемая для запросов
- `__meta_uyuni_system_id` : ID системы клиента

См. ниже параметры конфигурации для обнаружения Uyuni:

```
# URL для подключения к серверу Uyuni.
server: <string>

# Учетные данные используются для аутентификации запросов к Uyuni API.
username: <string>
password: <secret>

# Строка прав для фильтрации подходящих систем.
[ entitlement: <string> | default = monitoring_entitled ]

# Строка, по которой имена групп Uyuni объединяются в лейбл групп.
[ separator: <string> | default = , ]

# Интервал обновления для повторного чтения списка управляемых целей.
[ refresh_interval: <duration> | default = 60s ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]
```

См. файл конфигурации uyuni-sd для Deckhouse Prom++ для практического примера настройки конфигурации Uyuni Deckhouse Prom++.

vultr_sd_config

Конфигурации Vultr SD позволяют получать цели для сбора данных с Vultr.

Это обнаружение сервисов по умолчанию использует основной IPv4-адрес, который можно изменить с помощью переназначения, как показано в файле конфигурации vultr-sd для Deckhouse Prom++.

Следующие мета лейблы доступны на целях во время переназначения:

- `__meta_vultr_instance_id` : Уникальный ID для инстанса Vultr.
- `__meta_vultr_instance_label` : лейбл, предоставленный пользователем для этого инстанса.
- `__meta_vultr_instance_os` : Имя операционной системы.
- `__meta_vultr_instance_os_id` : ID операционной системы, используемой этим инстансом.
- `__meta_vultr_instance_region` : ID региона, где расположен инстанс.
- `__meta_vultr_instance_plan` : Уникальный ID для плана.
- `__meta_vultr_instance_main_ip` : Основной IPv4-адрес.
- `__meta_vultr_instance_internal_ip` : Частный IP-адрес.
- `__meta_vultr_instance_main_ipv6` : Основной IPv6-адрес.
- `__meta_vultr_instance_features` : Список функций, доступных для инстанса.
- `__meta_vultr_instance_tags` : Список тегов, связанных с инстансом.
- `__meta_vultr_instance_hostname` : Имя хоста для этого инстанса.
- `__meta_vultr_instance_server_status` : Статус здоровья сервера.
- `__meta_vultr_instance_vcpu_count` : Количество виртуальных процессоров.
- `__meta_vultr_instance_ram_mb` : Объем оперативной памяти в МБ.
- `__meta_vultr_instance_disk_gb` : Размер диска в ГБ.
- `__meta_vultr_instance_allowed_bandwidth_gb` : Ежемесячная квота на пропускную способность в ГБ.

```
# Порт для сбора метрик.
[ port: <int> | default = 80 ]

# Время, после которого инстансы обновляются.
[ refresh_interval: <duration> | default = 60s ]

# Настройки HTTP-клиента, включая методы аутентификации (такие как базовая
аутентификация и
# авторизация), конфигурации прокси, параметры TLS, пользовательские HTTP-заголовки и
т.д.
[ <http_config> ]
```

static_config

`static_config` позволяет указать список целей и общий набор лейблов для них. Это канонический способ указания статических целей в конфигурации сбора данных.

```
# Цели, указанные в статической конфигурации.
targets:
  [ - '<host>' ]

# лейблы, назначенные всем метрикам, собранным с целей.
labels:
  [ <labelname>: <labelvalue> ... ]
```

relabel_config

Переназначение - это мощный инструмент для динамического переписывания набора лейблов цели перед ее сбором. Несколько шагов переназначения могут быть настроены для каждой конфигурации сбора данных. Они применяются к набору лейблов каждой цели в порядке их появления в конфигурационном файле.

Изначально, помимо настроенных лейблов для каждой цели, лейбл `job` цели устанавливается в значение `job_name` соответствующей конфигурации сбора данных. лейбл `__address__` устанавливается в адрес `<host>:<port>` цели. После переназначения лейбл `instance` по умолчанию устанавливается в значение `__address__`, если он не был установлен во время переназначения.

лейблы `__scheme__` и `__metrics_path__` устанавливаются в схему и путь метрик цели соответственно, как указано в `scrape_config`.

лейбл `__param_<name>` устанавливается в значение первого переданного параметра URL с именем `<name>`, как определено в `scrape_config`.

лейблы `__scrape_interval__` и `__scrape_timeout__` устанавливаются в интервал и тайм-аут цели, как указано в `scrape_config`.

Дополнительные лейблы с префиксом `__meta_` могут быть доступны во время фазы переназначения. Они устанавливаются механизмом обнаружения сервисов, который предоставил цель, и варьируются в зависимости от механизмов.

лейблы, начинающиеся с `__`, будут удалены из набора лейблов после завершения переназначения цели.

Если шаг переназначения должен временно сохранить значение лейбла (как вход для последующего шага переназначения), используйте префикс имени лейбла `__tmp`. Этот префикс гарантированно никогда не будет использоваться самим Deckhouse Prom++.

```
# source_labels указывает правило, какие лейблы извлекать из серии. Любые
# лейблы, которые не существуют, получают пустое значение (""). Их содержимое
конкатенируется
# с использованием настроенного разделителя и сопоставляется с настроенным регулярным
выражением
# для действий замены, сохранения и удаления.
[ source_labels: '[' <labelname> [, ...] ']' ]

# Разделитель, помещаемый между конкатенированными значениями исходных лейблов.
[ separator: <string> | default = ; ]

# лейбл, в который записывается результирующее значение в действии замены.
# Он обязателен для действий замены. Доступны группы захвата регулярных выражений.
[ target_label: <labelname> ]

# Регулярное выражение, с которым сопоставляется извлеченное значение.
[ regex: <regex> | default = (.*) ]

# Модуль для взятия хеша значений исходных лейблов.
[ modulus: <int> ]

# Значение замены, с которым выполняется замена регулярного выражения, если
# регулярное выражение совпадает. Доступны группы захвата регулярных выражений.
[ replacement: <string> | default = $1 ]

# Действие, выполняемое на основе совпадения регулярного выражения.
[ action: <relabel_action> | default = replace ]
```

`<regex>` - это любое допустимое регулярное выражение RE2. Оно требуется для действий `replace`, `keep`, `drop`, `labelmap`, `labeldrop` и `labelkeep`. Регулярное выражение закреплено с обеих сторон. Чтобы снять закрепление, используйте `.*<regex>.*`.

`<relabel_action>` определяет действие переназначения, которое нужно выполнить:

- `replace`: Сопоставить `regex` с конкатенированными `source_labels`. Затем установить `target_label` в `replacement`, с подстановкой ссылок на группы захвата (`${1}`, `${2}`, ...) в `replacement` их значениями. Если `regex` не совпадает, замена не выполняется.
- `lowercase`: Преобразовать конкатенированные `source_labels` в нижний регистр.
- `uppercase`: Преобразовать конкатенированные `source_labels` в верхний регистр.
- `keep`: Удалить цели, для которых `regex` не совпадает с конкатенированными `source_labels`.
- `drop`: Удалить цели, для которых `regex` совпадает с конкатенированными `source_labels`.
- `keepequal`: Удалить цели, для которых конкатенированные `source_labels` не совпадают с `target_label`.
- `dropequal`: Удалить цели, для которых конкатенированные `source_labels` совпадают с `target_label`.

- `hashmod`: Установить `target_label` в `modulus` хеша конкатенированных `source_labels`.
- `labelmap`: Сопоставить `regex` со всеми именами исходных лейблов, а не только с указанными в `source_labels`. Затем скопировать значения совпадающих лейблов в имена лейблов, указанные в `replacement`, с подстановкой ссылок на группы захвата (`${1}`, `${2}`, …) в `replacement` их значениями.
- `labeldrop`: Сопоставить `regex` со всеми именами лейблов. Любой лейбл, который совпадает, будет удален из набора лейблов.
- `labelkeep`: Сопоставить `regex` со всеми именами лейблов. Любой лейбл, который не совпадает, будет удален из набора лейблов.

Необходимо соблюдать осторожность с `labeldrop` и `labelkeep`, чтобы гарантировать, что метрики по-прежнему уникально маркированы после удаления лейблов.

metric_relabel_configs

Переназначение метрик применяется к образцам на последнем этапе перед их загрузкой. Оно имеет такой же формат конфигурации и действия, как и переназначение целей. Переназначение метрик не применяется к автоматически сгенерированным временным рядам, таким как `up`.

Одним из применений этого является исключение временных рядов, которые слишком дороги для загрузки.

alert_relabel_configs

Переназначение оповещений применяется к оповещениям перед их отправкой в Alertmanager. Оно имеет такой же формат конфигурации и действия, как и переназначение целей. Переназначение оповещений применяется после внешних лейблов.

Одним из применений этого является обеспечение того, чтобы пара серверов Deckhouse Prom++ с разными внешними лейблами отправляла идентичные оповещения.

alertmanager_config

Раздел `alertmanager_config` указывает экземпляры Alertmanager, которым сервер Deckhouse Prom++ отправляет оповещения. Он также предоставляет параметры для настройки связи с этими Alertmanager.

Alertmanager может быть статически настроен через параметр `static_configs` или динамически обнаружен с использованием одного из поддерживаемых механизмов обнаружения сервисов.

Кроме того, `relabel_configs` позволяет выбирать Alertmanager из обнаруженных сущностей и предоставляет расширенные модификации используемого API пути, который отображается через лейбл `__alerts_path__`.

```
# Тайм-аут для каждой цели Alertmanager при отправке оповещений.
[ timeout: <duration> | default = 10s ]

# Версия API Alertmanager.
[ api_version: <string> | default = v2 ]

# Префикс для HTTP пути, по которому отправляются оповещения.
[ path_prefix: <path> | default = / ]

# Настраивает схему протокола, используемую для запросов.
[ scheme: <scheme> | default = http ]

# Опционально настраивает процесс подписания запросов с использованием AWS Signature
Verification 4.
# Не может быть установлен одновременно с basic_auth, authorization, oauth2, azuread
или google_iam.
# Чтобы использовать учетные данные по умолчанию из AWS SDK, используйте `sigv4: {}`.
sigv4:
  # Регион AWS. Если пусто, используется регион из цепочки учетных данных по
умолчанию.
  [ region: <string> ]


# Ключи API AWS. Если пусто, используются переменные окружения `AWS_ACCESS_KEY_ID` и
`AWS_SECRET_ACCESS_KEY`.
  [ access_key: <string> ]
  [ secret_key: <secret> ]

  # Имя профиля AWS, используемого для аутентификации.
  [ profile: <string> ]

  # AWS Role ARN, альтернатива использованию ключей API AWS.
  [ role_arn: <string> ]

# Настройки HTTP клиента, включая методы аутентификации (такие как basic auth и
authorization), конфигурации прокси, параметры TLS, пользовательские HTTP заголовки и
т.д.
[ <http_config> ]

# Список конфигураций обнаружения сервисов Azure.
azure_sd_configs:
  [ - <azure_sd_config> ... ]

# Список конфигураций обнаружения сервисов Consul.
consul_sd_configs:
  [ - <consul_sd_config> ... ]

# Список конфигураций обнаружения сервисов DNS.
```

```
dns_sd_configs:
  [ - <dns_sd_config> ... ]

# Список конфигураций обнаружения сервисов EC2.
ec2_sd_configs:
  [ - <ec2_sd_config> ... ]

# Список конфигураций обнаружения сервисов Eureka.
eureka_sd_configs:
  [ - <eureka_sd_config> ... ]

# Список конфигураций обнаружения сервисов файлов.
file_sd_configs:
  [ - <file_sd_config> ... ]

# Список конфигураций обнаружения сервисов DigitalOcean.
digitalocean_sd_configs:
  [ - <digitalocean_sd_config> ... ]

# Список конфигураций обнаружения сервисов Docker.
docker_sd_configs:
  [ - <docker_sd_config> ... ]

# Список конфигураций обнаружения сервисов Docker Swarm.
dockerswarm_sd_configs:
  [ - <dockerswarm_sd_config> ... ]

# Список конфигураций обнаружения сервисов GCE.
gce_sd_configs:
  [ - <gce_sd_config> ... ]

# Список конфигураций обнаружения сервисов Hetzner.
hetzner_sd_configs:
  [ - <hetzner_sd_config> ... ]

# Список конфигураций обнаружения сервисов HTTP.
http_sd_configs:
  [ - <http_sd_config> ... ]

# Список конфигураций обнаружения сервисов IONOS.
ionos_sd_configs:
  [ - <ionos_sd_config> ... ]

# Список конфигураций обнаружения сервисов Kubernetes.
kubernetes_sd_configs:
  [ - <kubernetes_sd_config> ... ]

# Список конфигураций обнаружения сервисов Lightsail.
lightsail_sd_configs:
```

```
  [ - <lightsail_sd_config> ... ]

# Список конфигураций обнаружения сервисов Linode.
linode_sd_configs:
  [ - <linode_sd_config> ... ]

# Список конфигураций обнаружения сервисов Marathon.
marathon_sd_configs:
  [ - <marathon_sd_config> ... ]

# Список конфигураций обнаружения сервисов Nerve от AirBnB.
nerve_sd_configs:
  [ - <nerve_sd_config> ... ]

# Список конфигураций обнаружения сервисов Nomad.
nomad_sd_configs:
  [ - <nomad_sd_config> ... ]

# Список конфигураций обнаружения сервисов OpenStack.
openstack_sd_configs:
  [ - <openstack_sd_config> ... ]

# Список конфигураций обнаружения сервисов OVHcloud.
ovhcloud_sd_configs:
  [ - <ovhcloud_sd_config> ... ]

# Список конфигураций обнаружения сервисов PuppetDB.
puppetdb_sd_configs:
  [ - <puppetdb_sd_config> ... ]

# Список конфигураций обнаружения сервисов Scaleway.
scaleway_sd_configs:
  [ - <scaleway_sd_config> ... ]

# Список конфигураций обнаружения сервисов Zookeeper Serverset.
serverset_sd_configs:
  [ - <serverset_sd_config> ... ]

# Список конфигураций обнаружения сервисов Triton.
triton_sd_configs:
  [ - <triton_sd_config> ... ]

# Список конфигураций обнаружения сервисов Uyuni.
uyuni_sd_configs:
  [ - <uyuni_sd_config> ... ]

# Список конфигураций обнаружения сервисов Vultr.
vultr_sd_configs:
  [ - <vultr_sd_config> ... ]
```

```
# Список статически настроенных Alertmanager с лейблами.
static_configs:
  [ - <static_config> ... ]

# Список конфигураций переназначения Alertmanager.
relabel_configs:
  [ - <relabel_config> ... ]

# Список конфигураций переназначения оповещений.
alert_relabel_configs:
  [ - <relabel_config> ... ]
```

remote_write

`write_relabel_configs` — это переназначение, применяемое к образцам перед их отправкой на удаленную конечную точку. Переназначение записи применяется после внешних лейблов. Это может быть использовано для ограничения отправляемых образцов.

