        - Настройка Prometheus rules
        - PrometheusRule
- Логирование
- Error Reporting
- Tracing
- Мониторинг и логирование Kubernetes в Production

Зачем нужен мониторинг

Без сбора логов и трейсинг-запросов прожить можно, без метрик и мониторинга - нет.
Система мониторинга - это система сбора метрик.
Метрики - sets of numbers that give information about a particular process or activity. То есть это всегда цифры.
Из системы мониторинга не нужно собирать трейсинги и логи (поскольку метрики - это только цифры).

Группы метрик

System-level performance metrics

- относятся к серверам или ВМ
- загруженность CPU, утилизацию памяти, ввод-вывод

Application-level performance metrics

- относятся к показателям работы приложения
- I/O на уровне приложения, использование кэша, кол-во запросов итд

Server availability and uptime metrics - доступность, получаемая с помощью хелсчеков

Business-level metrics - абстрактные метрики, напрямую не связанные с приложением или утилизацией ресурсов. Например, кол-во заказов в час для интернет-магазина. Эти метрики наиболее приоритетные, они выводятся на дашборды и с ними связаны приоритетные инциденты.

Прометей не предназначен для длительного хранения метрик, в среднем до 14 дней, для длительного хранения нужна федерация с серией серверов с разной частотой скрейпинга.

Что мониторить

2 метода - **USE** и **RED**

RED:

(Request) Rate - the number of requests, per second, your services are serving.
(Request) Errors - the number of failed requests per second.
(Request) Duration - The amount of time each request takes expressed as a time interval.

USE:

Resource: all physical server functional components (CPUs, disks, busses, ...) [1]

For every resource:
Utilization: the average time that the resource was busy servicing work [2]
Saturation: the degree to which the resource has extra work which it can't service, often queued
Errors: the count of error events

Алертинг

Без алертов мониторинг - бесполезная штука, так как десятки тысяч метрик ни один инженер не обработает и не отреагирует вовремя.
Нужна автоматизация.

Первый тип алертинга - молчаливые обезьяны: ничего не вижу, не слышу и никому не скажу. Это когда уведомления приходят слишком поздно либо вообще не приходят.

Второй тип - пастух из басни, который кричит про волков, а когда волки реально приходят - ему никто не верит и всех овец съедают. Приходит огромное кол-во уведомлений - alert flooding.

Третий тип - хорошо настроенная система алертинга.
Правила такой системы: (правильные паттерны)

- алертинг только тех событий, которые требуют вмешательства инженера (когда кончается место на диске итд)
- любой алерт, который влечет за собой изменения в системе (чтобы далее не получать этот алерт - апдейты безопасности, обновление ПО итд)
- алерты прорабатывать совместно с разработчиками и бизнесом
- периодический анализ нагрузки исходя из отчетов мониторинга
- service discovery - автоматизированное обнаружение и подключение систем к системе мониторинга. В контексте Прометея: Prometheus service discovery is a standard method of finding endpoints to scrape for metrics.

Как делать не надо

- настраивать все вручную и вручную добавлять серверы к системе мониторинга
- алертинг на основании мониторинга ресурсов (потребление CPU памяти итд) - приводит к **alert flooding** и **false positive alerts** (например для приложений со скачками потребления CPU, что для приложения является нормальной работой)
- выделенные роли для специалиста по мониторингу (тогда как надо в первую очередь привлекать разработчиков для прикладных и бизнесовых метрик)
- использование единого инструмента для мониторинга, логирования, трейсинга запросов - про то, когда система мониторинга используется не по назначению. Мониторинг - это только сбор метрик и это всегда цифры!

Monitoring and Observability

Все компании хотят **observability**, никому недостаточно только лишь мониторинга.
Раньше были только физические сервера. На них работали монолитные приложения, базы данных.
Система мониторинга выполняла ограниченный круг задач - доступность и отдельные метрики по производительности.
Изменился IT-ландшафт, появился доступный интернет, микросервисы и повысились требования к сервисам.
Облака привели к тому, что возросло кол-во сущностей - теперь это сотни виртуальных машин и тысячи контейнеров даже в средней компании. Появилась потребность в Service discovery.

Работать с логами как раньше тоже не получалось - найти на тысячах сущностей нужный лог это сложно -> новый класс ПО - это логгинг-серверы. Это централизованный сборщик логов с веб-интерфейсов с поиском и фильтром по логам + опционально система алертинга на основе логов.
С появлением микросервисов появилась нужда в трейсинг-серверах. Так как очень много взаимодействий между сервисами и непонятно где и на каком этапе появляются задержки и таймауты.

Требования стабильности - об ошибках теперь требуется узнавать как можно раньше, причем и об ошибках, которые возникают на клиентской стороне (фронт-энд). Это из-за того, что все большая часть кода теперь выполняется на клиентской стороне -> появились Error Tracking-системы (например Sentry)

Observability нужен не всем, но знать про него нужно каждому.

- Service Discovery
- централизованное логирование
- трейсинг запросов для микросервисов
- сбор и отслеживание ошибок

Установка Prometheus

Введение

Есть 2 варианта установки Прометея в K8S:

- с помощью оператора - продвинутый с верхнеуровневыми абстракциями - CRD, многое остается за кадром.
- ванильная версия без оператора и CRD

Ставится в обоих случаях через Helm.

Специфичный для K8S экспортер - `kubeStateMetrics`

Установка в Kubernetes

Подключение к стенду:

```
$ ssh <номер студента>@sbox.slurm.io
```

Далее с джамп-хоста - подключение на `master-1.<номер студента>`

В качестве Persistent Volumes используется SC Local Storage

Поиск чарта Прометея в публичном хабе Хелма:

```
$ helm search hub prometheus
```

В репозитории `stable` почти все экспортеры в статусе `deprecated`, поэтому ставить надо из репозитория `prometheus-community`:

```
$ helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
```

Скачать чарт локально:

```
$ helm pull prometheus-community/prometheus --untar
$ cd prometheus/
```

Разбор чарта

`values.yaml` - очень жирный файл, больше 1600 строк. Много возможностей конфигурации. Вот ключевые параметры, которые важны на проде:

- `rbac: create: true` - создание нужных прав в кластере для сервисных аккаунтов, ролей, RoleBinding
- `podSecurityPolicy: enabled: false` - если в кластере используется PSP, то выставить в `true`
- `imagePullSecrets` - указать доступы, если используются приватные реджистри
- `alertmanager: priorityClassName` - задать, если в кластере используются классы приоритетов (например, если в одном кластере крутятся поды дева и прода - класс для прода выставляется выше)
- `alertmanager: ingress` - создание ингресса для веб-морды алертменеджера. Включать или нет - дело вкуса, но веб-морда у него минималистичная и почти бесполезна.
- `alertmanager: podDisruptionBudget` - нужно на проде, где нужна высокая доступность системы мониторинга и на ней часто происходят изменения.
- `alertmanager: persistentVolume` - по умолчанию включено, создаются PVC с режимом доступа `readWriteOnce`. По размеру как PVC, так и ресурсы пода надо подбирать эмпирически, при необходимости увеличивать.
- `alertmanager: replicaCount` - в проде ставить больше 1 и включить `statefulSet`
- `alertmanager: securityContext` - лучше не трогать
- `configmapReload` - оставить всё в `true`, чтобы не дергать руками при изменении конфигов.
- `nodeExporter` - оставить включенным, также включенным оставить параметры `hostNetwork` и `hostPID`, чтобы экспортер мог обращаться к хостовому сетевому и PID-неймспейсу.
- `nodeExporter: tolerations` - надо включать, чтобы обойти taint-ы на мастерах, без этого с мастеров собирать данные будет нельзя:
- ```
  tolerations:
    - operator: "Exists"
      effect: "NoSchedule"
  ```

- `nodeExporter: resources` - лучше задать ресурсы для подов экспортера
- `server: global`: важные параметры: `scrape_interval` - с какой частотой Прометей будет ходить к своим таргетам и собирать метрики, `scrape_timeout` - сколько ждать до сброса запроса на сбор метрики, `evaluation_interval` - сколько ждать до пересчитывания правил.
- `server: remoteWrite` и `server: remoteRead` - работа с внешним хранилищем (например с VictoriaMetrics)
- `server: ingress` - лучше включить. Аннотации можно включить, если в кластере стоит `cert-manager`, чтобы на ингресс автоматически выписывались сертификаты. `hosts:` - DNS-имя для обращений. Выставляется как `prometheus.<номер студента>.edu.slurm.io`.
- `server: persistentVolume` - в проде обязательно. В лабе поменять `size` на 5ГБ и `storageClass` выставить в `local-storage`
  ```

- `server: replicaCount` - в проде надо больше 1.
- `server: retentionTime` - время в днях до удаления старых метрик, настраивается эмпирически в зависимости от наполнения дискового пространства. В альфа-версии так же есть возможность удаления метрик не только по времени, но и по размеру.
- `pushgateway` - включается, если нужен Push Gateway.

Перед тем, как запускать создание чарта, нужно сделать папки на воркерах, на которые ссылаются PV.