Существует небольшая демонстрация использования этой функциональности.

```
# URL конечной точки для отправки образцов.
url: <string>

# protobuf сообщение, используемое при записи на удаленную конечную точку.
#
# * `prometheus.WriteRequest` представляет сообщение, введенное в Remote Write 1.0,
которое
# в конечном итоге будет устаревшим.
# * `io.prometheus.write.v2.Request` было введено в Remote Write 2.0 и заменяет
предыдущее,
# улучшая эффективность и отправляя метаданные, созданные временные метки и нативные
гистограммы по умолчанию.
#
# Перед изменением этого значения проконсультируйтесь с вашим поставщиком удаленного
хранилища (или протестируйте), какие сообщения он поддерживает.
# Подробнее читайте на https://prometheus.io/docs/specs/remote_write_spec_2_0/#io-
prometheus-write-v2-request
[ protobuf_message: <prometheus.WriteRequest | io.prometheus.write.v2.Request> |
default = prometheus.WriteRequest ]

# Тайм-аут для запросов к удаленной конечной точке записи.
[ remote_timeout: <duration> | default = 30s ]

# Пользовательские HTTP заголовки, отправляемые с каждым запросом на удаленную запись.
# Учтите, что заголовки, установленные самим Deckhouse Prom++, не могут быть
перезаписаны.
headers:
  [ <string>: <string> ... ]

# Список конфигураций переназначения удаленной записи.
write_relabel_configs:
  [ - <relabel_config> ... ]

# Имя конфигурации удаленной записи, которое, если указано, должно быть уникальным
среди конфигураций удаленной записи.
# Имя будет использоваться в метриках и логах вместо сгенерированного значения, чтобы
помочь пользователям различать конфигурации удаленной записи.
[ name: <string> ]

# Включает отправку образцов через удаленную запись. Обратите внимание, что само
хранилище образцов должно быть включено, чтобы образцы могли быть собраны.
[ send_exemplars: <boolean> | default = false ]

# Включает отправку нативных гистограмм, также известных как разреженные гистограммы,
через удаленную запись.
# Для сообщения `io.prometheus.write.v2.Request` эта опция неактивна (всегда
включена).
[ send_native_histograms: <boolean> | default = false ]
```

```
# При включении удаленная запись будет разрешать имя хоста URL через DNS, выбирать
один из IP-адресов случайным образом и подключаться к нему.
# При отключении удаленная запись полагается на стандартное поведение Go, которое
заключается в попытке подключиться к каждому адресу по очереди.
# Тайм-аут подключения применяется ко всей операции, т.е. в последнем случае он
распределяется по всем попыткам.
# Это экспериментальная функция, и ее поведение может измениться или даже быть
удалено.
[ round_robin_dns: <boolean> | default = false ]

# Опционально настраивает процесс подписания запросов с использованием AWS Signature
Verification 4.
# Не может быть установлен одновременно с basic_auth, authorization, oauth2 или
azuread.
# Чтобы использовать учетные данные по умолчанию из AWS SDK, используйте `sigv4: {}`.
sigv4:
  # Регион AWS. Если пусто, используется регион из цепочки учетных данных по
умолчанию.
  [ region: <string> ]


# Ключи API AWS. Если пусто, используются переменные окружения `AWS_ACCESS_KEY_ID` и
`AWS_SECRET_ACCESS_KEY`.
  [ access_key: <string> ]
  [ secret_key: <secret> ]

  # Имя профиля AWS, используемого для аутентификации.
  [ profile: <string> ]

  # AWS Role ARN, альтернатива использованию ключей API AWS.
  [ role_arn: <string> ]

# Опциональная конфигурация AzureAD.
# Не может быть использована одновременно с basic_auth, authorization, oauth2, sigv4
или google_iam.
azuread:
  # Облако Azure. Варианты: 'AzurePublic', 'AzureChina' или 'AzureGovernment'.
  [ cloud: <string> | default = AzurePublic ]

  # Назначенная пользователем управляемая идентичность Azure.
  [ managed_identity:
      [ client_id: <string> ] ]

  # OAuth Azure.
  [ oauth:
      [ client_id: <string> ]
      [ client_secret: <string> ]
      [ tenant_id: <string> ] ]
```

```
  # Аутентификация SDK Azure.
  # См. https://learn.microsoft.com/en-us/azure/developer/go/azure-sdk-authentication
  [ sdk:
      [ tenant_id: <string> ] ]

# ПРЕДУПРЕЖДЕНИЕ: Удаленная запись НЕ ПОДДЕРЖИВАЕТСЯ Google Cloud. Эта конфигурация
зарезервирована для будущего использования.
# Опциональная конфигурация мониторинга Google Cloud.
# Не может быть использована одновременно с basic_auth, authorization, oauth2, sigv4
или azuread.
# Чтобы использовать учетные данные по умолчанию из Google Cloud SDK, используйте
`google_iam: {}`.
google_iam:
  # Ключ учетной записи с правами на запись мониторинга.
  credentials_file: <file_name>

# Настраивает очередь, используемую для записи в удаленное хранилище.
queue_config:
  # Количество образцов для буферизации на каждый шард перед блокировкой чтения
дополнительных
  # образцов из WAL. Рекомендуется иметь достаточную емкость в каждом
  # шарде для буферизации нескольких запросов, чтобы поддерживать пропускную
способность при обработке
  # случайных медленных удаленных запросов.
  [ capacity: <int> | default = 10000 ]
  # Максимальное количество шардов, т.е. уровень параллелизма.
  [ max_shards: <int> | default = 50 ]
  # Минимальное количество шардов, т.е. уровень параллелизма.
  [ min_shards: <int> | default = 1 ]
  # Максимальное количество образцов на отправку.
  [ max_samples_per_send: <int> | default = 2000]
  # Максимальное время ожидания образца для отправки. Образец может ждать меньше
  # если буфер заполнен. Дополнительное время может пройти из-за возможных повторных
попыток.
  [ batch_send_deadline: <duration> | default = 5s ]
  # Начальная задержка повторной попытки. Удваивается для каждой повторной попытки.
  [ min_backoff: <duration> | default = 30ms ]
  # Максимальная задержка повторной попытки.
  [ max_backoff: <duration> | default = 5s ]
  # Повторная попытка при получении кода состояния 429 от удаленного хранилища записи.
  # Это экспериментально и может измениться в будущем.
  [ retry_on_http_429: <boolean> | default = false ]
  # Если установлено, любой образец, который старше sample_age_limit
  # не будет отправлен в удаленное хранилище. Значение по умолчанию — 0s,
  # что означает, что все образцы отправляются.
  [ sample_age_limit: <duration> | default = 0s ]

# Настраивает отправку метаданных серий в удаленное хранилище
```

```
# если было выбрано сообщение `prometheus.WriteRequest`. Когда
# используется `io.prometheus.write.v2.Request`, метаданные всегда отправляются.
#
# Конфигурация метаданных может измениться в любой момент
# или быть удалена в будущих выпусках.
metadata_config:
  # Отправляются ли метаданные метрик в удаленное хранилище или нет.
  [ send: <boolean> | default = true ]
  # Как часто метаданные метрик отправляются в удаленное хранилище.
  [ send_interval: <duration> | default = 1m ]
  # Максимальное количество образцов на отправку.
  [ max_samples_per_send: <int> | default = 500]

# Настройки HTTP клиента, включая методы аутентификации (такие как basic auth и
# authorization), конфигурации прокси, параметры TLS, пользовательские HTTP заголовки
и т.д.
# enable_http2 по умолчанию отключен для удаленной записи.
[ <http_config> ]
```

Существует список интеграций с этой функцией.

remote_read

```
# URL конечной точки для выполнения запросов.
url: <string>

# Имя конфигурации удаленного чтения, которое, если указано, должно быть уникальным
среди конфигураций удаленного чтения.
# Имя будет использоваться в метриках и логах вместо сгенерированного значения, чтобы
помочь пользователям различать конфигурации удаленного чтения.
[ name: <string> ]

# Опциональный список эквивалентных сопоставителей, которые должны быть
# присутствовать в селекторе для выполнения запроса к удаленной конечной точке чтения.
required_matchers:
  [ <labelname>: <labelvalue> ... ]

# Тайм-аут для запросов к удаленной конечной точке чтения.
[ remote_timeout: <duration> | default = 1m ]

# Пользовательские HTTP заголовки, отправляемые с каждым запросом на удаленное чтение.
# Учтите, что заголовки, установленные самим Deckhouse Prom++, не могут быть
перезаписаны.
headers:
  [ <string>: <string> ... ]

# Должны ли выполняться чтения для запросов на временные диапазоны, для которых
# локальное хранилище должно иметь полные данные.
[ read_recent: <boolean> | default = false ]

# Использовать ли внешние лейблы в качестве селекторов для удаленной конечной точки
чтения.
[ filter_external_labels: <boolean> | default = true ]

# Настройки HTTP клиента, включая методы аутентификации (такие как basic auth и
# authorization), конфигурации прокси, параметры TLS, пользовательские HTTP заголовки
и т.д.
[ <http_config> ]
```

Существует список интеграций с этой функцией.

tsdb

`tsdb` позволяет настроить параметры конфигурации TSDB, которые можно перезагрузить во время выполнения.

```
# Настраивает, насколько старым может быть образец, находящийся вне порядка/границ,
относительно максимального времени TSDB.
# Образец, находящийся вне порядка/границ, загружается в TSDB, если временная метка
# образца >= TSDB.MaxTime-out_of_order_time_window.
#
# Когда out_of_order_time_window >0, ошибки вне порядка и вне границ
# объединяются в одну ошибку, называемую 'слишком старый'; образец либо (a) может быть
загружен
# в TSDB, т.е. это образец в порядке или образец вне порядка/границ,
# который находится в пределах окна вне порядка, либо (b) слишком старый, т.е. не в
порядке
# и до окна вне порядка.
#
# Когда out_of_order_time_window больше 0, это также влияет на экспериментальный
агент. Это позволяет
# WAL агента принимать образцы вне порядка, которые попадают в указанное временное
окно относительно
# временной метки последнего добавленного образца для той же серии.
[ out_of_order_time_window: <duration> | default = 0s ]
```

exemplars

Обратите внимание, что хранилище образцов все еще считается экспериментальным и должно быть включено через `--enable-feature=exemplar-storage`.

```
# Настраивает максимальный размер кольцевого буфера, используемого для хранения
образцов для всех серий. Может изменяться во время выполнения.
[ max_exemplars: <int> | default = 100000 ]
```

tracing_config

`tracing_config` настраивает экспорт трассировок из Deckhouse Prom++ в бэкенд трассировки через протокол OTLP. Трассировка в настоящее время является **экспериментальной** функцией и может измениться в будущем.

```
# Клиент, используемый для экспорта трассировок. Варианты: 'http' или 'grpc'.
[ client_type: <string> | default = grpc ]

# Конечная точка для отправки трассировок. Должна быть предоставлена в формате
<host>:<port>.
[ endpoint: <string> ]

# Устанавливает вероятность, с которой будет выбрана данная трассировка. Должно быть
числом с плавающей запятой от 0 до 1.
[ sampling_fraction: <float> | default = 0 ]

# Если отключено, клиент будет использовать защищенное соединение.
[ insecure: <boolean> | default = false ]

# Пары ключ-значение, используемые в качестве заголовков, связанных с gRPC или HTTP
запросами.
headers:
  [ <string>: <string> ... ]

# Ключ сжатия для поддерживаемых типов сжатия. Поддерживаемое сжатие: gzip.
[ compression: <string> ]

# Максимальное время, которое экспортер будет ждать для каждой пакетной отправки.
[ timeout: <duration> | default = 10s ]

# Конфигурация TLS.
tls_config:
  [ <tls_config> ]
```

Правила оповещения

Правила оповещения (alerting rules) в Deckhouse Prom++ позволяют определять условия оповещения на основе выражений языка запросов PromQL и отправлять уведомления о срабатывающих оповещениях во внешние сервисы. Когда выражение оповещения возвращает один или несколько элементов векторного типа в определенный момент времени, оповещение считается активным для этих наборов лейблов.

Определение правил оповещения

Правила оповещения настраиваются в Deckhouse Prom++ аналогично правилам производных метрик. Пример файла правил с оповещением:

```
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
```

Опциональная директива `for` заставляет Deckhouse Prom++ заставляет Prometheus подождать указанный интервал времени между первым срабатыванием выражения и фактической активацией алерта для данного элемента. В этом случае Deckhouse Prom++ будет проверять, что условие алерта продолжает выполняться на протяжении каждой оценки в течение, например, 10 минут, прежде чем перевести алерт в состояние «firing» (сработал). Элементы, для которых условие выполняется, но ещё не сработали, считаются находящимися в состоянии pending. Алерты без секции `for` переходят в активное состояние сразу после первой удачной проверки условия.

Секция `labels` позволяет указать дополнительные метки, которые будут прикреплены к алерту. Если какие-то метки уже существуют, они будут перезаписаны. Значения лейблов можно задавать с помощью шаблонов (templating).

Секция `annotations` задаёт информационные метки, используемые для хранения дополнительной информации, например, описания алерта или ссылки на runbook. Значения аннотаций также могут быть шаблонизированы.

Шаблонизация (Templates)

Значения лейблов (labels) и аннотаций (annotation) могут быть шаблонизированы с помощью шаблонов консоли (console templates). Переменная `$labels` содержит пары ключ/значение лейблов (labels) конкретного экземпляра алерта. К сконфигурированным внешним лейблам (label) можно получить доступ через переменную `$externalLabels`. Переменная `$value` содержит вычисленное значение, при котором сработал алерт.

```
#Для вставки значений лейблов (labels) сработавшего элемента:
{{ $labels.<имя_лейбла> }}

#Для вставки значения выражения сработавшего элемента:
{{ $value }}
```

Примеры:

```
groups:
- name: example
  rules:

  # Алерт для любого экземпляра, недоступного более 5 минут.
  - alert: InstanceDown
    expr: up == 0
    for: 5m
    labels:
      severity: page
    annotations:
      summary: "Instance {{ $labels.instance }} down"
      description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for
more than 5 minutes."

  # Алерт для любого экземпляра с медианной задержкой запроса >1c.
  - alert: APIHighRequestLatency
    expr: api_http_request_latencies_second{quantile="0.5"} > 1
    for: 10m
    annotations:
      summary: "High request latency on {{ $labels.instance }}"
      description: "{{ $labels.instance }} has a median request latency above 1s
(current value: {{ $value }}s)"
```

Просмотр алертов во время исполнения

Чтобы вручную посмотреть, какие алерты активны (в состоянии pending или firing), перейдите на вкладку **Алерты (Alerts)** в интерфейсе Deckhouse Prom++.
Там вы увидите точные наборы лейблов (labels), для которых каждый определённый алерт в данный момент активен.

Для алертов в состояниях pending и firing Deckhouse Prom++ также создаёт синтетические временные ряды следующего вида:

`ALERTS{alertname="<имя алерта>", alertstate="<pending или firing>", <дополнительные лейблы алерта>}`

Значение sample устанавливается в `1`, пока алерт остаётся в активном состоянии (pending или firing).
Когда это состояние больше не актуально, временной ряд помечается как устаревший (stale).

Отправка уведомлений на основании правил оповещений

Правила оповещений в Deckhouse Prom++ хорошо справляются с определением того, что “сломано прямо сейчас”,
но сами по себе не являются полноценным решением для доставки уведомлений.
Требуется дополнительный уровень, чтобы добавить такие возможности, как доставка

уведомлений, агрегация, ограничение частоты уведомлений, приглушение и зависимые алерты. Эту роль может выполнять Alertmanager.

Deckhouse Prom++ может быть сконфигурирован на периодическую отправку информации о состоянии алертов в Alertmanager,
который далее занимается доставкой уведомлений получателям.

Также Deckhouse Prom++ можно настроить на автоматическое обнаружение доступных экземпляров Alertmanager с помощью существующих механизмов service discovery.

Производные метрики

Производные метрики (recording rules) в Prom++ позволяют предварительно вычислять часто используемые или ресурсоемкие выражения и сохранять их результаты как новый набор временных рядов. Это ускоряет выполнение запросов, поскольку вместо повторного вычисления выражения каждый раз используется заранее сохраненный результат. Это особенно полезно для дашбордов, которые регулярно обновляют одни и те же выражения.

Определение производных метрик

Производные метрики и правила оповещения объединяются в группы правил. Внутри группы правила выполняются последовательно с регулярным интервалом, используя одно и то же время оценки. Имена производных метрик должны быть допустимыми именами метрик, а имена правил оповещения — допустимыми значениями лейблов.
Синтаксис файла правил выглядит следующим образом:

```
groups:
  - name: <имя_группы>
    rules:
      - record: <имя_метрики>
        expr: <выражение_PromQL>
        [ labels:
            <имя_метки>: <значение_метки>
            ...
        ]
```

Пример простого файла правил:

```
groups:
  - name: example
    rules:
    - record: code:prompp_http_requests_total:sum
      expr: sum by (code) (prompp_http_requests_total)
```

Параметры rule_group

- name: Имя группы. Должно быть уникальным в пределах файла.
- interval: Как часто оцениваются правила в группе. По умолчанию используется значение global.evaluation_interval.
- limit: Ограничивает количество оповещений, создаваемых правилом оповещения, и временных рядов, создаваемых производной метрикой. Значение 0 означает отсутствие ограничений.
- query_offset: Смещение времени оценки правила для этой группы на указанную длительность в прошлое. Полезно для обеспечения того, чтобы базовые метрики были получены и сохранены в Prom++.
- labels: Метки, которые добавляются или перезаписываются перед сохранением результата для правил этой группы. Метки, определенные в , переопределяют совпадающие ключи.
- rules: Список правил в группе.

Параметры rule

Для производных метрик:

- record: Имя метрики для сохранения результата. Должно быть допустимым именем метрики.
- expr: Выражение PromQL для оценки. Каждый цикл оценки это выражение вычисляется в текущее время, и результат записывается как новый набор временных рядов с именем метрики, указанным в record.
- labels: Метки, которые добавляются или перезаписываются перед сохранением результата.

Для правил оповещения:

- alert: Имя оповещения. Должно быть допустимым значением метки.
- expr: Выражение PromQL для оценки. Каждый цикл оценки это выражение вычисляется в текущее время, и все полученные временные ряды становятся ожидающими или активными оповещениями.
- for: Оповещения считаются активными после того, как они существуют в течение этого времени. Оповещения, которые еще не активны достаточно долго, считаются ожидающими.
- keep_firing_for: Как долго оповещение будет продолжать оставаться активным после того, как условие, которое его вызвало, исчезло.
- labels: Метки, которые добавляются или перезаписываются для каждого оповещения.
- annotations: Аннотации, которые добавляются к каждому оповещению.

Ограничение количества оповещений и временных рядов

Можно настроить ограничение на количество оповещений, создаваемых правилами оповещения, и временных рядов, создаваемых производными метриками, для каждой группы. Если это ограничение превышено, все серии, созданные правилом, отбрасываются, а для правил оповещения все оповещения (активные, ожидающие или неактивные) также очищаются. Это событие будет зарегистрировано как ошибка в оценке, и, таким образом, никакие устаревшие метки не будут записаны.

Смещение запроса правила

Смещение времени оценки правила полезно для обеспечения того, чтобы базовые метрики были получены и сохранены в Prom++. Задержки в доступности метрик более вероятны, когда Prom++ работает как цель для удаленной записи из-за природы распределенных систем, но могут также возникать при аномалиях с опросом и/или короткими интервалами оценки.

Неудачные оценки правил из-за медленной оценки

Если группа правил не завершила оценку до того, как должна начаться следующая оценка (определенная evaluation_interval), следующая оценка будет пропущена. Последующие оценки группы правил будут продолжать пропускаться до тех пор, пока начальная оценка не завершится или не истечет время ожидания. Когда это происходит, в метрике, созданной производной метрикой, будет пробел. Метрика rule_group_iterations_missed_total будет увеличиваться для каждой пропущенной итерации группы правил.

Справочник по шаблонизации

Deckhouse Prom++ поддерживает шаблонизацию в аннотациях и лейблах алертов, а также на отображаемых консольных страницах. Шаблоны позволяют выполнять запросы к локальной базе данных, итерировать данные, использовать условные конструкции, форматировать данные и многое другое. Язык шаблонов Deckhouse Prom++ основан на системе Go шаблонов.

Структуры данных

Основной структурой данных для работы с временными рядами является сэмпл, определённый следующим образом:

```
type sample struct {
        Labels map[string]string
        Value  interface{}
}
```

Имя метрики сэмпла кодируется в специальной метке `__name__` в карте `Labels`.

`[]sample` означает список сэмплов.

`interface{}` в Go аналогичен указателю void в C.

Функции

Помимо стандартных функций, предоставляемых Go-шаблонами, Deckhouse Prom++ предоставляет функции для упрощённой обработки результатов запросов в шаблонах.

Если функции используются в конвейере, значение конвейера передаётся в качестве последнего аргумента.

Запросы

Имя	Аргументы	Возвращает	Примечания
query	строка запроса	[]sample	Выполняет запрос к базе данных, не поддерживает возврат векторов диапазона.
first	[]sample	sample	Эквивалентно `index a 0`.
label	лейбл, sample	string	Эквивалентно `index sample.Labels label`.
value	sample	interface{}	Эквивалентно `sample.Value`.
sortByLabel	лейбл, []sample	[]sample	Сортирует сэмплы по указанному лейблу. Стабильная сортировка.

Функции `first`, `label` и `value` предназначены для упрощения использования результатов запросов в конвейерах.

Числа

Имя	Аргументы	Возвращает	Примечания
humanize	число или строка	строка	Преобразует число в более читаемый формат, используя метрические префиксы.
humanize1024	число или строка	строка	Как `humanize`, но использует основание 1024 вместо 1000.
humanizeDuration	число или строка	строка	Преобразует длительность в секундах в более читаемый формат.
humanizePercentage	число или строка	строка	Преобразует значение отношения в долю от 100.
humanizeTimestamp	число или строка	строка	Преобразует Unix-временную метку в секундах в более читаемый формат.
toTime	число или строка	*time.Time	Преобразует Unix-временную метку в секундах в time.Time.

Функции для “очеловечивания” предназначены для создания удобного для восприятия человеком вывода и не гарантируют одинаковые результаты между версиями Deckhouse Prom++.

Строки

Имя	Аргументы	Возвращает	Примечания
title	строка	строка	cases.Title, делает первую букву каждого слова заглавной.
toUpper	строка	строка	strings.ToUpper, преобразует все символы в верхний регистр.
toLower	строка	строка	strings.ToLower, преобразует все символы в нижний регистр.
stripPort	строка	строка	net.SplitHostPort, разделяет строку на хост и порт, возвращает только хост.
match	шаблон, текст	boolean	regexp.MatchString, проверяет совпадение с регулярным выражением.
reReplaceAll	шаблон, замена, текст	строка	Regexp.ReplaceAllString, замена по регулярному выражению.
graphLink	выражение	строка	Возвращает путь к графическому представлению в браузере выражений для данного выражения.
tableLink	выражение	строка	Возвращает путь к табличному представлению в браузере выражений

Имя	Аргументы	Возвращает	Примечания
			для данного выражения.
parseDuration	строка	float	Разбирает строку длительности, такую как “1h”, в количество секунд.
stripDomain	строка	строка	Удаляет доменную часть из полного доменного имени (FQDN). Порт остаётся нетронутым.

Прочее

Имя	Аргументы	Возвращает	Примечания
args	[]interface{}	map[string]interface{}	Преобразует список объектов в карту с ключами arg0, arg1 и т.д. Это позволяет передавать несколько аргументов в шаблоны.
tmpl	строка, []interface{}	ничего	Как встроенная `template`, но позволяет использовать не литералы в качестве имени шаблона. Результат считается безопасным и не будет автоматически экранироваться. Доступно только в консолях.
safeHtml	строка	строка	Помечает строку как HTML, не требующий автоматического экранирования.
externalURL	*нет*	строка	Внешний URL, по которому Deckhouse Prom++ доступен извне.
pathPrefix	*нет*	строка	Внешний URL [путь](#) для использования в консольных шаблонах.

Различия типов шаблонов

Каждый из типов шаблонов предоставляет различную информацию, которая может быть использована для параметризации шаблонов, и имеет несколько других отличий.

Шаблоны полей алертов

`.Value`, `.Labels`, `.ExternalLabels` и `.ExternalURL` содержат значение алерта, лейблы алерта, глобально настроенные внешние лейблы и внешний URL (настраивается с помощью `--web.external-url`) соответственно. Они также доступны как переменные `$value`, `$labels`, `$externalLabels` и `$externalURL` для удобства.

Консольные шаблоны

Консоли доступны по адресу `/consoles/` и берутся из директории, указанной флагом `-web.console.templates`.

Консольные шаблоны рендерятся с использованием html/template, который обеспечивает автоматическое экранирование. Чтобы обойти автоматическое экранирование, используйте функции `safe*`.

Параметры URL доступны как карта в `.Params`. Для доступа к нескольким параметрам URL с одинаковым именем `.RawParams` представляет собой карту со списком значений для каждого параметра. Путь URL доступен в `.Path`, исключая префикс `/consoles/`. Глобально настроенные внешние лейблы доступны в `.ExternalLabels`. Также доступны удобные переменные для всех четырёх: `$rawParams`, `$params`, `$path` и `$externalLabels`.

Консоли также имеют доступ ко всем шаблонам, определённым с помощью `{{define "templateName"}}...{{end}}`, найденным в файлах `*.lib` в директории, указанной флагом `-web.console.libraries`. Поскольку это общее пространство имён, избегайте конфликтов с другими пользователями. Имена шаблонов, начинающиеся с `prom`, `_prom` и `__`, зарезервированы для использования Deckhouse Prom++, как и перечисленные выше функции.

Примеры шаблонов

Prom++ поддерживает использование шаблонов в аннотациях и метках оповещений, а также на отображаемых консольных страницах. Шаблоны позволяют выполнять запросы к локальной базе данных, итерировать данные, использовать условные конструкции, форматировать данные и т.д. Шаблонизатор Prom++ основан на системе шаблонов Go.

Простые шаблоны полей оповещений

```
alert: InstanceDown
expr: up == 0
for: 5m
labels:
  severity: page
annotations:
  summary: "Instance {{$labels.instance}} down"
  description: "{{$labels.instance}} of job {{$labels.job}} has been down for more
than 5 minutes."
```

Шаблоны полей оповещений выполняются при каждой итерации правила для каждого сработавшего оповещения, поэтому рекомендуется, чтобы любые запросы и шаблоны были легковесными. Если требуется использовать более сложные шаблоны для оповещений, рекомендуется ссылаться на консоль.

Простая итерация

Этот пример отображает список экземпляров и их статус:

```
{{ range query "up" }}
  {{ .Labels.instance }} {{ .Value }}
{{ end }}
```

Специальная переменная `.` содержит значение текущей выборки для каждой итерации цикла.

Отображение одного значения

```
{{ with query "some_metric{instance='someinstance'}" }}
  {{ . | first | value | humanize }}
{{ end }}
```

Go и его система шаблонов имеют строгую типизацию, поэтому необходимо проверять, что выборки были возвращены, чтобы избежать ошибки выполнения. Например, это может произойти, если сбор данных или оценка правил еще не были выполнены, или если хост недоступен.

Встроенный шаблон `prom_query_drilldown` обрабатывает это, позволяет форматировать результаты и ссылаться на браузер выражений.

Использование параметров URL консоли

Шаблоны могут использовать параметры URL для настройки отображаемых данных. Например, можно создать консоль, которая принимает параметр `job` и отображает метрики для этого задания:

```
{{ $job := .QueryParams.job }}
{{ range query "up{job=\"$job\"}" }}
  {{ .Labels.instance }} {{ .Value }}
{{ end }}
```

В этом примере значение параметра `job` из URL используется для фильтрации метрик по заданию.

Продвинутая итерация

Шаблоны позволяют выполнять сложные итерации и условия. Например, можно отобразить список всех метрик с их значениями, отфильтровав по определенному префиксу:

```
{{ range query "{__name__=~\"^prefix_.*\"}" }}
  {{ .Labels.__name__ }}: {{ .Value }}
{{ end }}
```

Этот шаблон отобразит все метрики, имена которых начинаются с `prefix_`.

Определение повторно используемых шаблонов

Вы можете определить шаблоны, которые можно повторно использовать в разных частях консоли или оповещений:

```
{{ define "metricTable" }}
  <table>
    <tr><th>Метрика</th><th>Значение</th></tr>
    {{ range . }}
      <tr><td>{{ .Labels.__name__ }}</td><td>{{ .Value }}</td></tr>
    {{ end }}
  </table>
{{ end }}

{{ template "metricTable" (query "up") }}
```

В этом примере определяется шаблон `metricTable`, который создает HTML-таблицу для отображения метрик и их значений. Затем этот шаблон вызывается с результатами запроса `up`.

Эти примеры демонстрируют гибкость и мощность системы шаблонов Prom++ для настройки отображения данных и оповещений.

Язык запросов PromQL

Основы

Deckhouse Prom++ предоставляет функциональный язык запросов под названием PromQL (Prometheus Query Language), который позволяет пользователю выбирать и агрегировать данные временных рядов в реальном времени. Результат выражения может быть представлен в виде графика, просмотрен как табличные данные в браузере выражений Deckhouse Prom++ или использован внешними системами через HTTP API. citeturn0search0

Примеры

На этой странице представлены базовые сведения о языке PromQL. В целях обучения может быть проще начать с [нескольких примеров].

Типы данных в языке выражений

В языке выражений Deckhouse Prom++ выражение или подвыражение может возвращать значение одного из четырех типов:

- **Мгновенный вектор (instant vector)** — набор временных рядов, содержащих по одной выборке для каждого временного ряда, при этом всем выборкам назначена одинаковая временная метка.
- **Диапазонный вектор (range vector)** — набор временных рядов, содержащих диапазон точек данных за определенный промежуток времени для каждого временного ряда.
- **Скаляр (scalar)** — простое числовое значение с плавающей запятой.
- **Строка (string)** — простое строковое значение. В настоящее время не используется.

В зависимости от сценария использования (например, при построении графиков или отображении результата выражения) в качестве результата выражения допускаются не все типы данных. Например, выражение, которое возвращает мгновенный вектор, является единственным типом, который можно отобразить на графике.

Литералы

Строковые литералы

Строковые литералы обозначаются одинарными кавычками (`'`), двойными кавычками (`"`) или обратными апострофами (`` ` ``).

PromQL следует тем же правилам экранирования, что и язык Go. В строковых литералах в одинарных или двойных кавычках обратная косая черта начинает escape-последовательность, которая может включать `a`, `b`, `f`, `n`, `r`, `t`, `v` или `\`. Конкретные символы могут быть указаны с использованием восьмеричной (`\nnn`) или шестнадцатеричной (`\xnn`, `\unnnn` и `\Unnnnnnnn`) нотации.

Напротив, escape-последовательности не обрабатываются в строковых литералах, обозначенных обратными апострофами. Важно отметить, что в отличие от Go, Deckhouse Prom++ не игнорирует символы новой строки внутри обратных апострофов.

Примеры:

```
"это строка"
'это неэкранированные последовательности: \n \\ \t'
`это не неэкранированные последовательности: \n ' " \t`
```

Числовые литералы

Скалярные значения с плавающей точкой могут быть записаны как целочисленные или дробные литералы в следующем формате (пробелы добавлены для лучшей читаемости):

```
[-+]?(
      [0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?
    | 0[xX][0-9a-fA-F]+
    | [nN][aA][nN]
    | [iI][nN][fF]
)
```

Примеры:

```
23
-2.43
3.4e-9
0x8f
-Inf
NaN
```

Селекторы временных рядов

Селекторы временных рядов отвечают за выбор временных рядов и соответствующих сырых или вычисленных значений и временных лейблов.

Селекторы временных рядов не следует путать с концепцией мгновенных и интервальных запросов более высокого уровня, которые могут выполнять эти селекторы. Мгновенный запрос вычисляет данный селектор в одной точке времени, тогда как интервальный запрос вычисляет селектор в нескольких точках времени между начальной и конечной метками с регулярными интервалами.

Селекторы мгновенных векторов

Селекторы мгновенных векторов позволяют выбрать набор временных рядов и одно значение выборки для каждого в заданный момент времени. В простейшей форме указывается только имя метрики, что приводит к вектору, содержащему элементы для всех временных рядов с этим именем метрики.

Этот пример выбирает все временные ряды с именем метрики `http_requests_total`:

```
http_requests_total
```

Можно дополнительно отфильтровать эти временные ряды, добавив список сопоставителей лейблов в фигурных скобках (`{}`), разделенных запятыми.

Этот пример выбирает только те временные ряды с именем метрики `http_requests_total`, которые также имеют метку `job` со значением `Deckhouse Prom++` и метку `group` со значением `canary`:

```
http_requests_total{job="Deckhouse Prom++",group="canary"}
```

Также возможно отрицательное сопоставление значений лейблов или сопоставление значений лейблов с регулярными выражениями. Существуют следующие операторы сопоставления лейблов:

- `=` — выбирает лейблы, точно равные указанной строке;
- `!=` — выбирает лейблы, не равные указанной строке;
- `=~` — выбирает лейблы, соответствующие регулярному выражению;
- `!~` — выбирает лейблы, не соответствующие регулярному выражению.

Сопоставления с регулярными выражениями являются полностью закрепленными. Сопоставление `env=~"foo"` трактуется как `env=~"^foo$"`.

Например, этот запрос выбирает все временные ряды `http_requests_total` для сред `staging`, `testing` и `development` и HTTP-методов, отличных от `GET`:

```
http_requests_total{environment=~"staging|testing|development",method!="GET"}
```

Селекторы диапазонных векторов

Селекторы диапазонных векторов работают аналогично селекторам мгновенных векторов, но выбирают диапазон значений, начиная от текущего момента назад. Синтаксически, временной диапазон указывается в квадратных скобках (`[]`) после селектора вектора, определяя, как далеко в прошлое следует получить значения для каждого элемента диапазонного вектора. Диапазон является замкнутым интервалом, то есть выборки с временными метками, совпадающими с границами диапазона, все равно включаются в выбор.

В этом примере мы выбираем все значения, записанные за последние 5 минут, для всех временных рядов с именем метрики `http_requests_total` и меткой `job`, равной `Deckhouse Prom++`:

```
http_requests_total{job="Deckhouse Prom++"}[5m]
```

Временные диапазоны

Временные диапазоны указываются как число, за которым сразу следует одна из следующих единиц измерения:

- `ms` — миллисекунды;
- `s` — секунды;
- `m` — минуты;
- `h` — часы;
- `d` — дни (подразумевается, что день всегда равен 24 часам);
- `w` — недели (подразумевается, что неделя всегда равна 7 дням);
- `y` — годы (подразумевается, что год всегда равен 365 дням).

> Для дней в году високосные дни игнорируются. Аналогичным образом для минут игнорируются високосные секунды.

Временные диапазоны можно комбинировать путем конкатенации. Единицы измерения должны быть упорядочены от самой длинной к самой короткой. Каждая единица измерения может встречаться только один раз во временном диапазоне.

Вот несколько примеров допустимых временных диапазонов:

```
5h
1h30m
5m
10s
```

Модификаторы запросов

Модификатор offset

Модификатор `offset` позволяет изменять временное смещение для отдельных мгновенных и диапазонных векторов в запросе.

Например, следующее выражение возвращает значение `http_requests_total` 5 минут назад относительно времени выполнения текущего запроса:

```
http_requests_total offset 5m
```

> Модификатор `offset` всегда должен следовать непосредственно за селектором. Например, следующий вариант корректен:
>
> ```
> sum(http_requests_total{method="GET"} offset 5m) // ПРАВИЛЬНО
> ```
>
> Это некорректный вариант:
>
> ```
> sum(http_requests_total{method="GET"}) offset 5m // НЕПРАВИЛЬНО
> ```

То же самое применимо для диапазонных векторов. Этот запрос возвращает 5-минутный `rate`, который был у `http_requests_total` неделю назад:

```
rate(http_requests_total[5m] offset 1w)
```

При запросе данных из прошлого отрицательное смещение позволяет выполнять временные сравнения в будущем:

```
rate(http_requests_total[5m] offset -1w)
```

Это позволяет запросу «заглядывать» вперед относительно времени его выполнения.

Модификатор @

Модификатор `@` позволяет изменять время выполнения для отдельных мгновенных и диапазонных векторов в запросе. Время, передаваемое модификатору `@`, представляет собой временную метку в Unix-формате и задается числовым литералом с плавающей точкой.

Например, следующее выражение возвращает значение `http_requests_total` по состоянию на `2021-01-04T07:40:00+00:00`:

```
http_requests_total @ 1609746000
```

Модификатор `@` всегда должен следовать непосредственно за селектором. Например, следующий вариант корректен:

```
sum(http_requests_total{method="GET"} @ 1609746000) // ПРАВИЛЬНО
```

Это некорректный вариант:

```
sum(http_requests_total{method="GET"}) @ 1609746000 // НЕПРАВИЛЬНО
```

То же самое применимо для диапазонных векторов. Этот запрос возвращает 5-минутный `rate`, который был у `http_requests_total` по состоянию на `2021-01-04T07:40:00+00:00`:

```
rate(http_requests_total[5m] @ 1609746000)
```

Модификатор `@` поддерживает все форматы числовых литералов, описанные выше. Он работает совместно с модификатором `offset`, где смещение применяется относительно времени, указанного в `@`. Результаты будут одинаковыми независимо от порядка модификаторов.

Например, оба этих запроса дадут одинаковый результат:

```
# offset после @
http_requests_total @ 1609746000 offset 5m
# offset перед @
http_requests_total offset 5m @ 1609746000
```

Дополнительно, `start()` и `end()` также могут использоваться в качестве значений для модификатора `@` как специальные значения.

Для интервального запроса они соответствуют началу и концу диапазона запроса соответственно и остаются одинаковыми для всех шагов.

При мгновенном запросе `start()` и `end()` соответствуют времени выполнения запроса.

Примеры:

```
http_requests_total @ start()
rate(http_requests_total[5m] @ end())
```

> Модификатор `@` позволяет запросу «заглядывать» вперед относительно времени его выполнения.

Подзапросы

Подзапрос позволяет выполнить мгновенный запрос для заданного диапазона и разрешения. Результатом подзапроса является диапазонный вектор.

Синтаксис:

```
<instant_query> '[' <range> ':' [<resolution>] ']' [ @ <float_literal> ] [ offset
<duration> ]
```

где:

- — необязательный параметр. По умолчанию используется глобальный интервал вычисления.

Операторы

Deckhouse Prom++ поддерживает множество бинарных и агрегационных операторов. Они подробно описаны на странице Операторы PromQL.

Функции

Deckhouse Prom++ поддерживает несколько функций для работы с данными. Они подробно описаны на странице функции языка выражений.