```
$ mkdir /local/pv{1,2}
$ kubectl create -f pv{1,2}.yml
```

Ручных правок в чартах лучше не делать, в проде все надо делать через CI!

Установка чарта:

```
$ helm install prometheus --namespace monitoring ./prometheus/ --create-
namespace
```

Совсем недавно к Prometheus прикрутили нативную Basic Auth - авторизацию, авторизацию надо было прикручивать только через Ingress. Еще прикрутили авторизацию по TLS-сертификатам. Подробнее читать в `RELEASE NOTES`.

Разбор специфичных exporter-ов

Самый распространенный специфичный экспортер - это `kube-state-metrics`.

Этот под запускает с огромным списком аргументов, которые указывают, с каких объектов K8S собирать метрики.

Можно его покурлить, он отдает метрики по порту 8080.

Полезные метрики и запросы:

- `kube_deployment_status_replicas_unavailable` - метрика, которая отслеживает кол-во недоступных реплик в каждом деплойменте, можно фильтровать по имени деплоймента. Можно использовать для правил и поджигания алертов.
- `kube_node_status_capacity_memory_bytes` - сколько памяти на нодах. Если заменить capacity на allocateable - то покажет, сколько реально выделяемой памяти (за вычетом системного резерва)
- `kube_pod_container_status_restarts_total` - выводит поды и их кол-во рестартов. Удобно делать алерты типа "под рестартовал более 2 раз в течение получаса"
- `kube_pod_container_status_ready` - показывает поды, которые в статусе Ready. Соответственно, те которые NotReady - надо мерять по !=1

Еще набор часто используемых метрик

Resource Metric: Kubernetes CPU Usage

Expression:

```
sum(rate(container_cpu_usage_seconds_total{container_name!="POD",pod_name!=""}
[5m]))
```

Resource Metric: Kubernetes Container CPU usage

Expression:

```
rate(container_cpu_usage_seconds_total{container_name!="POD",pod_name!=""}[5m])
```

OR

```
sum(rate(container_cpu_usage_seconds_total{container_name!="POD",pod_name!=""}
[5m])) by (container_name)
```

Resource Metric: Kubernetes Pod CPU usage

Expression:

```
sum(rate(container_cpu_usage_seconds_total{container_name!="POD",pod_name!=""}
[5m])) by (pod_name)
```

Resource Metric: Kubernetes Namespace CPU usage

Expression:

```
sum(rate(container_cpu_usage_seconds_total{container_name!="POD",namespace!=""}
[5m])) by (namespace)
```

Resource Metric: Kubernetes CPU requests

Expression:

```
sum(kube_pod_container_resource_requests_cpu_cores)
```

Resource Metric: Kubernetes Pod CPU requests

Expression:

```
sum(kube_pod_container_resource_requests_cpu_cores) by (pod)
```

Resource Metric: Kubernetes Namespace CPU Requests

Expression:

```
sum(kube_pod_container_resource_requests_cpu_cores) by (namespace)
```

Resource Metric: Kubernetes CPU Limits

Expression:

```
sum(kube_pod_container_resource_limits_cpu_cores)
```

Resource metric: Kubernetes Namespace CPU Limits

Expression:

```
sum(kube_pod_container_resource_limits_cpu_cores) by (namespace)
```

Resource Metric: Kubernetes CPU Request Commitment

Expression:

```
sum(kube_pod_container_resource_requests_cpu_cores) /
sum(node:node_num_cpu:sum)*100
```

This will give us the CPU request commitment for the entire cluster.

Resource Metric: Kubernetes Node CPU Request Commitment

Expression:

```
(sum(kube_pod_container_resource_requests_cpu_cores)  by (node) /
node:node_num_cpu:sum)*100
```

Resource Metric: Kubernetes CPU Saturation

Expression:

```
sum(node_load1) by (node) / count(node_cpu{mode="system"}) by (node)*100
```

This gives us the CPU saturation for the cluster.

Resource Metric: Kubernetes Node CPU Saturation

Expression:

```
node:node_cpu_saturation_load1:
```

Resource Metric: Kubernetes Memory Usage

Expression:

```
sum(container_memory_usage_bytes{container_name!="POD",container_name!=""})
```

This will give us the total memory usage for all pods in the cluster

Resource Metric: Kubernetes Node Memory Usage

Expression:

```
sum(container_memory_usage_bytes{container_name!="POD",container_name!=""}) by
(node)
```

Resource Metric: Kubernetes Memory Requests

Expression:

```
sum(kube_pod_container_resource_requests_memory_bytes)
```

Resource Metric: Kubernetes Node Memory Requests

Expression:

```
sum(kube_pod_container_resource_requests_memory_bytes) by (node)
```

Resource Metric: Kubernetes Memory Limits

Expression:

```
sum(kube_pod_container_resource_limits_memory_bytes)
```

This will give us the sum of memory limits for all pods in the cluster

Resource Metric: Kubernetes Node Memory Limits

Expression:

```
sum(kube_pod_container_resource_limits_memory_bytes) by (node)
```

Resource Metric: Kubernetes Memory Limit Commitment

Expression:

```
(sum(kube_pod_container_resource_limits_memory_bytes) /
:node_memory_MemTotal:sum)*100
```

This will give us the memory limit commitment for the entire cluster.

Тюнинг Prometheus

Config-файлы Prometheus

Основной конфиг Прометея - это ConfigMap. Так как это объект K8S, то редактировать его лучше через kubectl edit:

```
kubectl edit cm -n monitoring prometheus-server
```

Большинство настроек перекочевало из Values helm-чарта.

`scrape_configs` - описываются правила и инструкции, по которым Прометей будет искать и собирать метрики. Имя инструкции - в аннотации `- job_name` Эти инструкции можно посмотреть в Web UI в секции `Targets`

`static_configs` - можно задавать в этой секции таргеты удаленных хостов для сбора метрик.

Но лучшая практика - это не прописывать эндпойнты руками, а настроить Service Discovery.

Service Discovery в Prometheus

Роли в конфиге - категории данных, которые Прометей запрашивает у API сервера Kubernetes. Роли категоризируют все доступные эндпойнты (первая часть Service Discovery).

Фильтровать эндпойнты можно по лейблам (аннотация `relabel_configs`)

В Web UI можно в секции Service Discovery увидеть все собранные роли согласно конфигу (например, роль `kubernetes-apiservers`), и внутри роли - все обнаруженные лейблы (`Discovered Labels`) и лейблы, по которым была проведена фильтрация (`Target Labels`)

Пример куска конфига:

```
scrape_configs:
  relabel_configs:
  - action: keep
    regex: default;kubernetes;https
    source_labels:
    - __meta_kubernetes_namespace
    - __meta_kubernetes_service_name
    - __meta_kubernetes_endpoint_port_name
```

Что здесь видим? Из собранных лейблов берем три лейбла и согласно регэкспу фильтруем по `namespace=default`, `service_name=kubernetes` и `port_name=https`. Так осуществляется Service Discovery API-сервера Kubernetes.

Далее - выдержка из официальной документации Prometheus про конфигурацию Service Discovery.

Prometheus is configured via command-line flags and a configuration file. While the command-line flags configure immutable system parameters (such as storage locations, amount of data to keep on disk and in memory, etc.), the configuration file defines everything related to scraping jobs and their instances, as well as which rule files to load.

To view all available command-line flags, run `./prometheus -h`.

Prometheus can reload its configuration at runtime. If the new configuration is not well-formed, the changes will not be applied. A configuration reload is triggered by sending a `SIGHUP` to the Prometheus process or sending a HTTP POST request to the `/-/reload` endpoint (when the `--web.enable-lifecycle` flag is enabled). This will also reload any configured rule files.

Configuration file

To specify which configuration file to load, use the `--config.file` flag.

The file is written in YAML format, defined by the scheme described below. Brackets indicate that a parameter is optional. For non-list parameters the value is set to the specified default.

Generic placeholders are defined as follows:

- `<boolean>`: a boolean that can take the values `true` or `false`
- `<duration>`: a duration matching the regular expression `((([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?|0)`, e.g. `1d`, `1h30m`, `5m`, `10s`
- `<filename>`: a valid path in the current working directory

- `<host>`: a valid string consisting of a hostname or IP followed by an optional port number
- `<int>`: an integer value
- `<labelname>`: a string matching the regular expression `[a-zA-Z_][a-zA-Z0-9_]*`
- `<labelvalue>`: a string of unicode characters
- `<path>`: a valid URL path
- `<scheme>`: a string that can take the values `http` or `https`
- `<secret>`: a regular string that is a secret, such as a password
- `<string>`: a regular string
- `<tmpl_string>`: a string which is template-expanded before usage

The other placeholders are specified separately.

A valid example file can be found here.

The global configuration specifies parameters that are valid in all other configuration contexts. They also serve as defaults for other configuration sections.

```
global:
  # How frequently to scrape targets by default.
  [ scrape_interval: <duration> | default = 1m ]

  # How long until a scrape request times out.
  [ scrape_timeout: <duration> | default = 10s ]

  # How frequently to evaluate rules.
  [ evaluation_interval: <duration> | default = 1m ]

  # The labels to add to any time series or alerts when communicating with
  # external systems (federation, remote storage, Alertmanager).
  external_labels:
    [ <labelname>: <labelvalue> ... ]

  # File to which PromQL queries are logged.
  # Reloading the configuration will reopen the file.
  [ query_log_file: <string> ]

# Rule files specifies a list of globs. Rules and alerts are read from
# all matching files.
rule_files:
  [ - <filepath_glob> ... ]

# A list of scrape configurations.
scrape_configs:
  [ - <scrape_config> ... ]

# Alerting specifies settings related to the Alertmanager.
alerting:
  alert_relabel_configs:
    [ - <relabel_config> ... ]
  alertmanagers:
    [ - <alertmanager_config> ... ]

# Settings related to the remote write feature.
remote_write:
  [ - <remote_write> ... ]

# Settings related to the remote read feature.
```

```
remote_read:
  [ - <remote_read> ... ]
```

<scrape_config>

A `scrape_config` section specifies a set of targets and parameters describing how to scrape them. In the general case, one scrape configuration specifies a single job. In advanced configurations, this may change.

Targets may be statically configured via the `static_configs` parameter or dynamically discovered using one of the supported service-discovery mechanisms.

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

```
# The job name assigned to scraped metrics by default.
job_name: <job_name>

# How frequently to scrape targets from this job.
[ scrape_interval: <duration> | default = <global_config.scrape_interval> ]

# Per-scrape timeout when scraping this job.
[ scrape_timeout: <duration> | default = <global_config.scrape_timeout> ]

# The HTTP resource path on which to fetch metrics from targets.
[ metrics_path: <path> | default = /metrics ]

# honor_labels controls how Prometheus handles conflicts between labels that are
# already present in scraped data and labels that Prometheus would attach
# server-side ("job" and "instance" labels, manually configured target
# labels, and labels generated by service discovery implementations).
#
# If honor_labels is set to "true", label conflicts are resolved by keeping
label
# values from the scraped data and ignoring the conflicting server-side labels.
#
# If honor_labels is set to "false", label conflicts are resolved by renaming
# conflicting labels in the scraped data to "exported_<original-label>" (for
# example "exported_instance", "exported_job") and then attaching server-side
# labels.
#
# Setting honor_labels to "true" is useful for use cases such as federation and
# scraping the Pushgateway, where all labels specified in the target should be
# preserved.
#
# Note that any globally configured "external_labels" are unaffected by this
# setting. In communication with external systems, they are always applied only
# when a time series does not have a given label yet and are ignored otherwise.
[ honor_labels: <boolean> | default = false ]