Комментарии

PromQL поддерживает строчные комментарии, начинающиеся с `#`. Пример:

```
# Это комментарий.
   metric{label="value"} # Это тоже комментарий.
```

Особенности и подводные камни

Устаревание данных

Временные метки, по которым производится выборка данных во время запроса, выбираются независимо от фактических данных временных рядов. Это сделано в основном для поддержки таких случаев, как агрегация (`sum` , `avg` и т.д.), где несколько агрегируемых временных рядов не выровнены точно по времени. Из-за этой независимости Deckhouse Prom++ должен присвоить значение для каждого соответствующего временного ряда в этих временных метках. Это делается путем выбора самой новой выборки перед этой временной меткой в пределах периода обратного просмотра. По умолчанию период обратного просмотра составляет 5 минут.

Если сбор данных с цели или вычисление правил больше не возвращает выборки для временного ряда, который присутствовал ранее, этот временной ряд будет помечен как устаревший. Если цель удаляется, ранее полученные временные ряды будут вскоре помечены как устаревшие.

Если запрос выполняется после того, как временной ряд был помечен как устаревший, то для этого временного ряда не возвращается никакое значение. Если впоследствии для этого временного ряда поступают новые выборки, они будут возвращаться как обычно.

Временной ряд становится устаревшим, когда он больше не экспортируется или когда цель больше не существует. Такие временные ряды исчезают из графиков в моменты времени их последних собранных выборок, и они не возвращаются в запросах после того, как были помечены как устаревшие.

Некоторые экспортеры, которые добавляют свои собственные метки времени к выборкам, действуют иначе: ряды, которые перестают экспортироваться, сохраняют последнее значение в течение 5 минут (значение по умолчанию) перед исчезновением. Настройка `track_timestamps_staleness` может изменить это поведение.

Избегание медленных запросов и перегрузок

Если запрос должен обработать значительное количество данных, его построение в виде графика может привести к таймауту или перегрузке сервера или браузера. Поэтому при создании запросов к неизвестным данным всегда начинайте с табличного представления в браузере выражений Deckhouse Prom++, пока результирующий набор не покажется разумным (сотни, а не тысячи временных рядов максимум). Только после того, как вы достаточно отфильтровали или агрегировали данные, переключайтесь в режим графика. Если выражение все еще занимает слишком много времени для построения в реальном времени, предварительно вычислите его с помощью производных метрик.

Это особенно актуально для языка запросов Deckhouse Prom++, где простой селектор имени метрики, такой как `api_http_requests_total` , может развернуться в тысячи временных рядов с различными метками. Также имейте в виду, что выражения, которые агрегируют множество временных рядов, будут создавать нагрузку на сервер, даже если результат содержит небольшое количество временных рядов. Это аналогично тому, как вычисление суммы всех значений столбца

в реляционной базе данных может проходить медленно, даже если результатом является однозначное число.

Операторы PromQL

Deckhouse Prom++ поддерживает различные операторы, которые можно использовать в выражениях для обработки и анализа временных рядов. Операторы делятся на **бинарные** и **агрегационные**.

Бинарные операторы

Бинарные операторы применяются между двумя операндами и включают арифметические, тригонометрические, операторы сравнения и логические/множественные операторы.

Арифметические бинарные операторы

В Deckhouse Prom++ доступны следующие арифметические операторы:

- `+` (сложение)
- `-` (вычитание)
- `*` (умножение)
- `/` (деление)
- `%` (остаток от деления)
- `^` (возведение в степень)

Эти операторы могут применяться между:

- **Скаляр/Скаляр**: результатом будет скалярное значение, полученное применением оператора к обоим операндам.
- **Вектор/Скаляр**: оператор применяется к каждому значению вектора с использованием скалярного значения. Например, умножение вектора на 2 приведёт к умножению каждого значения вектора на 2. При этом имя метрики удаляется.
- **Вектор/Вектор**: оператор применяется к соответствующим элементам двух векторов. Результат содержит элементы, для которых найдена соответствующая пара в обоих векторах. Имя метрики удаляется.

Тригонометрические бинарные операторы

Deckhouse Prom++ поддерживает следующие тригонометрические операторы:

- `atan2` (арктангенс с двумя параметрами)

Оператор `atan2` принимает два аргумента и возвращает арктангенс частного этих аргументов.

Операторы сравнения

Операторы сравнения позволяют сравнивать значения и включают:

- `==` (равно)
- `!=` (не равно)
- `>` (больше)
- `<` (меньше)
- `>=` (больше или равно)
- `<=` (меньше или равно)

Операторы сравнения могут применяться между скалярами, векторами и комбинациями скаляров и векторов. При сравнении векторов результатом будет вектор, содержащий только те элементы, для которых условие сравнения истинно. Имя метрики удаляется.

Логические/множественные операторы

Логические операторы используются для работы с множествами временных рядов и включают:

- `and` (пересечение)
- `or` (объединение)
- `unless` (разность множеств)

Эти операторы позволяют комбинировать или исключать временные ряды на основе условий. Например, оператор `and` возвращает только те временные ряды, которые присутствуют в обоих векторах.

Сопоставление векторов (Vector Matching)

При применении бинарных операторов к двум векторным выражениям необходимо определить, как элементы одного вектора сопоставляются с элементами другого. Это называется **сопоставлением векторов**.

Ключевые слова сопоставления векторов

Сопоставление векторов регулируется следующими ключевыми словами:

- `on` (**по**): определяет список лейблов, по которым должно происходить сопоставление элементов векторов.
- `ignoring` (**игнорируя**): определяет список лейблов, которые должны быть проигнорированы при сопоставлении элементов векторов.

Например:

```
metric1 * on(instance) metric2
```

В этом примере элементы `metric1` и `metric2` будут сопоставляться по метке `instance`.

Модификаторы группировки

Модификаторы группировки используются для управления тем, как элементы векторов группируются при сопоставлении:

- `group_left` (**группа_слева**): используется, когда левый вектор имеет меньше элементов, чем правый, и необходимо сохранить все элементы правого вектора.
- `group_right` (**группа_справа**): используется, когда правый вектор имеет меньше элементов, чем левый, и необходимо сохранить все элементы левого вектора.

Например:

```
metric1 * on(instance) group_left(job) metric2
```

В этом примере все элементы `metric2` будут сохранены, даже если для них нет соответствующих элементов в `metric1`.

Агрегационные операторы

Агрегационные операторы позволяют объединять несколько временных рядов в один на основе определённых лейблов. Основные агрегационные операторы включают:

- `sum` (**сумма**): вычисляет сумму значений.
- `min` (**минимум**): находит минимальное значение.
- `max` (**максимум**): находит максимальное значение.
- `avg` (**среднее**): вычисляет среднее значение.

Векторное соответствие

Операции между векторами пытаются найти соответствующий элемент в правом векторе для каждой записи в левом векторе. Существует два основных типа поведения при сопоставлении: один-к-одному и много-к-одному/один-ко-многим.

Ключевые слова векторного соответствия

Эти ключевые слова векторного соответствия позволяют выполнять сопоставление между рядами с разными наборами лейблов:

- on
- ignoring

Списки лейблов, предоставленные для ключевых слов соответствия, определяют, как векторы будут объединены. Примеры можно найти в разделах “Один-к-одному векторные соответствия” и “Много-к-одному и один-ко-многим векторные соответствия”.

Модификаторы групп

Эти модификаторы групп включают возможность много-к-одному/один-ко-многим векторного соответствия:

- group_left
- group_right

Списки лейблов могут быть предоставлены модификатору группы, которые содержат метки со стороны «один», которые должны быть включены в результирующие метрики.

Много-к-одному и **один-ко-многим** соответствия являются сложными случаями использования, которые должны быть тщательно продуманы. Часто правильное использование ignoring(<метки>) дает желаемый результат.

Модификаторы группировки могут использоваться только для сравнения и арифметических операций. Операторы and, unless и or по умолчанию сопоставляются со всеми возможными записями в правом векторе.

Один-к-одному векторные соответствия

Один-к-одному находит уникальную пару записей с каждой стороны операции. В случае по умолчанию, то есть при операции вида vector1 <оператор> vector2, две записи соответствуют, если они имеют одинаковый набор лейблов и соответствующих значений. Ключевое слово ignoring позволяет игнорировать определенные метки при сопоставлении, в то время как ключевое слово on позволяет сократить набор рассматриваемых лейблов до предоставленного списка:

```
<vector expr> <bin-op> ignoring(<label list>) <vector expr>
<vector expr> <bin-op> on(<label list>) <vector expr>
```

Примеры входных данных:

```
method_code:http_errors:rate5m{method="get", code="500"}  24
method_code:http_errors:rate5m{method="get", code="404"}  30
method_code:http_errors:rate5m{method="put", code="501"}  3
method_code:http_errors:rate5m{method="post", code="500"} 6
method_code:http_errors:rate5m{method="post", code="404"} 21

method:http_requests:rate5m{method="get"}  600
method:http_requests:rate5m{method="del"}  34
method:http_requests:rate5m{method="post"} 120
```

Пример запроса:

```
method_code:http_errors:rate5m{code="500"} / ignoring(code)
method:http_requests:rate5m
```

Это возвращает вектор результатов, содержащий долю HTTP-запросов со статус-кодом 500 для каждого метода, измеренную за последние 5 минут. Без ignoring(code) соответствия не было бы, так как метрики не имеют одинакового набора лейблов. Записи с методами put и del не имеют соответствия и не появятся в результате:

```
{method="get"}  0.04  # 24 / 600
{method="post"} 0.05  # 6 / 120
```

Много-к-одному и один-ко-многим векторные соответствия

Много-к-одному и один-ко-многим соответствия относятся к случаю, когда каждый элемент вектора со стороны «один» может соответствовать нескольким элементам со стороны «много». Это должно быть явно указано с использованием модификаторов group_left или group_right, где left/right определяет, какой вектор имеет более высокую мощность.

```
<vector expr> <bin-op> ignoring(<label list>) group_left(<label list>) <vector expr>
<vector expr> <bin-op> ignoring(<label list>) group_right(<label list>) <vector expr>
<vector expr> <bin-op> on(<label list>) group_left(<label list>) <vector expr>
<vector expr> <bin-op> on(<label list>) group_right(<label list>) <vector expr>
```

Список лейблов, предоставленный с модификатором группы, содержит дополнительные метки со стороны “один”, которые должны быть включены в результирующие метрики. Для on метка может появляться только в одном из списков. Каждый временной ряд результирующего вектора должен быть однозначно идентифицируемым.

Пример запроса:

```
method_code:http_errors:rate5m / ignoring(code) group_left method:http_requests:rate5m
```

В этом случае левый вектор содержит более одной записи для каждого значения метки method. Таким образом, мы указываем это с помощью group_left. Элементы с правой стороны теперь сопоставляются с несколькими элементами с той же меткой method слева:

```
{method="get", code="500"}  0.04   # 24 / 600
{method="get", code="404"}  0.05   # 30 / 600
{method="post", code="500"} 0.05   # 6 / 120
{method="post", code="404"} 0.175  # 21 / 120
```

Операторы агрегации

Deckhouse Prom++ поддерживает следующие встроенные операторы агрегации, которые могут быть использованы для агрегирования элементов одного мгновенного вектора, создавая новый вектор с меньшим количеством элементов и агрегированными значениями:

- sum (вычисление суммы по измерениям)

- min (выбор минимального значения по измерениям)
- max (выбор максимального значения по измерениям)
- avg (вычисление среднего значения по измерениям)
- group (все значения в результирующем векторе равны 1)
- stddev (вычисление стандартного отклонения по измерениям)
- stdvar (вычисление стандартной дисперсии по измерениям)
- count (подсчет количества элементов в векторе)
- count_values (подсчет количества элементов с одинаковым значением)
- bottomk (наименьшие k элементов по значению выборки)
- topk (наибольшие k элементов по значению выборки)
- quantile (вычисление φ-квантиля (0 ≤ φ ≤ 1) по измерениям)

Эти операторы могут быть использованы либо для агрегации по всем измерениям лейблов, либо для сохранения отдельных измерений с использованием предложений without или by. Эти предложения могут быть использованы до или после выражения.

```
<aggr-op> [without|by (<label list>)] ([parameter,] <vector expression>)
```

или

```
<aggr-op>([parameter,] <vector expression>) [without|by (<label list>)]
```

Список лейблов - это список не заключенных в кавычки лейблов, который может включать завершающую запятую, то есть оба варианта (label1, label2) и (label1, label2,) являются допустимым синтаксисом.

without удаляет перечисленные метки из результирующего вектора, сохраняя все другие метки в выходных данных. by делает обратное и удаляет метки, которые не перечислены в предложении by, даже если их значения лейблов идентичны между всеми элементами вектора.

Параметр требуется только для count_values, quantile, topk и bottomk.

count_values выводит один временной ряд для каждого уникального значения выборки. Каждый ряд имеет дополнительную метку. Имя этой метки задается параметром агрегации, а значение метки - это уникальное значение выборки. Значение каждого временного ряда - это количество раз, когда это значение выборки присутствовало.

topk и bottomk отличаются от других агрегаторов тем, что возвращают подмножество входных выборок, включая исходные метки. by и without используются только для группировки входного вектора.

quantile вычисляет φ-квантиль - значение, которое занимает позицию φ*N среди N значений метрик агрегированных измерений. φ предоставляется как параметр агрегации. Например, quantile(0.5, …) вычисляет медиану, quantile(0.95, …) - 95-й процентиль. Для φ = NaN возвращается NaN. Для φ < 0 возвращается -Inf. Для φ > 1 возвращается +Inf.

Пример:

Если метрика http_requests_total имела временные ряды, разбитые по меткам application, instance и group, мы могли бы вычислить общее количество наблюдаемых HTTP-запросов на приложение и группу по всем экземплярам с помощью:

```
sum without (instance) (http_requests_total)
```

Что эквивалентно:

```
sum by (application, group) (http_requests_total)
```

Если нас интересует только общее количество HTTP-запросов, которые мы видели во всех приложениях, мы могли бы просто написать:

```
sum(http_requests_total)
```

Чтобы подсчитать количество двоичных файлов, работающих с каждой версией сборки, мы могли бы написать:

```
count_values("version", build_version)
```

Чтобы получить 5 наибольших количеств HTTP-запросов по всем экземплярам, мы могли бы написать:

```
topk(5, http_requests_total)
```

Приоритет бинарных операторов

Следующий список показывает приоритет бинарных операторов в Deckhouse Prom++, от высшего к низшему:

1. ^
2. *, /, %, atan2
3. +, -
4. ==, !=, <=, <, >=, >
5. and, unless
6. or

Операторы с одинаковым приоритетом являются левоассоциативными. Например, 2 *3 % 2 эквивалентно (2* 3) % 2. Однако ^ является правоассоциативным, поэтому 2 ^ 3 ^ 2 эквивалентно 2 ^ (3 ^ 2).

Операторы для нативных гистограмм

Нативные гистограммы являются экспериментальной функцией. Прием нативных гистограмм должен быть включен через флаг функции. После приема нативных гистограмм их можно запрашивать (даже после отключения флага функции). Однако поддержка операторов для нативных гистограмм все еще очень ограничена.

Логические/множественные бинарные операторы работают как ожидается, даже если задействованы выборки гистограмм. Они только проверяют существование элемента вектора и не изменяют свое поведение в зависимости от типа выборки элемента (float или гистограмма). Оператор агрегации count работает аналогично.

Бинарные операторы + и - между двумя нативными гистограммами и операторы агрегации sum и avg для агрегации нативных гистограмм полностью поддерживаются. Даже если задействованные гистограммы имеют разную структуру корзин, корзины автоматически преобразуются соответствующим образом, чтобы операция могла быть выполнена. (При текущих поддерживаемых схемах корзин это всегда возможно.) Если оператор должен агрегировать смесь выборок гистограмм и float, соответствующий элемент вектора полностью удаляется из выходного вектора.

Бинарный оператор * работает между нативной гистограммой и float в любом порядке, а бинарный оператор / может быть использован между нативной гистограммой и float в этом порядке.

Все остальные операторы (и неупомянутые случаи для вышеуказанных операторов) не ведут себя осмысленным образом. Они либо рассматривают выборку гистограммы как если бы это была выборка float со значением 0, либо (в случае арифметических операций между скаляром и вектором) оставляют выборку гистограммы неизменной. Это поведение изменится на осмысленное до того, как нативные гистограммы станут стабильной функцией.

Функции PromQL

Некоторые функции имеют аргументы по умолчанию Некоторые функции имеют аргументы по умолчанию, например `year(v=vector(time()) instant-vector)`. Это означает, что есть один аргумент `v`, который является instant-вектором, и если он не указан, по умолчанию будет использоваться значение выражения `vector(time())`.

Примечания о нативных гистограммах (экспериментальная функция)

- Прием нативных гистограмм должен быть включен через флаг функциональности. Пока в TSDB не было записано ни одной нативной гистограммы, все функции будут вести себя как обычно.
- Функции, которые явно не упоминают нативные гистограммы в своей документации (см. ниже), будут игнорировать образцы гистограмм.
- Функции, которые уже работают с нативными гистограммами, могут изменить свое поведение в будущем.

- Если функции требуется одинаковая схема корзин между несколькими нативными гистограммами, с которыми она работает, она автоматически преобразует их соответствующим образом. (При текущих поддерживаемых схемах корзин это всегда возможно.)

abs()

`abs(v instant-vector)` возвращает входной вектор со всеми значениями образцов, преобразованными в их абсолютные значения.

absent()

`absent(v instant-vector)` возвращает пустой вектор, если переданный вектор содержит какие-либо элементы (числа с плавающей точкой или нативные гистограммы), и вектор с одним элементом со значением 1, если переданный вектор не содержит элементов.

Это полезно для создания предупреждений, когда не существует временных рядов для заданной комбинации имени метрики и лейблов.

Примеры:

```
absent(nonexistent{job="myjob"})
# => {job="myjob"}

absent(nonexistent{job="myjob",instance=~".*"})
# => {job="myjob"}

absent(sum(nonexistent{job="myjob"}))
# => {}
```

В первых двух примерах absent() пытается интеллектуально вывести метки выходного вектора из входного вектора.

absent_over_time()

absent_over_time(v range-vector) возвращает пустой вектор, если переданный range-вектор содержит какие-либо элементы (числа с плавающей точкой или нативные гистограммы), и вектор с одним элементом со значением 1, если переданный вектор не содержит элементов.

Это полезно для создания предупреждений, когда в течение определенного времени не существует временных рядов для заданной комбинации имени метрики и лейблов.

Примеры:

```
absent_over_time(nonexistent{job="myjob"}[1h])
# => {job="myjob"}

absent_over_time(nonexistent{job="myjob",instance=~".*"}[1h])
# => {job="myjob"}

absent_over_time(sum(nonexistent{job="myjob"})[1h:])
# => {}
```

В первых двух примерах absent_over_time() пытается интеллектуально вывести метки выходного вектора из входного вектора.

ceil()

ceil(v instant-vector) округляет значения всех элементов в v до ближайшего целого числа в большую сторону.

changes()

Для каждого входного временного ряда changes(v range-vector) возвращает количество изменений его значения в заданном временном диапазоне как instant-вектор.

clamp()

clamp(v instant-vector, min scalar, max scalar) ограничивает значения всех элементов в v нижним пределом min и верхним пределом max.

Особые случаи: Возвращает пустой вектор, если min > max Возвращает NaN, если min или max равно NaN

clamp_max()

clamp_max(v instant-vector, max scalar) ограничивает значения всех элементов в v верхним пределом max.

clamp_min()

clamp_min(v instant-vector, min scalar) ограничивает значения всех элементов в v нижним пределом min.

day_of_month()

day_of_month(v=vector(time()) instant-vector) возвращает день месяца для каждого из заданных времен в UTC. Возвращаемые значения находятся в диапазоне от 1 до 31.

day_of_week()

day_of_week(v=vector(time()) instant-vector) возвращает день недели для каждого из заданных времен в UTC. Возвращаемые значения находятся в диапазоне от 0 до 6, где 0 означает воскресенье и т.д.

day_of_year()

day_of_year(v=vector(time()) instant-vector) возвращает день года для каждого из заданных времен в UTC. Возвращаемые значения находятся в диапазоне от 1 до 365 для невисокосных лет и от 1 до 366 для високосных лет.

days_in_month()

days_in_month(v=vector(time()) instant-vector) возвращает количество дней в месяце для каждого из заданных времен в UTC. Возвращаемые значения находятся в диапазоне от 28 до 31.

delta()

delta(v range-vector) вычисляет разницу между первым и последним значением каждого элемента временного ряда в range-векторе v, возвращая instant-вектор с полученными дельтами и эквивалентными метками. Дельта экстраполируется на весь временной диапазон, указанный в селекторе range-вектора, поэтому результат может быть нецелым, даже если все значения образцов целые.

Следующий пример выражения возвращает разницу в температуре CPU между текущим моментом и 2 часами назад:

```
delta(cpu_temp_celsius{host="zeus"}[2h])
```

delta работает с нативными гистограммами, вычисляя новую гистограмму, где каждый компонент (сумма и количество наблюдений, корзины) представляет собой разницу между соответствующими компонентами первой и последней нативной гистограммы в v. Однако любой элемент в v, содержащий смесь чисел с плавающей точкой и образцов нативных гистограмм в диапазоне, будет отсутствовать в результирующем векторе.

delta следует использовать только с gauge-метриками и нативными гистограммами, где компоненты ведут себя как gauge (так называемые gauge-гистограммы).

deriv()

deriv(v range-vector) вычисляет производную в секунду временного ряда в range-векторе v, используя простую линейную регрессию. Range-вектор должен содержать как минимум два образца для выполнения расчета. Если в range-векторе встречаются +Inf или -Inf, вычисленные значения наклона и смещения будут NaN.

deriv следует использовать только с gauge-метриками.

exp()

exp(v instant-vector) вычисляет экспоненциальную функцию для всех элементов в v. Особые случаи:

Exp(+Inf) = +Inf Exp(NaN) = NaN

floor() floor(v instant-vector) округляет значения всех элементов в v до ближайшего целого числа в меньшую сторону.

histogram_avg()

Эта функция работает только с нативными гистограммами, которые являются экспериментальной функцией. Поведение этой функции может измениться в будущих версиях Deckhouse Prom++, включая ее удаление из PromQL.

histogram_avg(v instant-vector) возвращает среднее арифметическое наблюдаемых значений, хранящихся в нативной гистограмме. Образцы, не являющиеся нативными гистограммами, игнорируются и не появляются в возвращаемом векторе.

Используйте histogram_avg, как показано ниже, для вычисления средней продолжительности запроса за 5-минутное окно из нативной гистограммы:

```
histogram_avg(rate(http_request_duration_seconds[5m]))
```

Что эквивалентно следующему запросу:

```
  histogram_sum(rate(http_request_duration_seconds[5m]))
/
  histogram_count(rate(http_request_duration_seconds[5m]))
```

histogram_count() и histogram_sum()

Обе функции работают только с нативными гистограммами, которые являются экспериментальной функцией. Поведение этих функций может измениться в будущих версиях Deckhouse Prom++, включая их удаление из PromQL.

histogram_count(v instant-vector) возвращает количество наблюдений, хранящихся в нативной гистограмме. Образцы, не являющиеся нативными гистограммами, игнорируются и не появляются в возвращаемом векторе.

Аналогично, histogram_sum(v instant-vector) возвращает сумму наблюдений, хранящихся в нативной гистограмме.

Используйте histogram_count следующим образом для вычисления rate наблюдений (в данном случае соответствующего «запросам в секунду») из нативной гистограммы:

```
histogram_count(rate(http_request_duration_seconds[10m]))
```

histogram_fraction()

Эта функция работает только с нативными гистограммами, которые являются экспериментальной функцией. Поведение этой функции может измениться в будущих версиях Deckhouse Prom++, включая ее удаление из PromQL.

Для нативной гистограммы histogram_fraction(lower scalar, upper scalar, v instant-vector) возвращает предполагаемую долю наблюдений между заданными нижним и верхним значениями. Образцы, не являющиеся нативными гистограммами, игнорируются и не появляются в возвращаемом векторе.

Например, следующее выражение вычисляет долю HTTP-запросов за последний час, которые заняли 200 мс или меньше:

```
histogram_fraction(0, 0.2, rate(http_request_duration_seconds[1h]))
```

Ошибка оценки зависит от разрешения базовой нативной гистограммы и того, насколько точно заданные границы совпадают с границами корзин в гистограмме.

+Inf и -Inf являются допустимыми значениями границ. Например, если гистограмма в приведенном выше выражении включала отрицательные наблюдения (чего не должно быть для длительностей запросов), соответствующей нижней границей для включения всех наблюдений меньше или равных 0,2 будет -Inf, а не 0.

Вопрос о том, являются ли заданные границы включительными или исключительными, актуален только если заданные границы точно совпадают с границами корзин в базовой нативной гистограмме. В этом случае поведение зависит от определения схемы гистограммы. В настоящее время поддерживаемые схемы имеют включительные верхние границы и исключительные нижние границы для положительных значений (и наоборот для отрицательных значений). Без точного совпадения границ функция использует линейную интерполяцию для оценки доли. С возникающей неопределенностью становится неважным, являются ли границы включительными или исключительными.

histogram_quantile()

histogram_quantile(φ scalar, b instant-vector) вычисляет φ-квантиль (0 ≤ φ ≤ 1) из классической гистограммы или нативной гистограммы. (См. гистограммы и сводки для подробного объяснения φ-квантилей и использования типа метрики (классической) гистограммы в целом.)

Обратите внимание, что нативные гистограммы являются экспериментальной функцией. Поведение этой функции при работе с нативными гистограммами может измениться в будущих версиях Deckhouse Prom++.

Образцы с плавающей точкой в b считаются счетчиками наблюдений в каждой корзине одной или нескольких классических гистограмм. Каждый образец с плавающей точкой должен иметь метку le, где значение метки обозначает включительную верхнюю границу корзины. (Образцы с плавающей точкой без такой метки тихо игнорируются.) Другие метки и имя метрики используются для идентификации корзин, принадлежащих каждой классической гистограмме. Тип метрики гистограмма автоматически предоставляет временные ряды с суффиксом _bucket и соответствующими метками.

Образцы нативных гистограмм в b обрабатываются каждый отдельно как отдельная гистограмма для вычисления квантиля. Пока не возникает конфликтов имен, b может содержать смесь классических и нативных гистограмм. Используйте функцию rate() для указания временного окна для вычисления квантиля.

Пример: Метрика гистограммы называется http_request_duration_seconds (и, следовательно, имя метрики для корзин классической гистограммы - http_request_duration_seconds_bucket). Чтобы вычислить 90-й процентиль длительностей запросов за последние 10 минут, используйте следующее выражение в случае, если http_request_duration_seconds является классической гистограммой:

```
histogram_quantile(0.9, rate(http_request_duration_seconds_bucket[10m]))
```

Для нативной гистограммы используйте следующее выражение:

```
histogram_quantile(0.9, rate(http_request_duration_seconds[10m]))
```

Квантиль вычисляется для каждой комбинации лейблов в http_request_duration_seconds. Для агрегации используйте агрегатор sum() вокруг функции rate(). Поскольку метка le требуется histogram_quantile() для работы с классическими гистограммами, она должна быть включена в предложение by. Следующее выражение агрегирует 90-й процентиль по job для классических гистограмм:

```
histogram_quantile(0.9, sum by (job, le)
(rate(http_request_duration_seconds_bucket[10m])))
```

При агрегации нативных гистограмм выражение упрощается до:

```
histogram_quantile(0.9, sum by (job) (rate(http_request_duration_seconds[10m])))
```

Для агрегации всех классических гистограмм укажите только метку le:

```
histogram_quantile(0.9, sum by (le) (rate(http_request_duration_seconds_bucket[10m])))
```

С нативными гистограммами агрегация всего работает как обычно без предложения by:

```
histogram_quantile(0.9, sum(rate(http_request_duration_seconds[10m])))
```

Функция histogram_quantile() интерполирует значения квантилей, предполагая линейное распределение внутри корзины.

Если в b 0 наблюдений, возвращается NaN. Для φ < 0 возвращается -Inf. Для φ > 1 возвращается +Inf. Для φ = NaN возвращается NaN.

Следующее актуально только для классических гистограмм: Если b содержит менее двух корзин, возвращается NaN. Самая верхняя корзина должна иметь верхнюю границу +Inf. (В противном случае возвращается NaN.) Если квантиль находится в самой верхней корзине, возвращается верхняя граница второй по высоте корзины. Нижний предел самой нижней корзины предполагается равным 0, если верхняя граница этой корзины больше 0. В этом случае применяется обычная линейная интерполяция внутри этой корзины. В противном случае для квантилей, находящихся в самой нижней корзине, возвращается верхняя граница самой нижней корзины.

Вы можете использовать histogram_quantile(0, v instant-vector) для получения предполагаемого минимального значения, хранящегося в гистограмме. Вы можете использовать histogram_quantile(1, v instant-vector) для получения предполагаемого максимального значения, хранящегося в гистограмме. Корзины классических гистограмм являются кумулятивными. Поэтому всегда должно выполняться следующее: Счетчики в корзинах монотонно возрастают (строго не убывают). Отсутствие наблюдений между верхними пределами двух последовательных корзин приводит к равным счетчикам в этих двух корзинах.

Однако проблемы с точностью чисел с плавающей точкой (например, небольшие расхождения, вызванные вычислением корзин с sum(rate(…))) или недопустимые данные могут нарушить эти предположения. В этом случае histogram_quantile не сможет вернуть осмысленные результаты. Чтобы устранить проблему, histogram_quantile предполагает, что крошечные относительные различия между последовательными корзинами возникают из-за ошибок точности чисел с плавающей точкой, и игнорирует их. (Порог для игнорирования разницы между двумя корзинами составляет одну триллионную (1e-12) от суммы обеих корзин.)

Кроме того, если после этой корректировки все еще есть немонотонные счетчики корзин, они увеличиваются до значения предыдущих корзин для обеспечения монотонности. Последнее является свидетельством реальной проблемы с входными данными и поэтому помечается информационной аннотацией “input to histogram_quantile needed to be fixed for monotonicity”. Если вы столкнулись с этой аннотацией, вам следует найти и устранить источник недопустимых данных.

histogram_stddev() и histogram_stdvar()

Обе функции работают только с нативными гистограммами, которые являются экспериментальной функцией. Поведение этих функций может измениться в будущих версиях Deckhouse Prom++, включая их удаление из PromQL.

histogram_stddev(v instant-vector) возвращает предполагаемое стандартное отклонение наблюдений в нативной гистограмме, основанное на среднем геометрическом корзин, в которых лежат наблюдения. Образцы, не являющиеся нативными гистограммами, игнорируются и не появляются в возвращаемом векторе.

Аналогично, histogram_stdvar(v instant-vector) возвращает предполагаемую стандартную дисперсию наблюдений в нативной гистограмме.

holt_winters()

holt_winters(v range-vector, sf scalar, tf scalar) производит сглаженное значение для временного ряда на основе диапазона в v. Чем ниже коэффициент сглаживания sf, тем больше важность придается старым данным. Чем выше коэффициент тренда tf, тем больше учитываются тренды в данных. И sf, и tf должны быть между 0 и 1.

holt_winters следует использовать только с gauge-метриками.

hour()

hour(v=vector(time()) instant-vector) возвращает час дня для каждого из заданных времен в UTC. Возвращаемые значения находятся в диапазоне от 0 до 23.

idelta()

idelta(v range-vector) вычисляет разницу между последними двумя образцами в range-векторе v, возвращая instant-вектор с полученными дельтами и эквивалентными метками.

idelta следует использовать только с gauge-метриками.

increase()

increase(v range-vector) вычисляет увеличение временного ряда в range-векторе. Разрывы монотонности (такие как сбросы счетчиков из-за перезапусков целей) автоматически корректируются. Увеличение экстраполируется на весь временной диапазон, указанный в селекторе range-вектора, поэтому результат может быть нецелым, даже если счетчик увеличивается только на целые значения.

Следующий пример выражения возвращает количество HTTP-запросов, измеренное за последние 5 минут, для каждого временного ряда в range-векторе:

```
increase(http_requests_total{job="api-server"}[5m])
```

increase работает с нативными гистограммами, вычисляя новую гистограмму, где каждый компонент (сумма и количество наблюдений, корзины) представляет собой увеличение между соответствующими компонентами первой и последней нативной гистограммы в v. Однако любой элемент в v, содержащий смесь чисел с плавающей точкой и образцов нативных гистограмм в диапазоне, будет отсутствовать в результирующем векторе.

increase следует использовать только с counter-метриками и нативными гистограммами, где компоненты ведут себя как счетчики. Это синтаксический сахар для rate(v), умноженного на количество секунд в указанном временном окне, и в первую очередь предназначен для удобства чтения человеком. Используйте rate в правилах записи, чтобы увеличения отслеживались последовательно в расчете на секунду.

irate()

irate(v range-vector) вычисляет мгновенную скорость увеличения в секунду временного ряда в range-векторе. Это основано на последних двух точках данных. Разрывы монотонности (такие как сбросы счетчиков из-за перезапусков целей) автоматически корректируются.

Следующий пример выражения возвращает скорость HTTP-запросов в секунду, просматривая до 5 минут назад для двух последних точек данных, для каждого временного ряда в range-векторе:

```
irate(http_requests_total{job="api-server"}[5m])
```

irate следует использовать только при построении графиков волатильных, быстро меняющихся счетчиков. Используйте rate для предупреждений и медленно меняющихся счетчиков, так как кратковременные изменения скорости могут сбросить предложение FOR, а графики, полностью состоящие из редких пиков, трудно читать.

Обратите внимание, что при комбинировании irate() с оператором агрегации (например, sum()) или функцией, агрегирующей по времени (любая функция, оканчивающаяся на _over_time), всегда сначала применяйте irate(), затем агрегируйте. В противном случае irate() не сможет обнаружить сбросы счетчиков при перезапусках вашей цели.

label_join()

Для каждого временного ряда в v, label_join(v instant-vector, dst_label string, separator string, src_label_1 string, src_label_2 string, …) объединяет все значения всех src_labels с использованием separator и возвращает временной ряд с меткой dst_label, содержащей объединенное значение. В этой функции может быть любое количество src_labels.

label_join одинаково работает с образцами чисел с плавающей точкой и гистограммами.

Этот пример вернет вектор с каждым временным рядом, имеющим метку foo со значением a,b,c, добавленным к нему:

```
label_join(up{job="api-server",src1="a",src2="b",src3="c"}, "foo", ",", "src1",
"src2", "src3")
```

label_replace()

Для каждого временного ряда в v, label_replace(v instant-vector, dst_label string, replacement string, src_label string, regex string) сопоставляет регулярное выражение regex со значением метки

src_label. Если совпадение найдено, значение метки dst_label в возвращаемом временном ряде будет заменой replacement, вместе с исходными метками во входных данных. Группы захвата в регулярном выражении могут быть ссылаться с помощью $1, $2 и т.д. Именованные группы захвата в регулярном выражении могут быть ссылаться с помощью $name (где name - имя группы захвата). Если регулярное выражение не совпадает, временной ряд возвращается без изменений.

label_replace одинаково работает с образцами чисел с плавающей точкой и гистограммами.

Этот пример вернет временные ряды со значениями a:c у метки service и a у метки foo:

```
label_replace(up{job="api-server",service="a:c"}, "foo", "$1", "service", "(.*):.*")
```

Этот второй пример имеет тот же эффект, что и первый пример, и иллюстрирует использование именованных групп захвата:

```
label_replace(up{job="api-server",service="a:c"}, "foo", "$name", "service", "(?
P<name>.*):(?P<version>.*)")
```

ln()

ln(v instant-vector) вычисляет натуральный логарифм для всех элементов в v. Особые случаи:

```
ln(+Inf) = +Inf
ln(0) = -Inf
ln(x < 0) = NaN
ln(NaN) = NaN
```

log2()

log2(v instant-vector) вычисляет двоичный логарифм для всех элементов в v. Особые случаи эквивалентны таковым в ln.

log10()

log10(v instant-vector) вычисляет десятичный логарифм для всех элементов в v. Особые случаи эквивалентны таковым в ln.

minute()

minute(v=vector(time()) instant-vector) возвращает минуту часа для каждого из заданных времен в UTC. Возвращаемые значения находятся в диапазоне от 0 до 59.

month()

month(v=vector(time()) instant-vector) возвращает месяц года для каждого из заданных времен в UTC. Возвращаемые значения находятся в диапазоне от 1 до 12, где 1 означает январь и т.д.

predict_linear()

predict_linear(v range-vector, t scalar) предсказывает значение временного ряда через t секунд от текущего момента, основываясь на range-векторе v, используя простую линейную регрессию. Range-вектор должен содержать как минимум два образца для выполнения расчета. Если в range-векторе встречаются +Inf или -Inf, вычисленные значения наклона и смещения будут NaN.

predict_linear следует использовать только с gauge-метриками.

rate()

rate(v range-vector) вычисляет среднюю скорость увеличения в секунду временного ряда в range-векторе. Разрывы монотонности (такие как сбросы счетчиков из-за перезапусков целей) автоматически корректируются. Кроме того, расчет экстраполируется до концов временного диапазона, позволяя пропущенные сборы или несовершенное выравнивание циклов сбора с периодом времени диапазона.