# honor_timestamps controls whether Prometheus respects the timestamps present
# in scraped data.
#
# If honor_timestamps is set to "true", the timestamps of the metrics exposed
# by the target will be used.
#
# If honor_timestamps is set to "false", the timestamps of the metrics exposed
```

```
# by the target will be ignored.
[ honor_timestamps: <boolean> | default = true ]

# Configures the protocol scheme used for requests.
[ scheme: <scheme> | default = http ]

# Optional HTTP URL parameters.
params:
  [ <string>: [<string>, ...] ]

# Sets the `Authorization` header on every scrape request with the
# configured username and password.
# password and password_file are mutually exclusive.
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# Sets the `Authorization` header on every scrape request with
# the configured bearer token. It is mutually exclusive with
`bearer_token_file`.
[ bearer_token: <secret> ]

# Sets the `Authorization` header on every scrape request with the bearer token
# read from the configured file. It is mutually exclusive with `bearer_token`.
[ bearer_token_file: <filename> ]

# Configures the scrape request's TLS settings.
tls_config:
  [ <tls_config> ]

# Optional proxy URL.
[ proxy_url: <string> ]

# List of Azure service discovery configurations.
azure_sd_configs:
  [ - <azure_sd_config> ... ]

# List of Consul service discovery configurations.
consul_sd_configs:
  [ - <consul_sd_config> ... ]

# List of DigitalOcean service discovery configurations.
digitalocean_sd_configs:
  [ - <digitalocean_sd_config> ... ]

# List of Docker Swarm service discovery configurations.
dockerswarm_sd_configs:
  [ - <dockerswarm_sd_config> ... ]

# List of DNS service discovery configurations.
dns_sd_configs:
  [ - <dns_sd_config> ... ]

# List of EC2 service discovery configurations.
ec2_sd_configs:
  [ - <ec2_sd_config> ... ]
```

```
# List of Eureka service discovery configurations.
eureka_sd_configs:
  [ - <eureka_sd_config> ... ]

# List of file service discovery configurations.
file_sd_configs:
  [ - <file_sd_config> ... ]

# List of GCE service discovery configurations.
gce_sd_configs:
  [ - <gce_sd_config> ... ]

# List of Hetzner service discovery configurations.
hetzner_sd_configs:
  [ - <hetzner_sd_config> ... ]

# List of Kubernetes service discovery configurations.
kubernetes_sd_configs:
  [ - <kubernetes_sd_config> ... ]

# List of Marathon service discovery configurations.
marathon_sd_configs:
  [ - <marathon_sd_config> ... ]

# List of AirBnB's Nerve service discovery configurations.
nerve_sd_configs:
  [ - <nerve_sd_config> ... ]

# List of OpenStack service discovery configurations.
openstack_sd_configs:
  [ - <openstack_sd_config> ... ]

# List of Zookeeper Serverset service discovery configurations.
serverset_sd_configs:
  [ - <serverset_sd_config> ... ]

# List of Triton service discovery configurations.
triton_sd_configs:
  [ - <triton_sd_config> ... ]

# List of labeled statically configured targets for this job.
static_configs:
  [ - <static_config> ... ]

# List of target relabel configurations.
relabel_configs:
  [ - <relabel_config> ... ]

# List of metric relabel configurations.
metric_relabel_configs:
  [ - <relabel_config> ... ]

# Per-scrape limit on number of scraped samples that will be accepted.
# If more than this number of samples are present after metric relabeling
# the entire scrape will be treated as failed. 0 means no limit.
[ sample_limit: <int> | default = 0 ]

# Per-scrape config limit on number of unique targets that will be
```

```
# accepted. If more than this number of targets are present after target
# relabeling, Prometheus will mark the targets as failed without scraping them.
# 0 means no limit. This is an experimental feature, this behaviour could
# change in the future.
[ target_limit: <int> | default = 0 ]
```

Where `<job_name>` must be unique across all scrape configurations.

`<tls_config>`

A `tls_config` allows configuring TLS connections.

```
# CA certificate to validate API server certificate with.
[ ca_file: <filename> ]

# Certificate and key files for client cert authentication to the server.
[ cert_file: <filename> ]
[ key_file: <filename> ]

# ServerName extension to indicate the name of the server.
# https://tools.ietf.org/html/rfc4366#section-3.1
[ server_name: <string> ]

# Disable validation of the server certificate.
[ insecure_skip_verify: <boolean> ]
```

`<kubernetes_sd_config>`

Kubernetes SD configurations allow retrieving scrape targets from Kubernetes' REST API and always staying synchronized with the cluster state.

One of the following `role` types can be configured to discover targets:

`node`

The `node` role discovers one target per cluster node with the address defaulting to the Kubelet's HTTP port. The target address defaults to the first existing address of the Kubernetes node object in the address type order of `NodeInternalIP`, `NodeExternalIP`, `NodeLegacyHostIP`, and `NodeHostName`.

Available meta labels:

- `__meta_kubernetes_node_name`: The name of the node object.
- `__meta_kubernetes_node_label_<labelname>`: Each label from the node object.
- `__meta_kubernetes_node_labelpresent_<labelname>`: `true` for each label from the node object.
- `__meta_kubernetes_node_annotation_<annotationname>`: Each annotation from the node object.
- `__meta_kubernetes_node_annotationpresent_<annotationname>`: `true` for each annotation from the node object.
- `__meta_kubernetes_node_address_<address_type>`: The first address for each node address type, if it exists.

In addition, the `instance` label for the node will be set to the node name as retrieved from the API server.

`service`

The `service` role discovers a target for each service port for each service. This is generally useful for blackbox monitoring of a service. The address will be set to the Kubernetes DNS name of the service and respective service port.

Available meta labels:

- `__meta_kubernetes_namespace`: The namespace of the service object.
- `__meta_kubernetes_service_annotation_<annotationname>`: Each annotation from the service object.
- `__meta_kubernetes_service_annotationpresent_<annotationname>`: "true" for each annotation of the service object.
- `__meta_kubernetes_service_cluster_ip`: The cluster IP address of the service. (Does not apply to services of type ExternalName)
- `__meta_kubernetes_service_external_name`: The DNS name of the service. (Applies to services of type ExternalName)
- `__meta_kubernetes_service_label_<labelname>`: Each label from the service object.
- `__meta_kubernetes_service_labelpresent_<labelname>`: `true` for each label of the service object.
- `__meta_kubernetes_service_name`: The name of the service object.
- `__meta_kubernetes_service_port_name`: Name of the service port for the target.
- `__meta_kubernetes_service_port_protocol`: Protocol of the service port for the target.
- `__meta_kubernetes_service_type`: The type of the service.

`pod`

The `pod` role discovers all pods and exposes their containers as targets. For each declared port of a container, a single target is generated. If a container has no specified ports, a port-free target per container is created for manually adding a port via relabeling.

Available meta labels:

- `__meta_kubernetes_namespace`: The namespace of the pod object.
- `__meta_kubernetes_pod_name`: The name of the pod object.
- `__meta_kubernetes_pod_ip`: The pod IP of the pod object.
- `__meta_kubernetes_pod_label_<labelname>`: Each label from the pod object.
- `__meta_kubernetes_pod_labelpresent_<labelname>`: `true` for each label from the pod object.
- `__meta_kubernetes_pod_annotation_<annotationname>`: Each annotation from the pod object.
- `__meta_kubernetes_pod_annotationpresent_<annotationname>`: `true` for each annotation from the pod object.
- `__meta_kubernetes_pod_container_init`: `true` if the container is an InitContainer
- `__meta_kubernetes_pod_container_name`: Name of the container the target address points to.
- `__meta_kubernetes_pod_container_port_name`: Name of the container port.
- `__meta_kubernetes_pod_container_port_number`: Number of the container port.
- `__meta_kubernetes_pod_container_port_protocol`: Protocol of the container port.
- `__meta_kubernetes_pod_ready`: Set to `true` or `false` for the pod's ready state.
- `__meta_kubernetes_pod_phase`: Set to `Pending`, `Running`, `Succeeded`, `Failed` or `Unknown` in the lifecycle.
- `__meta_kubernetes_pod_node_name`: The name of the node the pod is scheduled onto.

- `__meta_kubernetes_pod_host_ip`: The current host IP of the pod object.
- `__meta_kubernetes_pod_uid`: The UID of the pod object.
- `__meta_kubernetes_pod_controller_kind`: Object kind of the pod controller.
- `__meta_kubernetes_pod_controller_name`: Name of the pod controller.

`endpoints`

The `endpoints` role discovers targets from listed endpoints of a service. For each endpoint address one target is discovered per port. If the endpoint is backed by a pod, all additional container ports of the pod, not bound to an endpoint port, are discovered as targets as well.

Available meta labels:

- `__meta_kubernetes_namespace`: The namespace of the endpoints object.
- `__meta_kubernetes_endpoints_name`: The names of the endpoints object.
- For all targets discovered directly from the endpoints list (those not additionally inferred from underlying pods), the following labels are attached:
  - `__meta_kubernetes_endpoint_hostname`: Hostname of the endpoint.
  - `__meta_kubernetes_endpoint_node_name`: Name of the node hosting the endpoint.
  - `__meta_kubernetes_endpoint_ready`: Set to `true` or `false` for the endpoint's ready state.
  - `__meta_kubernetes_endpoint_port_name`: Name of the endpoint port.
  - `__meta_kubernetes_endpoint_port_protocol`: Protocol of the endpoint port.
  - `__meta_kubernetes_endpoint_address_target_kind`: Kind of the endpoint address target.
  - `__meta_kubernetes_endpoint_address_target_name`: Name of the endpoint address target.
- If the endpoints belong to a service, all labels of the `role: service` discovery are attached.
- For all targets backed by a pod, all labels of the `role: pod` discovery are attached.

`ingress`

The `ingress` role discovers a target for each path of each ingress. This is generally useful for blackbox monitoring of an ingress. The address will be set to the host specified in the ingress spec.

Available meta labels:

- `__meta_kubernetes_namespace`: The namespace of the ingress object.
- `__meta_kubernetes_ingress_name`: The name of the ingress object.
- `__meta_kubernetes_ingress_label_<labelname>`: Each label from the ingress object.
- `__meta_kubernetes_ingress_labelpresent_<labelname>`: `true` for each label from the ingress object.
- `__meta_kubernetes_ingress_annotation_<annotationname>`: Each annotation from the ingress object.
- `__meta_kubernetes_ingress_annotationpresent_<annotationname>`: `true` for each annotation from the ingress object.
- `__meta_kubernetes_ingress_scheme`: Protocol scheme of ingress, `https` if TLS config is set. Defaults to `http`.
- `__meta_kubernetes_ingress_path`: Path from ingress spec. Defaults to `/`.

See below for the configuration options for Kubernetes discovery:

```
# The information to access the Kubernetes API.