Следующий пример выражения возвращает скорость HTTP-запросов в секунду, измеренную за последние 5 минут, для каждого временного ряда в range-векторе:

```
rate(http_requests_total{job="api-server"}[5m])
```

rate работает с нативными гистограммами, вычисляя новую гистограмму, где каждый компонент (сумма и количество наблюдений, корзины) представляет собой скорость увеличения между соответствующими компонентами первой и последней нативной гистограммы в v. Однако любой элемент в v, содержащий смесь чисел с плавающей точкой и образцов нативных гистограмм в диапазоне, будет отсутствовать в результирующем векторе.

rate следует использовать только с counter-метриками и нативными гистограммами, где компоненты ведут себя как счетчики. Он лучше всего подходит для предупреждений и построения графиков медленно меняющихся счетчиков.

Обратите внимание, что при комбинировании rate() с оператором агрегации (например, sum()) или функцией, агрегирующей по времени (любая функция, оканчивающаяся на _over_time), всегда сначала применяйте rate(), затем агрегируйте. В противном случае rate() не сможет обнаружить сбросы счетчиков при перезапусках вашей цели.

resets()

Для каждого входного временного ряда resets(v range-vector) возвращает количество сбросов счетчика в пределах заданного временного диапазона как instant-вектор. Любое уменьшение значения между двумя последовательными образцами с плавающей точкой интерпретируется как сброс счетчика. Сброс в нативной гистограмме обнаруживается более сложным образом: любое уменьшение в любой корзине, включая нулевую корзину, или в количестве наблюдений составляет сброс счетчика, но также исчезновение любой ранее заполненной корзины, увеличение разрешения корзин или уменьшение ширины нулевой корзины.

resets следует использовать только с counter-метриками и counter-подобными нативными гистограммами.

Если range-вектор содержит смесь образцов с плавающей точкой и гистограмм для одного и того же ряда, сбросы счетчиков обнаруживаются отдельно, и их числа складываются. Изменение от образца с плавающей точкой к образцу гистограммы не считается сбросом счетчика. Каждый образец с плавающей точкой сравнивается со следующим образцом с плавающей точкой, и каждая гистограмма сравнивается со следующей гистограммой.

round()

round(v instant-vector, to_nearest=1 scalar) округляет значения всех элементов в v до ближайшего целого числа. Ничьи разрешаются округлением вверх. Необязательный аргумент to_nearest позволяет указать ближайшее кратное, до которого следует округлять значения образцов. Это кратное может быть также дробным.

scalar()

При наличии входного вектора с одним элементом scalar(v instant-vector) возвращает значение образца этого единственного элемента как скаляр. Если входной вектор не содержит ровно один элемент, scalar вернет NaN.

sgn()

sgn(v instant-vector) возвращает вектор со всеми значениями образцов, преобразованными в их знак, определенный следующим образом: 1, если v положительный, -1, если v отрицательный, и 0, если v равен нулю.

sort()

sort(v instant-vector) возвращает элементы вектора, отсортированные по их значениям образцов в порядке возрастания. Нативные гистограммы сортируются по их сумме наблюдений.

Обратите внимание, что sort влияет только на результаты мгновенных запросов, так как результаты запросов диапазона всегда имеют фиксированный порядок вывода.

sort_desc()

То же, что и sort, но сортирует в порядке убывания.

Как и sort, sort_desc влияет только на результаты мгновенных запросов, так как результаты запросов диапазона всегда имеют фиксированный порядок вывода.

sort_by_label()

Эта функция должна быть включена через флаг функциональности –enable-feature=promql-experimental-functions.

sort_by_label(v instant-vector, label string, …) возвращает элементы вектора, отсортированные по их значениям лейблов и значениям образцов в случае равенства значений лейблов, в порядке возрастания.

Обратите внимание, что функции сортировки по меткам влияют только на результаты мгновенных запросов, так как результаты запросов диапазона всегда имеют фиксированный порядок вывода.

Эта функция использует естественный порядок сортировки.

sort_by_label_desc()

Эта функция должна быть включена через флаг функциональности –enable-feature=promql-experimental-functions.

То же, что и sort_by_label, но сортирует в порядке убывания.

Обратите внимание, что функции сортировки по меткам влияют только на результаты мгновенных запросов, так как результаты запросов диапазона всегда имеют фиксированный порядок вывода.

Эта функция использует естественный порядок сортировки.

sqrt()

sqrt(v instant-vector) вычисляет квадратный корень из всех элементов в v.

time()

time() возвращает количество секунд с 1 января 1970 года UTC. Обратите внимание, что это фактически не возвращает текущее время, а время, в которое должно быть вычислено выражение.

timestamp()

timestamp(v instant-vector) возвращает временную метку каждого из образцов заданного вектора в виде количества секунд с 1 января 1970 года UTC. Также работает с образцами гистограмм.

vector()

vector(s scalar) возвращает скаляр s как вектор без лейблов.

year()

year(v=vector(time()) instant-vector) возвращает год для каждого из заданных времен в UTC.

_over_time()

Следующие функции позволяют агрегировать каждый ряд заданного range-вектора по времени и возвращать instant-вектор с результатами агрегации по рядам:

- avg_over_time(range-vector): среднее значение всех точек в указанном интервале.
- min_over_time(range-vector): минимальное значение всех точек в указанном интервале.

- max_over_time(range-vector): максимальное значение всех точек в указанном интервале.
- sum_over_time(range-vector): сумма всех значений в указанном интервале.
- count_over_time(range-vector): количество всех значений в указанном интервале.
- quantile_over_time(scalar, range-vector): φ-квантиль (0 ≤ φ ≤ 1) значений в указанном интервале.
- stddev_over_time(range-vector): стандартное отклонение совокупности значений в указанном интервале.
- stdvar_over_time(range-vector): стандартная дисперсия совокупности значений в указанном интервале.
- last_over_time(range-vector): значение самой последней точки в указанном интервале.
- present_over_time(range-vector): значение 1 для любого ряда в указанном интервале.
- Если установлен флаг функциональности –enable-feature=promql-experimental-functions, доступны следующие дополнительные функции:
- mad_over_time(range-vector): медианное абсолютное отклонение всех точек в указанном интервале.

Обратите внимание, что все значения в указанном интервале имеют одинаковый вес в агрегации, даже если значения не равноудалены в течение интервала.

avg_over_time, sum_over_time, count_over_time, last_over_time и present_over_time обрабатывают нативные гистограммы как ожидается. Все остальные функции игнорируют образцы гистограмм.

Тригонометрические функции

Тригонометрические функции работают в радианах:

- acos(v instant-vector): вычисляет арккосинус всех элементов в v (особые случаи).
- acosh(v instant-vector): вычисляет гиперболический арккосинус всех элементов в v (особые случаи).
- asin(v instant-vector): вычисляет арксинус всех элементов в v (особые случаи).
- asinh(v instant-vector): вычисляет гиперболический арксинус всех элементов в v (особые случаи).
- atan(v instant-vector): вычисляет арктангенс всех элементов в v (особые случаи).
- atanh(v instant-vector): вычисляет гиперболический арктангенс всех элементов в v (особые случаи).
- cos(v instant-vector): вычисляет косинус всех элементов в v (особые случаи).
- cosh(v instant-vector): вычисляет гиперболический косинус всех элементов в v (особые случаи).
- sin(v instant-vector): вычисляет синус всех элементов в v (особые случаи).
- sinh(v instant-vector): вычисляет гиперболический синус всех элементов в v (особые случаи).
- tan(v instant-vector): вычисляет тангенс всех элементов в v (особые случаи).
- tanh(v instant-vector): вычисляет гиперболический тангенс всех элементов в v (особые случаи).

Следующие функции полезны для преобразования между градусами и радианами:

- deg(v instant-vector): преобразует радианы в градусы для всех элементов в v.
- pi(): возвращает pi.
- rad(v instant-vector): преобразует градусы в радианы для всех элементов в v.

Примеры

Deckhouse Prom++ предоставляет мощный язык запросов PromQL, позволяющий извлекать и анализировать данные временных рядов. Ниже приведены некоторые примеры использования запросов в Deckhouse Prom++.

Простой выбор временных рядов

Возврат всех временных рядов с метрикой `http_requests_total`:

```
http_requests_total
```

Возврат всех временных рядов с метрикой `http_requests_total` и заданными метками `job` и `handler`:

```
http_requests_total{job="apiserver", handler="/api/comments"}
```

Возврат диапазона времени (в данном случае 5 минут до времени запроса) для того же вектора, превращая его в **диапазонный вектор**:

```
http_requests_total{job="apiserver", handler="/api/comments"}[5m]
```

Обратите внимание, что выражение, приводящее к диапазонному вектору, нельзя отобразить напрямую на графике, но его можно просмотреть в табличном (“Консоль”) представлении браузера выражений.

Используя регулярные выражения, можно выбрать временные ряды только для заданий, чьи имена соответствуют определённому шаблону, в данном случае, все задания, оканчивающиеся на `server`:

```
http_requests_total{job=~".*server"}
```

Все регулярные выражения в Deckhouse Prom++ используют синтаксис RE2.

Чтобы выбрать все HTTP-коды состояния, кроме 4xx, можно выполнить:

```
http_requests_total{status!~"4.."}
```

Подзапросы

Возврат 5-минутного rate метрики `http_requests_total` за последние 30 минут с разрешением в 1 минуту:

```
rate(http_requests_total[5m])[30m:1m]
```

Это пример вложенного подзапроса. Подзапрос для функции `deriv` использует разрешение по умолчанию. Обратите внимание, что нецелесообразно использовать подзапросы без необходимости.

```
max_over_time(deriv(rate(distance_covered_total[5s])[30s:5s])[10m:])
```

Использование функций, операторов и т.д

Возвращает rate-показатель в секунду для всех временных рядов с метрикой http_requests_total, измеренный за последние 5 минут:

```
rate(http_requests_total[5m])
```

Предполагая, что все временные ряды http_requests_total имеют метки job (разделение по названию job) и instance (разделение по экземпляру job), мы можем просуммировать rate всех экземпляров, чтобы получить меньше выходных временных рядов, сохраняя при этом измерение job:

```
sum by (job) (
  rate(http_requests_total[5m])
)
```

Если у нас есть две разные метрики с одинаковыми метками измерений, мы можем применить к ним бинарные операторы, и элементы с обеих сторон с одинаковым набором лейблов будут сопоставлены и переданы на выход. Например, это выражение возвращает неиспользуемую память в MiB для каждого экземпляра (в гипотетическом планировщике кластера, предоставляющем эти метрики о запущенных экземплярах):

```
(instance_memory_limit_bytes - instance_memory_usage_bytes) / 1024 / 1024
```

То же выражение, но агрегированное по приложению, можно записать так:

```
sum by (app, proc) (
  instance_memory_limit_bytes - instance_memory_usage_bytes
) / 1024 / 1024
```

Если бы этот гипотетический планировщик кластера предоставлял метрики использования CPU для каждого экземпляра в следующем виде:

```
instance_cpu_time_ns{app="lion", proc="web", rev="34d0f99", env="prod", job="cluster-
manager"}
instance_cpu_time_ns{app="elephant", proc="worker", rev="34d0f99", env="prod",
job="cluster-manager"}
instance_cpu_time_ns{app="turtle", proc="api", rev="4d3a513", env="prod",
job="cluster-manager"}
instance_cpu_time_ns{app="fox", proc="widget", rev="4d3a513", env="prod",
job="cluster-manager"}
```

…мы могли бы получить топ-3 пользователей CPU, сгруппированных по приложению (app) и типу процесса (proc), следующим образом:

```
topk(3, sum by (app, proc) (rate(instance_cpu_time_ns[5m])))
```

Предполагая, что эта метрика содержит один временной ряд на каждый запущенный экземпляр, вы можете подсчитать количество работающих экземпляров по приложениям следующим образом:

```
count by (app) (instance_cpu_time_ns)
```

Remote read API

Deckhouse Prom++ предоставляет API для remote read, доступный по эндпоинту `/api/v1/read`. Этот интерфейс ожидает сжатие с помощью алгоритма Snappy. Подробнее про API можно узнать в [репозитории Prom++]

> **Примечание:** Данный API в настоящее время не считается стабильным и может изменяться даже между минорными версиями Deckhouse Prom++.

Samples

Этот эндпоинт возвращает сообщение, содержащее список необработанных выборок, соответствующих запрашиваемому запросу.

Streamed Chunks

Стриминговые чанки используют алгоритм XOR, вдохновлённый сжатием Gorilla, для кодирования чанков. Однако он обеспечивает разрешение до миллисекунды вместо секунды.

HTTP API

Текущий стабильный HTTP API доступен по адресу /api/v1 на сервере Deckhouse Prom++. Любые обратно совместимые дополнения будут добавляться к этой конечной точке.

Обзор формата

Формат ответа API — JSON. Каждый успешный запрос к API возвращает код состояния 2xx.

Некорректные запросы, которые достигают обработчиков API, возвращают объект ошибки JSON и один из следующих кодов ответа HTTP:

- 400 Bad Request — когда параметры отсутствуют или некорректны.
- 422 Unprocessable Entity — когда выражение не может быть выполнено (RFC4918).
- 503 Service Unavailable — когда запросы превышают время ожидания или прерываются. Другие коды, не начинающиеся с 2xx, могут возвращаться для ошибок, возникающих до достижения конечной точки API.

Массив предупреждений может быть возвращен, если есть ошибки, которые не препятствуют выполнению запроса. Все успешно собранные данные будут возвращены в поле `data`.

Формат оболочки JSON-ответа выглядит следующим образом:

```
{
  "status": "success" | "error",
  "data": <data>,
  // Устанавливается только если status = "error". Поле data может всё ещё содержать
дополнительные данные.
  "errorType": "<string>",
  "error": "<string>",
  // Только если при выполнении запроса возникли предупреждения. Данные всё равно
будут в поле data.
  "warnings": ["<string>"]
}
```

Общие параметры определены следующим образом

- `<rfc3339 | unix_timestamp>`: Входные метки времени могут быть указаны либо в формате RFC3339, либо как Unix-метка времени в секундах, с необязательными десятичными знаками для точности до долей секунды. Выходные метки времени всегда представлены как Unix-метки времени в секундах.
- `<series_selector>`: Селекторы временных рядов Deckhouse Prom++, например, http_requests_total или http_requests_total{method=~"(GET|POST)"}, которые необходимо кодировать в URL.
- `<duration>`: Строки длительности Deckhouse Prom++. Например, 5m означает длительность в 5 минут.
- `<bool>`: Логические значения (строки true и false).

Примечание: Имена параметров запроса, которые могут повторяться, заканчиваются на [].

Запросы выражений

Выражения языка запросов могут быть вычислены для одного момента времени или за диапазон времени. В следующих разделах описаны конечные точки API для каждого типа запросов выражений.

Instant запросы

Следующая конечная точка вычисляет мгновенный запрос в определенный момент времени:

```
GET /api/v1/query
POST /api/v1/query
```

Параметры URL-запроса:

```
query=<string>: Строка выражения Deckhouse Prom++.
time=<rfc3339 | unix_timestamp>: Метка времени для вычисления. Необязательный.
timeout=<duration>: Таймаут вычисления. Необязательный. По умолчанию и ограничивается
значением флага -query.timeout.
```

Текущее время сервера используется, если параметр time не указан.

Вы можете закодировать эти параметры в теле запроса, используя метод POST и заголовок Content-Type: application/x-www-form-urlencoded. Это полезно при указании большого запроса, который может превысить лимиты символов URL на стороне сервера.

Раздел data результата запроса имеет следующий формат:

```
{
  "resultType": "matrix" | "vector" | "scalar" | "string",
  "result": <value>
}
```

относится к данным результата запроса, которые имеют различный формат в зависимости от resultType. См. форматы результатов выражений.

Следующий пример вычисляет выражение up на момент времени 2015-07-01T20:10:51.781Z:

```
curl 'http://localhost:9090/api/v1/query?query=up&time=2015-07-01T20:10:51.781Z'
```

Пример ответа:

```
{
   "status": "success",
   "data": {
      "resultType": "vector",
      "result": [
         {
            "metric": {
               "__name__": "up",
               "job": "Deckhouse Prom++",
               "instance": "localhost:9090"
            },
            "value": [ 1435781451.781, "1" ]
         },
         {
            "metric": {
               "__name__": "up",
               "job": "node",
               "instance": "localhost:9100"
            },
            "value": [ 1435781451.781, "0" ]
         }
      ]
   }
}
```

Запросы по диапазону

Следующая конечная точка вычисляет выражение за диапазон времени:

```
GET /api/v1/query_range
POST /api/v1/query_range
```

Параметры URL-запроса:

- query=: Строка выражения Deckhouse Prom++.
- start=<rfc3339 | unix_timestamp>: Начальная метка времени (включительно).
- end=<rfc3339 | unix_timestamp>: Конечная метка времени (включительно).
- step=<duration | float>: Шаг разрешения запроса в формате длительности или числе секунд с плавающей точкой.
- timeout=: Таймаут вычисления. Необязательный. По умолчанию и ограничивается значением флага -query.timeout.

Вы можете закодировать эти параметры в теле запроса, используя метод POST и заголовок Content-Type: application/x-www-form-urlencoded. Это полезно при указании большого запроса.

Раздел data результата запроса имеет следующий формат:

```
{
  "resultType": "matrix",
  "result": <value>
}
```

Формат заполнителя см. в разделе о формате результата для диапазонных векторов.

Следующий пример вычисляет выражение up за 30-секундный диапазон с шагом 15 секунд:

```
curl 'http://localhost:9090/api/v1/query_range?
query=up&start=2015-07-01T20:10:30.781Z&end=2015-07-01T20:11:00.781Z&step=15s'
```

Пример ответа:

```
{
   "status": "success",
   "data": {
      "resultType": "matrix",
      "result": [
         {
            "metric": {
               "__name__": "up",
               "job": "Deckhouse Prom++",
               "instance": "localhost:9090"
            },
            "values": [
               [ 1435781430.781, "1" ],
               [ 1435781445.781, "1" ],
               [ 1435781460.781, "1" ]
            ]
         },
         {
            "metric": {
               "__name__": "up",
               "job": "node",
               "instance": "localhost:9091"
            },
            "values": [
               [ 1435781430.781, "0" ],
               [ 1435781445.781, "0" ],
               [ 1435781460.781, "1" ]
            ]
         }
      ]
   }
}
```

Форматирование выражений запросов

Следующая конечная точка форматирует выражение PromQL в удобочитаемом виде:

```
GET /api/v1/format_query
POST /api/v1/format_query
```

Параметры URL-запроса:

- query=: Строка выражения Deckhouse Prom++.

Вы можете закодировать эти параметры в теле запроса, используя метод POST и заголовок Content-Type: application/x-www-form-urlencoded. Раздел data результата запроса — это строка с отформатированным выражением. Примечание: все комментарии удаляются в отформатированной строке.

Следующий пример форматирует выражение foo/bar:

```
curl 'http://localhost:9090/api/v1/format_query?query=foo/bar'
```

Пример ответа:

```
{
   "status": "success",
   "data": "foo / bar"
}
```

Запросы метаданных

Deckhouse Prom++ предоставляет набор API-конечных точек для запроса метаданных о рядах и их метках.

ПРИМЕЧАНИЕ: Эти конечные точки могут возвращать метаданные для рядов, для которых нет выборок в выбранном диапазоне времени, и/или для рядов, выборки которых были помечены как удаленные через API удаления. Точный объем дополнительно возвращаемых метаданных рядов является деталью реализации и может измениться в будущем.

Поиск рядов по селекторам лейблов

Следующая конечная точка возвращает список временных рядов, соответствующих определенному набору лейблов:

```
GET /api/v1/series
POST /api/v1/series
```

Параметры URL-запроса:

- match[]=<series_selector>: Повторяющийся аргумент селектора рядов, который выбирает ряды для возврата. Должен быть предоставлен хотя бы один аргумент match[].
- start=<rfc3339 | unix_timestamp>: Начальная метка времени.
- end=<rfc3339 | unix_timestamp>: Конечная метка времени.
- limit=: Максимальное количество возвращаемых рядов. Необязательный.

Раздел data результата запроса состоит из списка объектов, содержащих пары имя/значение метки, которые идентифицируют каждый ряд.

Следующий пример возвращает все ряды, соответствующие селекторам up или process_start_time_seconds{job="Deckhouse Prom++"}:

```
curl -g 'http://localhost:9090/api/v1/series?' --data-urlencode 'match[]=up' --data-urlencode 'match[]=process_start_time_seconds{job="Deckhouse Prom++"}'
```

Пример ответа:

```
{
   "status": "success",
   "data": [
      {
         "__name__": "up",
         "job": "Deckhouse Prom++",
         "instance": "localhost:9090"
      },
      {
         "__name__": "up",
         "job": "node",
         "instance": "localhost:9091"
      },
      {
         "__name__": "process_start_time_seconds",
         "job": "Deckhouse Prom++",
         "instance": "localhost:9090"
      }
   ]
}
```

Получение имен лейблов

Следующая конечная точка возвращает список имен лейблов:

```
GET /api/v1/labels
POST /api/v1/labels
```

Параметры URL-запроса:

- start=<rfc3339 | unix_timestamp>: Начальная метка времени. Необязательный.
- end=<rfc3339 | unix_timestamp>: Конечная метка времени. Необязательный.
- match[]=<series_selector>: Повторяющийся аргумент селектора рядов, который выбирает ряды, из которых читаются имена лейблов. Необязательный.
- limit=: Максимальное количество возвращаемых рядов. Необязательный.

Раздел data JSON-ответа — это список строковых имен лейблов.

Пример:

```
{
    "status": "success",
    "data": [
        "__name__",
        "call",
        "code",
        "config",
        "dialer_name",
        "endpoint",
        "event",
        "goversion",
        "handler",
        "instance",
        "interval",
        "job",
        "le",
        "listener_name",
        "name",
        "quantile",
        "reason",
        "role",
        "scrape_job",
        "slice",
        "version"
    ]
}
```

Запрос значений лейблов

Следующая конечная точка возвращает список значений лейблов для указанного имени метки:

```
GET /api/v1/label/<label_name>/values
```

Параметры URL-запроса:

- start=<rfc3339 | unix_timestamp>: Начальная метка времени. Необязательный.

- end=<rfc3339 | unix_timestamp>: Конечная метка времени. Необязательный.
- match[]=<series_selector>: Повторяющийся аргумент селектора рядов, который выбирает ряды, из которых читаются значения лейблов. Необязательный.
- limit=: Максимальное количество возвращаемых рядов. Необязательный.

Раздел data JSON-ответа — это список строковых значений лейблов.

Этот пример запрашивает все значения метки job:

```
curl http://localhost:9090/api/v1/label/job/values
```

```
{
   "status": "success",
   "data": [
      "node",
      "Deckhouse Prom++"
   ]
}
```

Запрос примеров (экземпляров)

Это экспериментальная функция, и она может измениться в будущем. Следующая конечная точка возвращает список примеров для допустимого запроса PromQL за определенный диапазон времени:

```
GET /api/v1/query_exemplars
POST /api/v1/query_exemplars
```

Параметры URL-запроса:

- query=: Строка выражения Deckhouse Prom++.
- start=<rfc3339 | unix_timestamp>: Начальная метка времени.
- end=<rfc3339 | unix_timestamp>: Конечная метка времени.

```
curl -g 'http://localhost:9090/api/v1/query_exemplars?
query=test_exemplar_metric_total&start=2020-09-14T15:22:25.479Z&end=2020-09-14T15:23:2
5.479Z'
```

Пример ответа:

```
{
    "status": "success",
    "data": [
        {
            "seriesLabels": {
                "__name__": "test_exemplar_metric_total",
                "instance": "localhost:8090",
                "job": "Deckhouse Prom++",
                "service": "bar"
            },
            "exemplars": [
                {
                    "labels": {
                        "trace_id": "EpTxMJ40fUus7aGY"
                    },
                    "value": "6",
                    "timestamp": 1600096945.479
                }
            ]
        },
        {
            "seriesLabels": {
                "__name__": "test_exemplar_metric_total",
                "instance": "localhost:8090",
                "job": "Deckhouse Prom++",
                "service": "foo"
            },
            "exemplars": [
                {
                    "labels": {
                        "trace_id": "Olp9XHlq763ccsfa"
                    },
                    "value": "19",
                    "timestamp": 1600096955.479
                },
                {
                    "labels": {
                        "trace_id": "hCtjygkIHwAN9vs4"
                    },
                    "value": "20",
                    "timestamp": 1600096965.489
                }
            ]
        }
    ]
}
```

Форматы результатов выражений

Запросы выражений могут возвращать следующие значения ответа в свойстве result раздела data. Заполнители <sample_value> представляют числовые значения выборок. JSON не поддерживает специальные значения с плавающей точкой, такие как NaN, Inf и -Inf, поэтому значения выборок передаются как строки JSON в кавычках, а не как сырые числа.

Ключи “histogram” и “histograms” появляются только в ответе, если присутствуют экспериментальные нативные гистограммы. Заполнитель подробно объяснен в соответствующем разделе ниже.

Диапазонные векторы

Диапазонные векторы возвращаются как результат типа matrix. Соответствующее свойство result имеет следующий формат:

```
[
  {
    "metric": { "<label_name>": "<label_value>", ... },
    "values": [ [ <unix_time>, "<sample_value>" ], ... ],
    "histograms": [ [ <unix_time>, <histogram> ], ... ]
  },
  ...
]
```

Каждый ряд может иметь ключ “values”, или ключ “histograms”, или оба. Для заданной метки времени будет только одна выборка типа float или гистограммы.

Ряды возвращаются отсортированными по метрике. Функции, такие как sort и sort_by_label, не влияют на диапазонные векторы.

Мгновенные векторы

Мгновенные векторы возвращаются как результат типа vector. Соответствующее свойство result имеет следующий формат:

```
[
  {
    "metric": { "<label_name>": "<label_value>", ... },
    "value": [ <unix_time>, "<sample_value>" ],
    "histogram": [ <unix_time>, <histogram> ]
  },
  ...
]
```

Каждый ряд может иметь ключ “value” или ключ “histogram”, но не оба.

Ряды не гарантированно возвращаются в каком-либо определенном порядке, если не используется функция, такая как sort или sort_by_label.

Скаляры

Скалярные результаты возвращаются как результат типа scalar. Соответствующее свойство result имеет следующий формат:

```
[ <unix_time>, "<scalar_value>" ]
```

Нативные гистограммы

Заполнитель , использованный выше, форматируется следующим образом.

Примечание: нативные гистограммы — это экспериментальная функция, и формат ниже может измениться.

```
{
  "count": "<count_of_observations>",
  "sum": "<sum_of_observations>",
  "buckets": [ [ <boundary_rule>, "<left_boundary>", "<right_boundary>",
"<count_in_bucket>" ], ... ]
}
```

Заполнитель <boundary_rule> — это целое число от 0 до 3 со следующим значением:

- 0: «открыто слева» (левая граница исключается, правая включается)
- 1: «открыто справа» (левая граница включается, правая исключается)
- 2: «открыто с обеих сторон» (обе границы исключаются)
- 3: «закрыто с обеих сторон» (обе границы включаются)

Примечание: В текущих реализованных схемах корзин положительные корзины «открыты слева», отрицательные — «открыты справа», а нулевая корзина (с отрицательной левой и положительной правой границами) — «закрыта с обеих сторон».

Цели

Следующая конечная точка возвращает обзор текущего состояния обнаружения целей Deckhouse Prom++:

```
GET /api/v1/targets
```

Как активные, так и отброшенные цели включены в ответ по умолчанию. Отброшенные цели подчиняются лимиту keep_dropped_targets, если он установлен. Поле labels представляет набор лейблов после применения перемаркировки. Поле discoveredLabels представляет немодифицированные метки, полученные во время обнаружения сервисов до перемаркировки.

```
curl http://localhost:9090/api/v1/targets
```

Пример ответа:


```
{
  "status": "success",
  "data": {
    "activeTargets": [
      {
        "discoveredLabels": {
          "__address__": "127.0.0.1:9090",
          "__metrics_path__": "/metrics",
          "__scheme__": "http",
          "job": "Deckhouse Prom++"
        },
        "labels": {
          "instance": "127.0.0.1:9090",
          "job": "Deckhouse Prom++"
        },
        "scrapePool": "Deckhouse Prom++",
        "scrapeUrl": "http://127.0.0.1:9090/metrics",
        "globalUrl": "http://example-Deckhouse Prom++:9090/metrics",
        "lastError": "",
        "lastScrape": "2017-01-17T15:07:44.723715405+01:00",
        "lastScrapeDuration": 0.050688943,
        "health": "up",
        "scrapeInterval": "1m",
        "scrapeTimeout": "10s"
      }
    ],
    "droppedTargets": [
      {
        "discoveredLabels": {
          "__address__": "127.0.0.1:9100",
          "__metrics_path__": "/metrics",
          "__scheme__": "http",
          "__scrape_interval__": "1m",
          "__scrape_timeout__": "10s",
          "job": "node"
        }
      }
    ]
  }
}
```


Параметр state позволяет фильтровать цели по состоянию (например, state=active, state=dropped, state=any). Пустой массив возвращается для отфильтрованных целей.

```
curl 'http://localhost:9090/api/v1/targets?state=active'
```

Пример ответа:


```
{
  "status": "success",
  "data": {
    "activeTargets": [
      {
        "discoveredLabels": {
          "__address__": "127.0.0.1:9090",
          "__metrics_path__": "/metrics",
          "__scheme__": "http",
          "job": "Deckhouse Prom++"
        },
        "labels": {
          "instance": "127.0.0.1:9090",
          "job": "Deckhouse Prom++"
        },
        "scrapePool": "Deckhouse Prom++",
        "scrapeUrl": "http://127.0.0.1:9090/metrics",
        "globalUrl": "http://example-Deckhouse Prom++:9090/metrics",
        "lastError": "",
        "lastScrape": "2017-01-17T15:07:44.723715405+01:00",
        "lastScrapeDuration": 50688943,
        "health": "up"
      }
    ],
    "droppedTargets": []
  }
}
```


Параметр scrapePool позволяет фильтровать цели по имени пула.

```
curl 'http://localhost:9090/api/v1/targets?scrapePool=node_exporter'
```

Пример ответа:

```
{
  "status": "success",
  "data": {
    "activeTargets": [
      {
        "discoveredLabels": {
          "__address__": "127.0.0.1:9091",
          "__metrics_path__": "/metrics",
          "__scheme__": "http",
          "job": "node_exporter"
        },
        "labels": {
          "instance": "127.0.0.1:9091",
          "job": "node_exporter"
        },
        "scrapePool": "node_exporter",
        "scrapeUrl": "http://127.0.0.1:9091/metrics",
        "globalUrl": "http://example-Deckhouse Prom++:9091/metrics",
        "lastError": "",
        "lastScrape": "2017-01-17T15:07:44.723715405+01:00",
        "lastScrapeDuration": 50688943,
        "health": "up"
      }
    ],
    "droppedTargets": []
  }
}
```


Правила

Конечная точка /rules возвращает список загруженных правил оповещения и записи. Кроме того, она возвращает текущие активные оповещения, сгенерированные каждым правилом оповещения.

Поскольку конечная точка /rules довольно новая, она не имеет таких же гарантий стабильности, как общий API v1.

```
GET /api/v1/rules
```

Параметры URL-запроса:

- type=alert|record: Возвращает только правила оповещения (например, type=alert) или правила записи (например, type=record). Если параметр отсутствует или пуст, фильтрация не применяется.
- rule_name[]=: Возвращает только правила с указанным именем. Если параметр повторяется, возвращаются правила с любым из указанных имен. Если все правила группы

отфильтрованы, группа не возвращается. Если параметр отсутствует или пуст, фильтрация не применяется.
- rule_group[]=: Возвращает только правила с указанным именем группы. Если параметр повторяется, возвращаются правила с любым из указанных имен групп. Если параметр отсутствует или пуст, фильтрация не применяется.
- file[]=: Возвращает только правила с указанным путем к файлу. Если параметр повторяется, возвращаются правила с любым из указанных путей. Если параметр отсутствует или пуст, фильтрация не применяется.
- exclude_alerts=: Возвращает только правила, без активных оповещений.

```
curl http://localhost:9090/api/v1/rules
```

Пример ответа:

```
{
    "data": {
        "groups": [
            {
                "rules": [
                    {
                        "alerts": [
                            {
                                "activeAt": "2018-07-04T20:27:12.60602144+02:00",
                                "annotations": {
                                    "summary": "High request latency"
                                },
                                "labels": {
                                    "alertname": "HighRequestLatency",
                                    "severity": "page"
                                },
                                "state": "firing",
                                "value": "1e+00"
                            }
                        ],
                        "annotations": {
                            "summary": "High request latency"
                        },
                        "duration": 600,
                        "health": "ok",
                        "labels": {
                            "severity": "page"
                        },
                        "name": "HighRequestLatency",
                        "query": "job:request_latency_seconds:mean5m{job=\"myjob\"} >
0.5",
                        "type": "alerting"
                    },
                    {
                        "health": "ok",
                        "name": "job:http_inprogress_requests:sum",
                        "query": "sum by (job) (http_inprogress_requests)",
                        "type": "recording"
                    }
                ],
                "file": "/rules.yaml",
                "interval": 60,
                "limit": 0,
                "name": "example"
            }
        ]
    },
```

```
    "status": "success"
}
```

Оповещения

Конечная точка /alerts возвращает список всех активных оповещений.

Поскольку конечная точка /alerts довольно новая, она не имеет таких же гарантий стабильности, как общий API v1.

```
GET /api/v1/alerts
```

```
curl http://localhost:9090/api/v1/alerts
```

Пример ответа:

```
{
    "data": {
        "alerts": [
            {
                "activeAt": "2018-07-04T20:27:12.60602144+02:00",
                "annotations": {},
                "labels": {
                    "alertname": "my-alert"
                },
                "state": "firing",
                "value": "1e+00"
            }
        ]
    },
    "status": "success"
}
```

Запрос метаданных целей

Следующая конечная точка возвращает метаданные о метриках, собираемых с целей. Это экспериментальная функция, и она может измениться в будущем.

```
GET /api/v1/targets/metadata
```

Параметры URL-запроса:

- match_target=<label_selectors>: Селекторы лейблов, которые сопоставляют цели по их наборам лейблов. Если не указано, выбираются все цели.
- metric=: Имя метрики, для которой запрашиваются метаданные. Если не указано, возвращаются метаданные для всех метрик.

- limit=: Максимальное количество целей для сопоставления.

Раздел data результата запроса состоит из списка объектов, содержащих метаданные метрик и набор лейблов цели.