# The API server addresses. If left empty, Prometheus is assumed to run inside
# of the cluster and will discover API servers automatically and use the pod's
# CA certificate and bearer token file at
/var/run/secrets/kubernetes.io/serviceaccount/.
[ api_server: <host> ]

# The Kubernetes role of entities that should be discovered.
# One of endpoints, service, pod, node, or ingress.
role: <string>

# Optional authentication information used to authenticate to the API server.
# Note that `basic_auth`, `bearer_token` and `bearer_token_file` options are
# mutually exclusive.
# password and password_file are mutually exclusive.

# Optional HTTP basic authentication information.
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# Optional bearer token authentication information.
[ bearer_token: <secret> ]

# Optional bearer token file authentication information.
[ bearer_token_file: <filename> ]

# Optional proxy URL.
[ proxy_url: <string> ]

# TLS configuration.
tls_config:
  [ <tls_config> ]

# Optional namespace discovery. If omitted, all namespaces are used.
namespaces:
  names:
    [ - <string> ]

# Optional label and field selectors to limit the discovery process to a subset
of available resources.
# See https://kubernetes.io/docs/concepts/overview/working-with-objects/field-
selectors/
# and https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
to learn more about the possible
# filters that can be used. Endpoints role supports pod, service and endpoints
selectors, other roles
# only support selectors matching the role itself (e.g. node role can only
contain node selectors).

# Note: When making decision about using field/label selector make sure that this
# is the best approach - it will prevent Prometheus from reusing single
list/watch
# for all scrape configs. This might result in a bigger load on the Kubernetes
API,
```

```
# because per each selector combination there will be additional LIST/WATCH. On
the other hand,
# if you just want to monitor small subset of pods in large cluster it's
recommended to use selectors.
# Decision, if selectors should be used or not depends on the particular
situation.
[ selectors:
  [ - role: <string>
    [ label: <string> ]
    [ field: <string> ] ]]
```

See this example Prometheus configuration file for a detailed example of configuring Prometheus for Kubernetes.

You may wish to check out the 3rd party Prometheus Operator, which automates the Prometheus setup on top of Kubernetes.

`<static_config>`

A `static_config` allows specifying a list of targets and a common label set for them. It is the canonical way to specify static targets in a scrape configuration.

```
# The targets specified by the static config.
targets:
  [ - '<host>' ]

# Labels assigned to all metrics scraped from the targets.
labels:
  [ <labelname>: <labelvalue> ... ]
```

`<relabel_config>`

Relabeling is a powerful tool to dynamically rewrite the label set of a target before it gets scraped. Multiple relabeling steps can be configured per scrape configuration. They are applied to the label set of each target in order of their appearance in the configuration file.

Initially, aside from the configured per-target labels, a target's `job` label is set to the `job_name` value of the respective scrape configuration. The `__address__` label is set to the `<host>:<port>` address of the target. After relabeling, the `instance` label is set to the value of `__address__` by default if it was not set during relabeling. The `__scheme__` and `__metrics_path__` labels are set to the scheme and metrics path of the target respectively. The `__param_<name>` label is set to the value of the first passed URL parameter called `<name>`.

Additional labels prefixed with `__meta_` may be available during the relabeling phase. They are set by the service discovery mechanism that provided the target and vary between mechanisms.

Labels starting with `__` will be removed from the label set after target relabeling is completed.

If a relabeling step needs to store a label value only temporarily (as the input to a subsequent relabeling step), use the `__tmp` label name prefix. This prefix is guaranteed to never be used by Prometheus itself.

```
# The source labels select values from existing labels. Their content is
concatenated
```

```
# using the configured separator and matched against the configured regular
expression
# for the replace, keep, and drop actions.
[ source_labels: '[' <labelname> [, ...] ']' ]

# Separator placed between concatenated source label values.
[ separator: <string> | default = ; ]

# Label to which the resulting value is written in a replace action.
# It is mandatory for replace actions. Regex capture groups are available.
[ target_label: <labelname> ]

# Regular expression against which the extracted value is matched.
[ regex: <regex> | default = (.*) ]

# Modulus to take of the hash of the source label values.
[ modulus: <int> ]

# Replacement value against which a regex replace is performed if the
# regular expression matches. Regex capture groups are available.
[ replacement: <string> | default = $1 ]

# Action to perform based on regex matching.
[ action: <relabel_action> | default = replace ]
```

`<regex>` is any valid RE2 regular expression. It is required for the `replace`, `keep`, `drop`, `labelmap`, `labeldrop` and `labelkeep` actions. The regex is anchored on both ends. To un-anchor the regex, use `.*<regex>.*`.

`<relabel_action>` determines the relabeling action to take:

- `replace`: Match `regex` against the concatenated `source_labels`. Then, set `target_label` to `replacement`, with match group references (`${1}`, `${2}`, ...) in `replacement` substituted by their value. If `regex` does not match, no replacement takes place.
- `keep`: Drop targets for which `regex` does not match the concatenated `source_labels`.
- `drop`: Drop targets for which `regex` matches the concatenated `source_labels`.
- `hashmod`: Set `target_label` to the `modulus` of a hash of the concatenated `source_labels`.
- `labelmap`: Match `regex` against all label names. Then copy the values of the matching labels to label names given by `replacement` with match group references (`${1}`, `${2}`, ...) in `replacement` substituted by their value.
- `labeldrop`: Match `regex` against all label names. Any label that matches will be removed from the set of labels.
- `labelkeep`: Match `regex` against all label names. Any label that does not match will be removed from the set of labels.

Care must be taken with `labeldrop` and `labelkeep` to ensure that metrics are still uniquely labeled once the labels are removed.

`<metric_relabel_configs>`

Metric relabeling is applied to samples as the last step before ingestion. It has the same configuration format and actions as target relabeling. Metric relabeling does not apply to automatically generated timeseries such as `up`.

One use for this is to exclude time series that are too expensive to ingest.

<alert_relabel_configs>

Alert relabeling is applied to alerts before they are sent to the Alertmanager. It has the same configuration format and actions as target relabeling. Alert relabeling is applied after external labels.

One use for this is ensuring a HA pair of Prometheus servers with different external labels send identical alerts.

<alertmanager_config>

An `alertmanager_config` section specifies Alertmanager instances the Prometheus server sends alerts to. It also provides parameters to configure how to communicate with these Alertmanagers.

Alertmanagers may be statically configured via the `static_configs` parameter or dynamically discovered using one of the supported service-discovery mechanisms.

Additionally, `relabel_configs` allow selecting Alertmanagers from discovered entities and provide advanced modifications to the used API path, which is exposed through the `__alerts_path__` label.

```
# Per-target Alertmanager timeout when pushing alerts.
[ timeout: <duration> | default = 10s ]

# The api version of Alertmanager.
[ api_version: <string> | default = v1 ]

# Prefix for the HTTP path alerts are pushed to.
[ path_prefix: <path> | default = / ]

# Configures the protocol scheme used for requests.
[ scheme: <scheme> | default = http ]

# Sets the `Authorization` header on every request with the
# configured username and password.
# password and password_file are mutually exclusive.
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# Sets the `Authorization` header on every request with
# the configured bearer token. It is mutually exclusive with
`bearer_token_file`.
[ bearer_token: <string> ]

# Sets the `Authorization` header on every request with the bearer token
# read from the configured file. It is mutually exclusive with `bearer_token`.
[ bearer_token_file: <filename> ]

# Configures the scrape request's TLS settings.
tls_config:
  [ <tls_config> ]

# Optional proxy URL.
[ proxy_url: <string> ]
```

```
# List of Azure service discovery configurations.
azure_sd_configs:
  [ - <azure_sd_config> ... ]

# List of Consul service discovery configurations.
consul_sd_configs:
  [ - <consul_sd_config> ... ]

# List of DNS service discovery configurations.
dns_sd_configs:
  [ - <dns_sd_config> ... ]

# List of EC2 service discovery configurations.
ec2_sd_configs:
  [ - <ec2_sd_config> ... ]

# List of Eureka service discovery configurations.
eureka_sd_configs:
  [ - <eureka_sd_config> ... ]

# List of file service discovery configurations.
file_sd_configs:
  [ - <file_sd_config> ... ]

# List of DigitalOcean service discovery configurations.
digitalocean_sd_configs:
  [ - <digitalocean_sd_config> ... ]

# List of Docker Swarm service discovery configurations.
dockerswarm_sd_configs:
  [ - <dockerswarm_sd_config> ... ]

# List of GCE service discovery configurations.
gce_sd_configs:
  [ - <gce_sd_config> ... ]

# List of Hetzner service discovery configurations.
hetzner_sd_configs:
  [ - <hetzner_sd_config> ... ]

# List of Kubernetes service discovery configurations.
kubernetes_sd_configs:
  [ - <kubernetes_sd_config> ... ]

# List of Marathon service discovery configurations.
marathon_sd_configs:
  [ - <marathon_sd_config> ... ]

# List of AirBnB's Nerve service discovery configurations.
nerve_sd_configs:
  [ - <nerve_sd_config> ... ]

# List of OpenStack service discovery configurations.
openstack_sd_configs:
  [ - <openstack_sd_config> ... ]

# List of Zookeeper Serverset service discovery configurations.
serverset_sd_configs:
```

```
  [ - <serverset_sd_config> ... ]

# List of Triton service discovery configurations.
triton_sd_configs:
  [ - <triton_sd_config> ... ]

# List of labeled statically configured Alertmanagers.
static_configs:
  [ - <static_config> ... ]

# List of Alertmanager relabel configurations.
relabel_configs:
  [ - <relabel_config> ... ]
```

`<remote_write>`

`write_relabel_configs` is relabeling applied to samples before sending them to the remote endpoint. Write relabeling is applied after external labels. This could be used to limit which samples are sent.

There is a small demo of how to use this functionality.

```
# The URL of the endpoint to send samples to.
url: <string>

# Timeout for requests to the remote write endpoint.
[ remote_timeout: <duration> | default = 30s ]

# List of remote write relabel configurations.
write_relabel_configs:
  [ - <relabel_config> ... ]

# Name of the remote write config, which if specified must be unique among remote
write configs.
# The name will be used in metrics and logging in place of a generated value to
help users distinguish between
# remote write configs.
[ name: <string> ]

# Sets the `Authorization` header on every remote write request with the
# configured username and password.
# password and password_file are mutually exclusive.
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# Sets the `Authorization` header on every remote write request with
# the configured bearer token. It is mutually exclusive with
`bearer_token_file`.
[ bearer_token: <string> ]

# Sets the `Authorization` header on every remote write request with the bearer
token
# read from the configured file. It is mutually exclusive with `bearer_token`.
[ bearer_token_file: <filename> ]
```

```
# Configures the remote write request's TLS settings.
tls_config:
  [ <tls_config> ]

# Optional proxy URL.
[ proxy_url: <string> ]

# Configures the queue used to write to remote storage.
queue_config:
  # Number of samples to buffer per shard before we block reading of more
  # samples from the WAL. It is recommended to have enough capacity in each
  # shard to buffer several requests to keep throughput up while processing
  # occasional slow remote requests.
  [ capacity: <int> | default = 2500 ]
  # Maximum number of shards, i.e. amount of concurrency.
  [ max_shards: <int> | default = 200 ]
  # Minimum number of shards, i.e. amount of concurrency.
  [ min_shards: <int> | default = 1 ]
  # Maximum number of samples per send.
  [ max_samples_per_send: <int> | default = 500]
  # Maximum time a sample will wait in buffer.
  [ batch_send_deadline: <duration> | default = 5s ]
  # Initial retry delay. Gets doubled for every retry.
  [ min_backoff: <duration> | default = 30ms ]
  # Maximum retry delay.
  [ max_backoff: <duration> | default = 100ms ]

# Configures the sending of series metadata to remote storage.
# Metadata configuration is subject to change at any point
# or be removed in future releases.
metadata_config:
  # Whether metric metadata is sent to remote storage or not.
  [ send: <boolean> | default = true ]
  # How frequently metric metadata is sent to remote storage.
  [ send_interval: <duration> | default = 1m ]
```

There is a list of integrations with this feature.

`<remote_read>`

```
# The URL of the endpoint to query from.
url: <string>

# Name of the remote read config, which if specified must be unique among remote
read configs.
# The name will be used in metrics and logging in place of a generated value to
help users distinguish between
# remote read configs.
[ name: <string> ]

# An optional list of equality matchers which have to be
# present in a selector to query the remote read endpoint.
required_matchers:
  [ <labelname>: <labelvalue> ... ]

# Timeout for requests to the remote read endpoint.
[ remote_timeout: <duration> | default = 1m ]
```

```
# Whether reads should be made for queries for time ranges that
# the local storage should have complete data for.
[ read_recent: <boolean> | default = false ]

# Sets the `Authorization` header on every remote read request with the
# configured username and password.
# password and password_file are mutually exclusive.
basic_auth:
  [ username: <string> ]
  [ password: <secret> ]
  [ password_file: <string> ]

# Sets the `Authorization` header on every remote read request with
# the configured bearer token. It is mutually exclusive with
`bearer_token_file`.
[ bearer_token: <string> ]

# Sets the `Authorization` header on every remote read request with the bearer
token
# read from the configured file. It is mutually exclusive with `bearer_token`.
[ bearer_token_file: <filename> ]

# Configures the remote read request's TLS settings.
tls_config:
  [ <tls_config> ]

# Optional proxy URL.
[ proxy_url: <string> ]
```

Установка приложения на сбор метрик

В конфиге Прометея есть готовая джоба:

```
- job_name: kubernetes-service-endpoints
```

Используется `kubernetes_sd_configs`, роль `endpoints`.

Это джоба для сбора метрик с конкретных заранее известных эндпойнтов.

Внутри этой джобы можно использовать `relabel_configs` для фильтрации нужных нам эндпойнтов.

Можно настроить фильтрацию на основе аннотаций.

Например - можно сделать

```
kubectl describe svc ... -o yaml
```

и взять аннотации конкретного объекта, и их с помощью регекспов вставить в конфиг Прометея.

Пример:

```
relabel_configs:
- action: keep
  regex: true
  source_labels:
  - <наша аннотация>
```

По умолчанию endpoint, с которого Прометей собирает метрики - `/metrics` и собирает он по `http`. Если нужно переопределить путь этого эндпойнта, или поменять на HTTPS, также **можно использовать аннотации**:

```
relabel_configs:
- action: replace
  regex: (https?)
  source_labels:
  - __meta_kubernetes_service_annotation_prometheus_io_scheme
  target_label: __scheme__
- action: replace
  regex: (.+)
  source_labels:
  - __meta_kubernetes_service_annotation_prometheus_io_path
  target_label: __metrics_path__
```

`__meta_kubernetes_service_annotation_` - это служебный префикс, после которого идет имя аннотации, которая проверяется.

`__scheme__` - это как раз протокол и порт по умолчанию, с которого Прометей собирает метрики.

`__metrics_path__` - эндпойнт, с которого собираются метрики.

Если в сервисе есть следующая аннотация, то Прометей опознает, что с этого эндпойнта можно собирать метрики:

```
annotations: prometheus.io/scrape: true
```

Мониторинг в Kubernetes на BlackBox

В отличие от `kubernetes-service-endpoints`, еще одна джоба в конфиге - `kubernetes-services` - предназначена для Blackbox Monitoring.

То есть в целом понимать доступность приложения снаружи и собирать какие-то базовые метрики - скорость доступа, скорость обработки запросов.

Используется уже роль service:

```
kubernetes_sd_configs:
- role: service
metrics_path: /probe
params:
  module:
  - http_2xx
relabel_configs:
- action: keep
  regex: true
  source_labels:
  - __meta_kubernetes_service_annotation_prometheus_io_probe
```

С помощью этого модуля через **Blackbox Exporter** по пути /probe смотреть доступность сервиса.

Проверяться будут те сервисы, у которых в аннотации есть `prometheus_io/probe`

Blackbox exporter надо поставить в лабу самому:

```
$ helm search hub blackbox
//искать надо в репозитории /prometheus-community
```

Собственно потом в любом сервисе в аннотации ставим `prometheus_io/probe: "true"`

После этого по прошествии времени в `scrape_interval` Прометей обнаружит новый эндпойнт и задействует джобу `kubernetes-services`.

Promtool и советы по Prometheus

С помощью `promtool` можно проверять правильность конфигов, правил, алертов и других объектов в Prometheus.

Это полезно внедрять в процесс CI для автоматизированных проверок.

Также полезно и разработчикам, поскольку может валидировать метрики, в том числе и прикладные.

Отдельно установить его сложно, но его можно выдернуть из архива prometheus - promtool идет просто бинарным файлом.

Полезные опции:

`check config` - проверка конфига

`check rules` - проверка правила

`curl -s <адрес сервиса>/metrics | promtool check metrics` - перенаправление и проверка метрик

`tsdb dump` - дамп базы и ее анализ

`debug all <server>` - получить дебаг информацию и траблшутинг Прометея

Советы

- Всегда выставлять реквесты и лимиты для Прометея, плюс отслеживать потребление по диску и памяти
- Не нужно использовать тег `:latest` в образе Прометея, иначе что-то может сломаться, так как может измениться синтаксис, флаги, опции и т.д.
- Сам по себе Прометей не очень отказоустойчив, даже в режиме федерации. Самый распространенный выход - запускать несколько инстансов Прометея и дублировать собранные метрики на каком-то внешнем хранилище (Victoria Metrics)
- Специфика TSDB - для долгосрочного хранения эта база не предназначена. Не стоит задавать `retention_time` больше дефолтных 15 дней. Производительность Прометея будет падать с ростом объема хранения данных.

Продвинутый Prometheus

Построение федерации

Федерация - это специализированный эндпойнт Прометея, при обращении на который можно получить список всех метрик и их значения в момент обращения.

Это удобно для построения максимально удобной системы алертинга и визуализации.

Либо для реализации некой конструкции типа шардирования, где несколько Прометеев специализированно мониторят разные подсистемы либо разные сегменты сети.

Служебные лейблы `instance` и `job` - добавляются Прометеем в процессе скрейпинга. Вышестоящий Прометей, забирая метрики у нижестоящего, их затрёт, и это не то, чего хотелось бы. Решается это специализированной настройкой:

```
honor_labels: true
```

Эта настройка говорит Прометею сохранит оригинальное значение всех лейблов.

Еще момент - как узнать о том, с какого именно нижестоящего Прометея была получена та или иная метрика. Для этого есть специализированная настройка, выставляющаяся на нижестоящих Прометеях:

```
global: external_labels
  prom: prom-0
```

Допустим, есть 2 node-exporter-а, с которых Прометей с именем `prom-0` скрейпит данные каждые 30 секунд.

Вышестоящий Прометей забирает с него метрики по пути `/federation` каждые 3 минуты.

За 3 минуты нижестоящий Прометей 6 раз успеет заскрейпить данные и сохранит 6 временных рядов. Но отдаст по запросу вышестоящего **только последний временной ряд, а не все 6!**

Настройка Federation в k8s кластере

Цели практики

Целью данной практической работы является демонстрация процесса настройки и работы Prometheus в режиме Federation. В рамках данного урока в Kubernetes кластер будет установлен и настроен Prometheus Server таким образом, что он будет собирать данные с уже установленного Prometheus сервера.

Установка Prometheus будет производиться с помощью Helm Chart от prometheus community. Для удобства все остальные компоненты будут отключены.

С чем будем работать

- Helm chart prometheus community - необходим для установки Prometheus server в кластер

Где будем выполнять практику

ВНИМАНИЕ: Перед тем, как работать со стендом, его необходимо запустить из Личного кабинета

- Подключаемся по SSH к **sbox.slurm.io**:

`ssh s<номер вашего логина>@sbox.slurm.io`

- Подключаемся к своему кластеру Kubernetes:

`ssh master-1.s<ваш номер логина>`

- Становимся root:

`sudo -s`

Практика

Подготовительная работа

1. Клонируем манифесты

Для выполнения практической части потребуются манифесты, их необходимо склонировать с Gitlab. Сделать это можно, выполнив команды:

```
cd ~
git clone git@gitlab.slurm.io:edu/lmk8s.git
```

2. Проверяем репо prometheus-community:

```
helm repo list | grep prometheus
```

Репозиторий уже был добавлен автоматически при создании стенда.

3. Создаем namespace

В кластере уже установлен Prometheus server в namespace: `monitoring`. В рамках данного урока Prometheus будет устанавливаться также в отдельный namespace: `prom-federation`. Для его создания необходимо выполнить команду:

```
kubectl create ns prom-federation
```

4. Создание secret для basic auth

Также как и основной Prometheus, новый сервер будет доступен через Ingress и доступ к нему будет ограничен basic auth. Для этого необходимо создать Secret, который содержит данные для basic auth. Для этого в подготовленном манифесте:
`~/lmk8s/4.prometheus_advanced/4.1.prometheus_federation/basic-auth-secret.yml`
необходимо подставить ваши данные аутентификации. Это можно сделать, выполнив команду:

```
sed -i "s/<basic auth>/$(k get secrets -n monitoring basic-auth -o
jsonpath='{.data.auth}')/g"
~/lmk8s/4.prometheus_advanced/4.1.prometheus_federation/basic-auth-secret.yml
```

В результате выполнения данной команды будет получены данные, необходимые для аутентификации, из уже созданного Secret, для основного Prometheus и подставлены в манифест. Далее этот манифест необходимо применить в кластер, выполнив команду:

```
kubectl apply -f ~/lmk8s/4.prometheus_advanced/4.1.prometheus_federation/basic-
auth-secret.yml -n prom-federation
```

5. Подготовка values

Для установки Prometheus будет использоваться уже подготовленные значения values:
`~/lmk8s/4.prometheus_advanced/4.1.prometheus_federation/federation.yml`

Но в них необходимо заменить адрес Ingress для Prometheus, это можно сделать выполнив команду:

```
sed -i 's/<student name>/Ваш номер студента, например s00000/g'
~/lmk8s/4.prometheus_advanced/4.1.prometheus_federation/federation.yml
```

В данных values уже настроен сбор данных по federation. Все сводится к этим строкам:

```
- job_name: federation
        scrape_interval: 30s

        metrics_path: '/federate'

        params:
          match[]:
            - '{job=~".+"}'
        static_configs:
          - targets:
            - prometheus-server.monitoring.svc:80
```

Тут надо обратить внимание на поля:

- match - является обязательным полем, и в нем указывается фильтр по labels, какие метрики мы хотим получать. Через этот параметр может быть ограничен набор метрик, которые забираются с нижестоящего Prometheus.
- targets - в нем указано обращение к Prometheus через имя сервиса.

Установка Prometheus в кластер

Теперь все готово для установки, и достаточно выполнить команду:

```
helm upgrade -i prom-federation -n prom-federation -f
~/lmk8s/4.prometheus_advanced/4.1.prometheus_federation/federation.yml
prometheus-community/prometheus
```

После установки необходимо проверить, что все работает корректно:

- Открываем в браузере в анонимном режиме URL `http://federation.<номер студента>.edu.slurm.io`
- Выполняем запрос: `node_cpu_seconds_total`

В результате выполнения данного запроса вам должны быть выведены данные по всем node в кластере.

Подключение внешних Long-term storage (Victoria Metrics)

Есть большое количество интеграций с внешними системами. Один пример - это Victoria Metrics.

В Викторию метрики отправляются через `remote_write`

Виктория может ставиться в 2 режимах - **single mode** и **cluster mode**. В сингле все компоненты идут в одном бинарнике, в кластерном - разбиты по разным бинарникам:

- `vmStorage` - это реализация TSDB для хранения метрик. Писать напрямую в TSDB нельзя. Это stateful приложение, но его можно масштабировать через шардирование.
- `vmInsert` - принимает метрики через `remote_write` пишет в vmStorage. Это stateless приложение и его можно горизонтально масштабировать. Плюс к тому - получив вектор с метриками, vmInsert может реплицировать его сразу в несколько шардов vmStorage.
- `vmSelect` - компонент для извлечения данных из vmStorage. Он же выполняет дедупликацию, это stateless приложение и тоже может горизонтально масштабироваться
- `HAProxy/Service` - ставится перед экземплярами vmSelect в случае их множественного числа. К HAProxy уже стучится Grafana для визуализации.
- `vmAuth` - аутентификация перед vmInsert, может создавать различные УЗ для различных тенантов, чтобы каждый Прометей писал в свой тенант.
- `vmBackup` - резервное копирование для vmStorage, как на файловую систему, так и во внешние клауд-провайдеры (точно поддерживаются AWS, GCP)
- `vmRestore` - восстановление из бекапа
- `vmAlert` + `vmalert-cli` - некий аналог Alertmanager-а. Синтаксис и логика работы тоже такая же, как у Alertmanager.
- `vmAgent` - роль, похожая на Prometheus Server - делает скрейпинг метрик с разных источников.

Тенанты - логическое разделение vmStorage на несколько баз данных (как namespace в K8S). Данные при этом не пересекаются друг с другом.

Изначально, когда Victoria Metrics разрабатывалась - задумка была разработать полный стек мониторинга, хранения и алертинга для замены Прометею. Отсюда и такой широкий набор компонентов.

Установка Victoria Metrics

Цели практики

Целью данной практической работы является демонстрация процесса настройки и работы Victoria Metrics. В рамках данного урока в Kubernetes кластере будет установлен и настроен Victoria Metrics в режиме Cluster, а также будет проведена настройка уже установленных в кластере Grafana и Prometheus.

Установка Victoria Metrics будет производиться с помощью Helm Chart.

С чем будем работать

- Helm chart victoria metrics - необходим для установки Prometheus server в кластер

Где будем выполнять практику

ВНИМАНИЕ: Перед тем, как работать со стендом, его необходимо запустить из Личного кабинета

- Подключаемся по SSH к sbox.slurm.io:

`ssh s<номер вашего логина>@sbox.slurm.io`

- Подключаемся к своему кластеру Kubernetes:

`ssh master-1.s<ваш номер логина>`

- Становимся root:

`sudo -s`

Практика

Подготовительная работа

1. Клонируем манифесты

Для выполнения практической части, потребуются манифесты. Их необходимо складировать с gitlab, сделать это можно выполнив команды:

```
cd ~
git clone git@gitlab.slurm.io:edu/lmk8s.git
```

2. Подключаем репо Victoria Metrics

```
helm repo add vm https://victoriametrics.github.io/helm-charts/
helm repo update
```

3. Создаем namespace

Victoria Metrics будет устанавливаться в отдельный namespace: victoria. Для его создания, необходимо выполнить команду:

```
kubectl create ns victoria
```

Установка Victoria Metrics в кластер

Теперь все готово для установки и достаточно выполнить команду:

```
helm upgrade -i vm-cluster -n victoria -f
~/lmk8s/4.prometheus_advanced/4.2.victoria_metrics/values.yml vm/victoria-
metrics-cluster
```

В результате будут установлены следующие компоненты:

- vmSelect - будет установлен как Deployment с значением replicas: 2
- vmInsert - будет установлен как Deployment с значением replicas: 2
- vmStorage - будет установлен как StatefulSet с значением replicas: 2

Ключевые изменения в values.yml

Обратите внимание, что в values.yml для чарта были сделаны изменения, вот наиболее ключевые из них:

- **Заданы ресурсы для все объектов** - это общее хорошее правило для всего, что запускается в Kubernetes кластере. Во первых, заданные requests и limits позволяют scheduler корректно распределять Pod по node. Во вторых, это важно для избегания ситуации утечки ресурсов, а в случае с Prometheus это особенно актуально, так как он является достаточно "прожорливым".
- Задан **podDisruptionBudget** - данная настройка влияет на то, сколько Pod может быть одновременно выключено (распространяется только на eviction API). Она позволяет гарантировать, что при обслуживании кластера Kubernetes не будут выключены все Pod'ы с приложением.
- **podAntiAffinity** - данная настройка позволяет гарантировать, что Pod из StatefulSet и Pod из Deployment не будут запущены на одной node. Без данной настройки возможна ситуация, когда все Pod будут запущены на одной node, и в случае выхода из строя этой node, кластер останется без мониторинга (Pod для statefulset не перезапускаются автоматически)
- **vmselect.extraArgs** - добавлены следующие параметры:

`search.maxQueueDuration``: "10s"` - ограничивает время выполнения запросов. Когда VictoriaMetrics пользуется большое число команд, это позволяет отсекать слишком длительные запросы. Ведь они, в свою очередь, могут сказаться на производительности всей БД.

`search.maxQueryLen``: "16384"` - ограничивает максимальную длину запроса. Опять же, когда VictoriaMetrics пользуется сразу несколько команд, это позволяет отсекать слишком большие запросы, которые могут сказаться на производительности БД.

`replicationFactor``: "2"` - задает сколько копий time series имеют доступ в vmStorage. Должен совпадать со значением для vmInsert.

`dedup.minScrapeInterval``: "1ms"` - data samples, которое отличаются на значение, указанное в этом поле и менее, будут "дедуплицированы". Этот параметр обязательно использовать при значении `replicationFactor` 2 и больше.

- **vmInsert.extraArgs** - добавлены следующие параметры:

`replicationFactor``: "2"` - задает сколько копий time series должны быть сохранены в vmStorage.

- **vmStorage.StorageClass** - задает имя StorageClass. При этом в default values этого поля вообще нет, а вот в Helm template он используется.

В поле extraArgs для каждого компонента задаются ключи запуска для этого компонента. При этом, большинство значений по умолчанию являются оптимальными. Полный список ключей запуска указан в последнем уроке данного раздела.

Настройка Prometheus

Prometheus уже установлен в кластере, установка производилась с помощью Helm Chart. Для внесения изменений, необходимо получить текущие values. Сделать это можно выполнив команду:

```
helm get values -n monitoring prometheus > prometheus_values.yml
```

В результате values будут сохранены в файл: `prometheus_values.yml`. Этот файл необходимо открыть на редактирование, найти в нем строку: `remoteWrite: []` и заменить ее на:

```
remoteWrite:
```

`- url: http://vm-cluster-victoria-metrics-cluster-vminsert.victoria.svc:8480/insert/0/prometheus/api/v1/write`

После этого изменения необходимо применить в кластер, выполнив команду:

`helm upgrade prometheus prometheus-community/prometheus -n monitoring -f prometheus_values.yml```

URL, куда отправляет данные Prometheus сервер, имеет следующую структуру: `/<component>/<clientID>/prometheus/<Prometheus API query>`

- `<component>` - префикс для компонента. vmInsert - `/insert`, vmSelect - `/select`
- `<clientID>` - ID клиента, может быть произвольным int и должен совпадать для insert и select запросов. За счет этого реализуется механизм tenant, когда в одну Victoria Metrics могут писать различные Prometheus сервера и их данные будут хранится в отдельных tenants (аналог namespace)
- `<Prometheus API query>` - исходный запрос Prometheus.

Данная структура используется только для Victoria Metrics в режиме **Cluster**. Если вы используете Victoria Metrics в режиме **Single Node**, то запросы будут иметь структуру: `/<Prometheus API query>`

Настройка Grafana

Grafana уже установлена в кластер, для ее настройки необходимо создать datasource и импортировать dashboard для node_exporter. Для этого достаточно применить два манифеста в кластер. Подробнее как это работает, будет рассмотрено в следующих уроках.

```
kubectl create -f ~/lmk8s/4.prometheus_advanced/4.2.victoria_metrics/grafana/ -n monitoring
```

> Для того, чтобы Grafana увидела новый datasource, нужно перезапустить её Pod. Сделать это можно командой:
>
> `kubectl delete po -n monitoring grafana-`

На этом всё! Можно проверять результат, для этого откройте в браузере в анонимном режиме адрес `http://grafana.<номер студента>.edu.slurm.io`. Имя пользователя и пароль совпадают с доступами в Личный кабинет.

Ключи запуска компонентов Victoria Metrics

Ключи запуска отсутствуют в документации и единственный способ их получить - это запуск соответствующего компонента с ключом `--help`. А при условии, что основной способ распространения Victoria Metrics это docker-образы, это не самый удобный способ. Ниже предоставлены ключи для основных компонентов:

vmInsert

```
-csvTrimTimestamp duration
```

Trim timestamps when importing csv data to this duration. Minimum practical duration is 1ms. Higher duration (i.e. 1s) may be used for reducing disk space usage for timestamp data (default 1ms)

```
-enableTCP6
```

Whether to enable IPv6 for listening and dialing. By default only IPv4 TCP is used

```
-envflag.enable
```

Whether to enable reading flags from environment variables additionally to command line. Command line flag values have priority over values from environment vars. Flags are read only from command line if this flag isn't set

```
-envflag.prefix string
```

Prefix for environment variables if -envflag.enable is set

```
-fs.disableMmap
```

Whether to use pread() instead of mmap() for reading data files. By default mmap() is used for 64-bit arches and pread() is used for 32-bit arches, since they cannot read data files bigger than 2^32 bytes in memory. mmap() is usually faster for reading small data chunks than pread()

```
-graphiteListenAddr string
```

TCP and UDP address to listen for Graphite plaintext data. Usually :2003 must be set. Doesn't work if empty

```
-graphiteTrimTimestamp duration
```

Trim timestamps for Graphite data to this duration. Minimum practical duration is 1s. Higher duration (i.e. 1m) may be used for reducing disk space usage for timestamp data (default 1s)

```
-http.connTimeout duration
```

Incoming http connections are closed after the configured timeout. This may help spreading incoming load among a cluster of services behind load balancer. Note that the real timeout may be bigger by up to 10% as a protection from Thundering herd problem (default 2m0s)

```
-http.disableResponseCompression
```

Disable compression of HTTP responses for saving CPU resources. By default compression is enabled to save network bandwidth

```
-http.idleConnTimeout duration
```

Timeout for incoming idle http connections (default 1m0s)

```
-http.maxGracefulShutdownDuration duration
```

The maximum duration for graceful shutdown of HTTP server. Highly loaded server may require increased value for graceful shutdown (default 7s)

```
-http.pathPrefix string
```

An optional prefix to add to all the paths handled by http server. For example, if '-http.pathPrefix=/foo/bar' is set, then all the http requests will be handled on '/foo/bar/*' paths. This may be useful for proxied requests. See https://www.robustperception.io/using-external-urls-and-proxies-with-prometheus

```
-http.shutdownDelay duration
```

Optional delay before http server shutdown. During this dealy the servier returns non-OK responses from /health page, so load balancers can route new requests to other servers

```
-httpListenAddr string
```

Address to listen for http connections (default ":8480")

```
-import.maxLineLen max_rows_per_line
```

The maximum length in bytes of a single line accepted by /api/v1/import; the line length can be limited with max_rows_per_line query arg passed to /api/v1/export
Supports the following optional suffixes for values: KB, MB, GB, KiB, MiB, GiB (default 104857600)

```
-influx.maxLineSize value
```

The maximum size in bytes for a single Influx line during parsing
Supports the following optional suffixes for values: KB, MB, GB, KiB, MiB, GiB (default 262144)

```
-influxListenAddr http://<vminsert>:8480/insert/<accountID>/influx/write
```

TCP and UDP address to listen for Influx line protocol data. Usually :8189 must be set. Doesn't work if empty. This flag isn't needed when ingesting data over HTTP - just send it to http://:8480/insert//influx/write

```
-influxMeasurementFieldSeparator string
```

Separator for '{measurement}{separator}{field_name}' metric name when inserted via Influx line protocol (default "_")

```
-influxSkipMeasurement
```

Uses '{field_name}' as a metric name while ignoring '{measurement}' and '-influxMeasurementFieldSeparator'

```
-influxSkipSingleField
```

Uses '{measurement}' instead of '{measurement}{separator}{field_name}' for metic name if Influx line contains only a single field

```
-influxTrimTimestamp duration
```

Trim timestamps for Influx line protocol data to this duration. Minimum practical duration is 1ms. Higher duration (i.e. 1s) may be used for reducing disk space usage for timestamp data (default 1ms)

```
-insert.maxQueueDuration duration
```

The maximum duration for waiting in the queue for insert requests due to -maxConcurrentInserts (default 1m0s)

```
-loggerDisableTimestamps
```

Whether to disable writing timestamps in logs

```
-loggerErrorsPerSecondLimit int
```

Per-second limit on the number of ERROR messages. If more than the given number of errors are emitted per second, then the remaining errors are suppressed. Zero value disables the rate limit

```
-loggerFormat string
```

Format for logs. Possible values: default, json (default "default")

```
-loggerLevel string
```

Minimum level of errors to log. Possible values: INFO, WARN, ERROR, FATAL, PANIC (default "INFO")

```
-loggerOutput string
```

Output for the logs. Supported values: stderr, stdout (default "stderr")

```
-loggerWarnsPerSecondLimit int
```

Per-second limit on the number of WARN messages. If more than the given number of warns are emitted per second, then the remaining warns are suppressed. Zero value disables the rate limit

```
-maxConcurrentInserts int
```

The maximum number of concurrent inserts. Default value should work for most cases, since it minimizes the overhead for concurrent inserts. This option is tigthly coupled with -insert.maxQueueDuration (default 24)

```
-maxInsertRequestSize value
```

The maximum size in bytes of a single Prometheus remote_write API request
Supports the following optional suffixes for values: KB, MB, GB, KiB, MiB, GiB (default 33554432)

```
-maxLabelsPerTimeseries int
```

The maximum number of labels accepted per time series. Superfluous labels are dropped (default 30)

```
-memory.allowedBytes value
```

Allowed size of system memory VictoriaMetrics caches may occupy. This option overrides -memory.allowedPercent if set to non-zero value. Too low value may increase cache miss rate, which usually results in higher CPU and disk IO usage. Too high value may evict too much data from OS page cache, which will result in higher disk IO usage
Supports the following optional suffixes for values: KB, MB, GB, KiB, MiB, GiB (default 0)

```
-memory.allowedPercent float
```

Allowed percent of system memory VictoriaMetrics caches may occupy. See also -memory.allowedBytes. Too low value may increase cache miss rate, which usually results in higher CPU and disk IO usage. Too high value may evict too much data from OS page cache, which will result in higher disk IO usage (default 60)

```
-opentsdbHTTPListenAddr string
```

TCP address to listen for OpentTSDB HTTP put requests. Usually :4242 must be set. Doesn't work if empty

```
-opentsdbListenAddr string
```

TCP and UDP address to listen for OpentTSDB metrics. Telnet put messages and HTTP /api/put messages are simultaneously served on TCP port. Usually :4242 must be set. Doesn't work if empty

```
-opentsdbTrimTimestamp duration
```

Trim timestamps for OpenTSDB 'telnet put' data to this duration. Minimum practical duration is 1s. Higher duration (i.e. 1m) may be used for reducing disk space usage for timestamp data (default 1s)

```
-opentsdbhttp.maxInsertRequestSize value
```

The maximum size of OpenTSDB HTTP put request
Supports the following optional suffixes for values: KB, MB, GB, KiB, MiB, GiB (default 33554432)

```
-opentsdbhttpTrimTimestamp duration
```

Trim timestamps for OpenTSDB HTTP data to this duration. Minimum practical duration is 1ms. Higher duration (i.e. 1s) may be used for reducing disk space usage for timestamp data (default 1ms)

```
-relabelConfig string
```

Optional path to a file with relabeling rules, which are applied to all the ingested metrics. See https://victoriametrics.github.io/#relabeling for details

```
-replicationFactor int
```

Replication factor for the ingested data, i.e. how many copies to make among distinct -storageNode instances. Note that vmselect must run with -dedup.minScrapeInterval=1ms for data de-duplication when replicationFactor is greater than 1. Higher values for -dedup.minScrapeInterval at vmselect is OK (default 1)

```
-rpc.disableCompression
```

Disable compression of RPC traffic. This reduces CPU usage at the cost of higher network bandwidth usage

```
-storageNode array
```

Address of vmstorage nodes; usage: -storageNode=vmstorage-host1:8400 -storageNode=vmstorage-host2:8400
Supports array of values separated by comma or specified via multiple flags.