Следующий пример возвращает все записи метаданных для метрики go_goroutines из первых двух целей с меткой job="Deckhouse Prom++".

```
curl -G http://localhost:9091/api/v1/targets/metadata \
    --data-urlencode 'metric=go_goroutines' \
    --data-urlencode 'match_target={job="Deckhouse Prom++"}' \
    --data-urlencode 'limit=2'
```

Пример ответа:

```
{
  "status": "success",
  "data": [
    {
      "target": {
        "instance": "127.0.0.1:9090",
        "job": "Deckhouse Prom++"
      },
      "type": "gauge",
      "help": "Number of goroutines that currently exist.",
      "unit": ""
    },
    {
      "target": {
        "instance": "127.0.0.1:9091",
        "job": "Deckhouse Prom++"
      },
      "type": "gauge",
      "help": "Number of goroutines that currently exist.",
      "unit": ""
    }
  ]
}
```

Следующий пример возвращает метаданные для всех метрик всех целей с меткой instance="127.0.0.1:9090".

```
curl -G http://localhost:9091/api/v1/targets/metadata \
    --data-urlencode 'match_target={instance="127.0.0.1:9090"}'
```

```
{
  "status": "success",
  "data": [
    // ...
    {
      "target": {
        "instance": "127.0.0.1:9090",
        "job": "Deckhouse Prom++"
      },
      "metric": "Deckhouse Prom++_treecache_zookeeper_failures_total",
      "type": "counter",
      "help": "The total number of ZooKeeper failures.",
      "unit": ""
    },
    {
      "target": {
        "instance": "127.0.0.1:9090",
        "job": "Deckhouse Prom++"
      },
      "metric": "Deckhouse Prom++_tsdb_reloads_total",
      "type": "counter",
      "help": "Number of times the database reloaded block data from disk.",
      "unit": ""
    },
    // ...
  ]
}
```

Запрос метаданных метрик

Она возвращает метаданные о метриках, собираемых с целей. Однако она не предоставляет информации о целях. Это экспериментальная функция, и она может измениться в будущем.

```
GET /api/v1/metadata
```

Параметры URL-запроса:

- limit=: Максимальное количество возвращаемых метрик.
- limit_per_metric=: Максимальное количество возвращаемых метаданных для каждой метрики.
- metric=: Имя метрики для фильтрации метаданных. Если не указано, возвращаются метаданные для всех метрик.

Раздел data результата запроса состоит из объекта, где каждый ключ — это имя метрики, а каждое значение — это список уникальных объектов метаданных, предоставленных для этого имени метрики всеми целями.

Следующий пример возвращает две метрики. Обратите внимание, что метрика http_requests_total имеет более одного объекта в списке. По крайней мере, у одной цели есть значение HELP, которое не совпадает с остальными.

```
curl -G http://localhost:9090/api/v1/metadata?limit=2
```

Пример ответа:

```
{
  "status": "success",
  "data": {
    "cortex_ring_tokens": [
      {
        "type": "gauge",
        "help": "Number of tokens in the ring",
        "unit": ""
      }
    ],
    "http_requests_total": [
      {
        "type": "counter",
        "help": "Number of HTTP requests",
        "unit": ""
      },
      {
        "type": "counter",
        "help": "Amount of HTTP requests",
        "unit": ""
      }
    ]
  }
}
```

Следующий пример возвращает только одну запись метаданных для каждой метрики.

```
curl -G http://localhost:9090/api/v1/metadata?limit_per_metric=1
```

Пример ответа:

```
{
  "status": "success",
  "data": {
    "cortex_ring_tokens": [
      {
        "type": "gauge",
        "help": "Number of tokens in the ring",
        "unit": ""
      }
    ],
    "http_requests_total": [
      {
        "type": "counter",
        "help": "Number of HTTP requests",
        "unit": ""
      }
    ]
  }
}
```


Следующий пример возвращает метаданные только для метрики http_requests_total

```
curl -G http://localhost:9090/api/v1/metadata?metric=http_requests_total
```

Пример ответа:


```
{
  "status": "success",
  "data": {
    "http_requests_total": [
      {
        "type": "counter",
        "help": "Number of HTTP requests",
        "unit": ""
      },
      {
        "type": "counter",
        "help": "Amount of HTTP requests",
        "unit": ""
      }
    ]
  }
}
```

Alertmanagers

Следующая конечная точка возвращает обзор текущего состояния обнаружения Alertmanager в Deckhouse Prom++:

```
GET /api/v1/alertmanagers
```

Как активные, так и отброшенные Alertmanager'ы включены в ответ.

```
curl http://localhost:9090/api/v1/alertmanagers
```

Пример ответа:

```
{
  "status": "success",
  "data": {
    "activeAlertmanagers": [
      {
        "url": "http://127.0.0.1:9090/api/v1/alerts"
      }
    ],
    "droppedAlertmanagers": [
      {
        "url": "http://127.0.0.1:9093/api/v1/alerts"
      }
    ]
  }
}
```

Состояние

Следующие конечные точки состояния предоставляют текущую конфигурацию Deckhouse Prom++.

Конфигурация

Следующая конечная точка возвращает текущий загруженный конфигурационный файл:

```
GET /api/v1/status/config
```

Конфигурация возвращается как дамп YAML-файла. Из-за ограничений библиотеки YAML комментарии не включаются.

```
curl http://localhost:9090/api/v1/status/config
```

Пример ответа:

```
{
  "status": "success",
  "data": {
    "yaml": "<содержимое загруженного конфигурационного файла в YAML>"
  }
}
```

Флаги

Следующая конечная точка возвращает значения флагов, с которыми был настроен Deckhouse Prom++:

```
GET /api/v1/status/flags
```

Все значения имеют тип результата string.

```
curl http://localhost:9090/api/v1/status/flags
```

Пример ответа:

```
{
  "status": "success",
  "data": {
    "alertmanager.notification-queue-capacity": "10000",
    "alertmanager.timeout": "10s",
    "log.level": "info",
    "query.lookback-delta": "5m",
    "query.max-concurrency": "20",
    ...
  }
}
```

Приемник OTLP

Deckhouse Prom++ можно настроить как приемник для протокола OTLP Metrics. Это не считается эффективным способом приема выборок. Используйте его с осторожностью для конкретных случаев с низким объемом данных. Он не подходит для замены приема через сканирование.

Включите приемник OTLP с помощью флага –enable-feature=otlp-write-receiver. Когда он включен, конечная точка приемника OTLP — /api/v1/otlp/v1/metrics.

Федерация

Федерация позволяет одному серверу Deckhouse Prom++ собирать выбранные временные ряды с другого сервера Deckhouse Prom++.

Сценарии использования

Федерация применяется в различных сценариях. Обычно её используют для обеспечения масштабируемости мониторинговых систем Deckhouse Prom++ или для получения связанных метрик из одного сервиса в другой.

Иерархическая федерация

Иерархическая федерация позволяет Deckhouse Prom++ масштабироваться в средах с десятками дата-центров и миллионами узлов. В этом случае топология федерации напоминает дерево, где серверы Deckhouse Prom++ верхнего уровня собирают агрегированные данные временных рядов с большого числа подчинённых серверов.

Например, настройка может состоять из множества серверов Deckhouse Prom++ в каждом дата-центре, которые собирают данные с высокой детализацией (уровень экземпляра), и набора глобальных серверов Deckhouse Prom++, которые собирают и хранят только агрегированные данные (уровень задания) с этих локальных серверов. Это обеспечивает общее глобальное представление и детальные локальные представления.

Федерация между сервисами

В федерации между сервисами сервер Deckhouse Prom++ одного сервиса настраивается для сбора выбранных данных с сервера Deckhouse Prom++ другого сервиса, чтобы обеспечить оповещения и запросы по обоим наборам данных в одном сервере.

Например, кластерный планировщик, запускающий несколько сервисов, может предоставлять информацию об использовании ресурсов (таких как использование памяти и CPU) о экземплярах сервисов, работающих в кластере. С другой стороны, сервис, работающий в этом кластере, будет предоставлять только специфичные для приложения метрики сервиса. Часто эти два набора метрик собираются отдельными серверами Deckhouse Prom++. С помощью федерации сервер Deckhouse Prom++, содержащий метрики уровня сервиса, может получать метрики использования ресурсов кластера о своём конкретном сервисе от кластерного Deckhouse Prom++, чтобы оба набора метрик могли использоваться в этом сервере.

Настройка федерации

На любом сервере Deckhouse Prom++ эндпоинт `/federate` позволяет получать текущие значения для выбранного набора временных рядов на этом сервере. Необходимо указать хотя бы один параметр URL `match[]` для выбора серий для экспорта. Каждый аргумент `match[]` должен задавать селектор мгновенного вектора, например, `up` или `{job="api-server"}`. Если предоставлено несколько параметров `match[]`, выбирается объединение всех совпадающих серий.

Чтобы федеративно собирать метрики с одного сервера на другой, настройте ваш целевой сервер Deckhouse Prom++ для сбора с эндпоинта `/federate` исходного сервера, при этом включите опцию сбора `honor_labels` (чтобы не перезаписывать метки, предоставленные исходным

сервером) и передайте необходимые параметры `match[]`. Например, следующая конфигурация `scrape_configs` собирает любые ряды с меткой `job="PromPP"` или с именем метрики, начинающимся с `job:`, с серверов Deckhouse Prom++ по адресам `source-PromPP-{1,2,3}:9090` в собирающий сервер Deckhouse Prom++:

```
scrape_configs:
  - job_name: 'federate'
    scrape_interval: 15s

    honor_labels: true
    metrics_path: '/federate'

    params:
      'match[]':
        - '{job="PromPP"}'
        - '{__name__=~"job:.*"}'

    static_configs:
      - targets:
        - 'source-PromPP-1:9090'
        - 'source-PromPP-2:9090'
        - 'source-PromPP-3:9090'
```

Обнаружение сервисов по HTTP (HTTP SD)

Deckhouse Prom++ предоставляет универсальный механизм HTTP Service Discovery (HTTP SD), который позволяет обнаруживать цели через HTTP-эндпоинт.

HTTP SD является дополнением к поддерживаемым механизмам обнаружения сервисов и представляет собой альтернативу механизму обнаружения на основе файлов (File-based SD).

Сравнение File-based SD и HTTP SD

Ниже представлена таблица сравнения двух универсальных реализаций Service Discovery.

Параметр	File-based SD	HTTP SD
Основано на событиях	Да, через inotify	Нет
Частота обновлений	Мгновенно, благодаря inotify	Согласно `refresh_interval`
Формат	YAML или JSON	JSON
Доставка	Локальный файл	HTTP/HTTPS
Безопасность	Права доступа к файлам	TLS, Basic-аутентификация, заголовок Authorization, OAuth2

Требования к эндпоинтам HTTP SD

Если вы реализуете эндпоинт HTTP SD, следует учитывать следующие требования.

Ответ принимается «как есть», без изменений. С каждым интервалом обновления (по умолчанию — 1 минута) Deckhouse Prom++ будет выполнять GET-запрос к эндпоинту HTTP SD. GET-запрос содержит HTTP-заголовок `X-Deckhouse Prom++-Refresh-Interval-Seconds` с указанием интервала обновления.

Ответ от SD-эндпоинта должен содержать HTTP-код `200` и HTTP-заголовок `Content-Type: application/json`. Ответ должен быть в формате UTF-8. Если не нужно передавать цели, также должен возвращаться HTTP-код `200` с пустым списком `[]`. Списки целей не упорядочены.

Deckhouse Prom++ кэширует списки целей. Если при получении обновленного списка целей происходит ошибка, Deckhouse Prom++ продолжает использовать текущий список целей. Список целей не сохраняется после перезапуска. Метрика `prompp_sd_http_failures_total` отслеживает количество неудачных обновлений.

Полный список целей должен возвращаться при каждом сканировании. Инкрементальные обновления не поддерживаются. Экземпляр Deckhouse Prom++ не отправляет свое имя хоста, и SD-эндпоинт не может определить, является ли запрос SD первым после перезапуска.

URL-адрес HTTP SD не считается секретным. Аутентификация и любые API-ключи должны передаваться с использованием соответствующих механизмов аутентификации. Deckhouse Prom++ поддерживает TLS-аутентификацию, Basic-аутентификацию, OAuth2 и заголовки авторизации.

Формат HTTP SD

```
[
  {
    "targets": [ "<host>", ... ],
    "labels": {
      "<labelname>": "<labelvalue>", ...
    }
  },
  ...
]
```

Примеры:

```
[
    {
        "targets": ["10.0.10.2:9100", "10.0.10.3:9100", "10.0.10.4:9100", "10.0.10.5:9
100"],
        "labels": {
            "__meta_datacenter": "london",
            "__meta_prompp_job": "node"
        }
    },
    {
        "targets": ["10.0.40.2:9100", "10.0.40.3:9100"],
        "labels": {
            "__meta_datacenter": "london",
            "__meta_prompp_job": "alertmanager"
        }
    },
    {
        "targets": ["10.0.40.2:9093", "10.0.40.3:9093"],
        "labels": {
            "__meta_datacenter": "newyork",
            "__meta_prompp_job": "alertmanager"
        }
    }
]
```

Командная строка

Ниже перечислены доступные флаги командной строки Deckhouse Prom++ и их описания.

Флаг	Описание	Значение по умолчанию
`-h`, `--help`	Показать контекстно-зависимую справку (также доступны `--help-long` и `--help-man`)	
`--version`	Показать версию приложения	
`--config.file`	Путь к файлу конфигурации Deckhouse Prom++	`prometheus.yml`
`--web.listen-address`	Адрес для прослушивания UI, API и телеметрии	`0.0.0.0:9090`
`--auto-gomemlimit.ratio`	Соотношение зарезервированной памяти GOMEMLIMIT к максимальной памяти системы	`0.9`
`--web.config.file`	(**Экспериментальный**) Путь к файлу конфигурации TLS или аутентификации	
`--web.read-timeout`	Максимальное время ожидания чтения запроса	`5m`
`--web.max-connections`	Максимальное количество одновременных подключений	`512`
`--web.external-url`	Внешний URL-адрес для доступа к Deckhouse Prom++ (через обратный прокси)	
`--web.route-prefix`	Префикс для внутренних маршрутов веб-интерфейса	
`--web.user-assets`	Путь к статическим ресурсам (`/user`)	
`--web.enable-lifecycle`	Разрешить выключение и перезагрузку через HTTP-запросы	`false`
`--web.enable-admin-api`	Включить API для административных действий	`false`

Флаг	Описание	Значение по умолчанию
`--web.enable-remote-write-receiver`	Включить API для remote-write-запросов	`false`
`--web.console.templates`	Путь к шаблонам консоли (`/consoles`)	`consoles`
`--web.console.libraries`	Путь к библиотекам консоли	`console_libraries`
`--web.page-title`	Заголовок страницы	`Deckhouse Prom++ Time Series Collection and Processing Server`
`--web.cors.origin`	Regex для CORS origin	
`--storage.tsdb.path`	Путь к данным метрик (server mode)	`data/`
`--storage.tsdb.retention`	(**Устаревший**) Время хранения samples	
`--storage.tsdb.retention.time`	Время хранения samples (переопределяет `retention`)	
`--storage.tsdb.retention.size`	Макс. объем хранилища (например `512MB`)	
`--storage.tsdb.no-lockfile`	Не создавать lock-файл	`false`
`--storage.tsdb.head-chunks-write-queue-size`	Размер очереди записи head chunks	`0`
`--storage.agent.path`	Путь к данным (agent mode)	`data-agent/`
`--storage.agent.wal-compression`	Сжимать WAL агента	`true`
`--storage.agent.retention.min-time`	Мин. время хранения samples (agent mode)	
`--storage.agent.retention.max-time`	Макс. время хранения samples (agent mode)	

Флаг	Описание	Значение по умолчанию
`--storage.agent.no-lockfile`	Не создавать lock-файл (agent mode)	`false`
`--storage.remote.flush-deadline`	Таймаут сброса данных при выключении	`1m`
`--storage.remote.read-sample-limit`	Лимит samples для remote read	`5e7`
`--storage.remote.read-concurrent-limit`	Лимит concurrent remote read	`10`
`--storage.remote.read-max-bytes-in-frame`	Макс. размер фрейма для remote read	`1048576`
`--rules.alert.for-outage-tolerance`	Макс. время простоя для восстановления алертов	`1h`
`--rules.alert.for-grace-period`	Минимальный интервал для алертов	`10m`
`--rules.alert.resend-delay`	Задержка перед повторной отправкой алерта	`1m`
`--rules.max-concurrent-evals`	Лимит параллельных оценок правил	`4`
`--alertmanager.notification-queue-capacity`	Размер очереди уведомлений	`10000`
`--query.lookback-delta`	Макс. lookback для запросов	`5m`
`--query.timeout`	Таймаут выполнения запроса	`2m`
`--query.max-concurrency`	Макс. параллельных запросов	`20`
`--query.max-samples`	Лимит samples на запрос	`50000000`
`--enable-feature`	Включение экспериментальных функций	
`--log.level`	Уровень логирования (`debug`, `info`, `warn`, `error`)	`info`

Флаг	Описание	Значение по умолчанию
`--log.format`	Формат логов (`logfmt` , `json`)	`logfmt`

API управления

Deckhouse Prom++ предоставляет набор API для управления, облегчающих автоматизацию и интеграцию.

Проверка состояния здоровья

```
GET /-/healthy
HEAD /-/healthy
```

Эндпоинт всегда возвращает статус `200` и предназначен для проверки состояния здоровья Deckhouse Prom++.

Проверка готовности

```
GET /-/ready
HEAD /-/ready
```

Эндпоинт возвращает статус 200, когда Deckhouse Prom++ готов обслуживать трафик (т.е. отвечать на запросы).

Перезагрузка

```
PUT  /-/reload
POST /-/reload
```

Эндпоинт инициирует перезагрузку конфигурации Deckhouse Prom++ и файлов правил. По умолчанию он отключен и может быть включен с помощью флага `--web.enable-lifecycle` .

Перезагрузку конфигурации можно также инициировать, отправив сигнал `SIGHUP` процессу Deckhouse Prom++.

Завершение работы

```
PUT  /-/quit
POST /-/quit
```

Эндпоинт инициирует корректное (graceful) завершение работы Deckhouse Prom++. По умолчанию он отключен и может быть включен с помощью флага `--web.enable-lifecycle`.

Корректное завершение работы можно также инициировать, отправив сигнал `SIGTERM` процессу Deckhouse Prom++.