```
-tls
```

Whether to enable TLS (aka HTTPS) for incoming requests. -tlsCertFile and -tlsKeyFile must be set if -tls is set

```
-tlsCertFile string
```

Path to file with TLS certificate. Used only if -tls is set. Prefer ECDSA certs instead of RSA certs, since RSA certs are slow

```
-tlsKeyFile string
```

Path to file with TLS key. Used only if -tls is set

vmSelect

```
-cacheDataPath string
```

Path to directory for cache files. Cache isn't saved if empty

```
-dedup.minScrapeInterval duration
```

Remove superflouos samples from time series if they are located closer to each other than this duration. This may be useful for reducing overhead when multiple identically configured Prometheus instances write data to the same VictoriaMetrics. Deduplication is disabled if the -dedup.minScrapeInterval is 0

```
-enableTCP6
```

Whether to enable IPv6 for listening and dialing. By default only IPv4 TCP is used

```
-envflag.enable
```

Whether to enable reading flags from environment variables additionally to command line. Command line flag values have priority over values from environment vars. Flags are read only from command line if this flag isn't set

```
-envflag.prefix string
```

Prefix for environment variables if -envflag.enable is set

```
-fs.disableMmap
```

Whether to use pread() instead of mmap() for reading data files. By default mmap() is used for 64-bit arches and pread() is used for 32-bit arches, since they cannot read data files bigger than 2^32 bytes in memory. mmap() is usually faster for reading small data chunks than pread()

```
-graphiteTrimTimestamp duration
```

Trim timestamps for Graphite data to this duration. Minimum practical duration is 1s. Higher duration (i.e. 1m) may be used for reducing disk space usage for timestamp data (default 1s)

```
-http.connTimeout duration
```

Incoming http connections are closed after the configured timeout. This may help spreading incoming load among a cluster of services behind load balancer. Note that the real timeout may be bigger by up to 10% as a protection from Thundering herd problem (default 2m0s)

```
-http.disableResponseCompression
```

Disable compression of HTTP responses for saving CPU resources. By default compression is enabled to save network bandwidth

```
-http.idleConnTimeout duration
```

Timeout for incoming idle http connections (default 1m0s)

```
-http.maxGracefulShutdownDuration duration
```

The maximum duration for graceful shutdown of HTTP server. Highly loaded server may require increased value for graceful shutdown (default 7s)

```
-http.pathPrefix string
```

An optional prefix to add to all the paths handled by http server. For example, if '-http.pathPrefix=/foo/bar' is set, then all the http requests will be handled on '/foo/bar/...' paths. This may be useful for proxied requests. See https://www.robustperception.io/using-external-urls-and-proxies-with-prometheus

```
-http.shutdownDelay duration
```

Optional delay before http server shutdown. During this dealy the servier returns non-OK responses from /health page, so load balancers can route new requests to other servers

```
-httpListenAddr string
```

Address to listen for http connections (default ":8481")

```
-loggerDisableTimestamps
```

Whether to disable writing timestamps in logs

```
-loggerErrorsPerSecondLimit int
```

Per-second limit on the number of ERROR messages. If more than the given number of errors are emitted per second, then the remaining errors are suppressed. Zero value disables the rate limit

```
-loggerFormat string
```

Format for logs. Possible values: default, json (default "default")

```
-loggerLevel string
```

Minimum level of errors to log. Possible values: INFO, WARN, ERROR, FATAL, PANIC (default "INFO")

```
-loggerOutput string
```

Output for the logs. Supported values: stderr, stdout (default "stderr")

```
-loggerWarnsPerSecondLimit int
```

Per-second limit on the number of WARN messages. If more than the given number of warns are emitted per second, then the remaining warns are suppressed. Zero value disables the rate limit

```
-memory.allowedBytes value
```

Allowed size of system memory VictoriaMetrics caches may occupy. This option overrides -memory.allowedPercent if set to non-zero value. Too low value may increase cache miss rate, which usually results in higher CPU and disk IO usage. Too high value may evict too much data from OS page cache, which will result in higher disk IO usage
Supports the following optional suffixes for values: KB, MB, GB, KiB, MiB, GiB (default 0)

```
-memory.allowedPercent float
```

Allowed percent of system memory VictoriaMetrics caches may occupy. See also -memory.allowedBytes. Too low value may increase cache miss rate, which usually results in higher CPU and disk IO usage. Too high value may evict too much data from OS page cache, which will result in higher disk IO usage (default 60)

```
-replicationFactor int
```

How many copies of every time series is available on vmstorage nodes. See -replicationFactor command-line flag for vminsert nodes (default 1)

```
-search.cacheTimestampOffset duration
```

The maximum duration since the current time for response data, which is always queried from the original raw data, without using the response cache. Increase this value if you see gaps in responses due to time synchronization issues between VictoriaMetrics and data sources (default 5m0s)

```
-search.denyPartialResponse
```

Whether to deny partial responses if a part of -storageNode instances fail to perform queries; this trades availability over consistency; see also -search.maxQueryDuration and -search.storageTimeout

```
-search.disableCache
```

Whether to disable response caching. This may be useful during data backfilling

```
-search.latencyOffset duration
```

The time when data points become visible in query results after the collection. Too small value can result in incomplete last points for query results (default 30s)

```
-search.logSlowQueryDuration duration
```

Log queries with execution time exceeding this value. Zero disables slow query logging (default 5s)

```
-search.maxConcurrentRequests int
```

The maximum number of concurrent search requests. It shouldn't be high, since a single request can saturate all the CPU cores. See also -search.maxQueueDuration (default 6)

```
-search.maxExportDuration duration
```

The maximum duration for /api/v1/export call (default 720h0m0s)

```
-search.maxLookback duration
```

Synonim to -search.lookback-delta from Prometheus. The value is dynamically detected from interval between time series datapoints if not set. It can be overridden on per-query basis via max_lookback arg. See also '-search.maxStalenessInterval' flag, which has the same meaining due to historical reasons

```
-search.maxPointsPerTimeseries int
```

The maximum points per a single timeseries returned from the search (default 30000)

```
-search.maxQueryDuration duration
```

The maximum duration for query execution; see also -search.storageTimeout (default 30s)

```
-search.maxQueryLen value
```

The maximum search query length in bytes
Supports the following optional suffixes for values: KB, MB, GB, KiB, MiB, GiB (default 16384)

```
-search.maxQueueDuration duration
```

The maximum time the request waits for execution when -search.maxConcurrentRequests limit is reached; see also -search.maxQueryDuration (default 10s)

```
-search.maxStalenessInterval duration
```

The maximum interval for staleness calculations. By default it is automatically calculated from the median interval between samples. This flag could be useful for tuning Prometheus data model closer to Influx-style data model. See https://prometheus.io/docs/prometheus/latest/querying/basics/#staleness for details. See also '-search.maxLookback' flag, which has the same meaning due to historical reasons

```
-search.minStalenessInterval duration
```

The minimum interval for staleness calculations. This flag could be useful for removing gaps on graphs generated from time series with irregular intervals between samples. See also '-search.maxStalenessInterval'

```
-search.queryStats.lastQueriesCount /api/v1/status/top_queries
```

Query stats for /api/v1/status/top_queries is tracked on this number of last queries. Zero value disables query stats tracking (default 20000)

```
-search.queryStats.minQueryDuration /api/v1/status/top_queries
```

The minimum duration for queries to track in query stats at /api/v1/status/top_queries. Queries with lower duration are ignored in query stats

```
-search.resetCacheAuthKey string
```

Optional authKey for resetting rollup cache via /internal/resetRollupResultCache call

```
-search.storageTimeout duration
```

The timeout for per-storage query processing; this allows returning partial responses if certain -storageNode instances slowly process the query; see also -search.maxQueryDuration and -search.denyPartialResponse command-line flags

```
-search.treatDotsAsIsInRegexps
```

Whether to treat dots as is in regexp label filters used in queries. For example, foo{bar=~"a.b.c"} will be automatically converted to foo{bar=~"a\.b\.c"}, i.e. all the dots in regexp filters will be automatically escaped in order to match only dot char instead of matching any char. Dots in ".+", ".*" and ".{n}" regexps aren't escaped. Such escaping can be useful when querying Graphite data

```
-selectNode array
```

Addresses of vmselect nodes; usage: -selectNode=vmselect-host1:8481 -selectNode=vmselect-host2:8481
Supports array of values separated by comma or specified via multiple flags.

```
-storageNode array
```

Addresses of vmstorage nodes; usage: -storageNode=vmstorage-host1:8401 -storageNode=vmstorage-host2:8401
Supports array of values separated by comma or specified via multiple flags.

```
-tls
```

Whether to enable TLS (aka HTTPS) for incoming requests. -tlsCertFile and -tlsKeyFile must be set if -tls is set

```
-tlsCertFile string
```

Path to file with TLS certificate. Used only if -tls is set. Prefer ECDSA certs instead of RSA certs, since RSA certs are slow

```
-tlsKeyFile string
```

Path to file with TLS key. Used only if -tls is set

vmStorage

```
-bigMergeConcurrency int
```

The maximum number of CPU cores to use for big merges. Default value is used if set to 0

```
-dedup.minScrapeInterval duration
```

Remove superflouos samples from time series if they are located closer to each other than this duration. This may be useful for reducing overhead when multiple identically configured Prometheus instances write data to the same VictoriaMetrics. Deduplication is disabled if the -dedup.minScrapeInterval is 0

```
-denyQueriesOutsideRetention
```

Whether to deny queries outside of the configured -retentionPeriod. When set, then /api/v1/query_range would return '503 Service Unavailable' error for queries with 'from' value outside -retentionPeriod. This may be useful when multiple data sources with distinct retentions are hidden behind query-tee

```
-enableTCP6
```

Whether to enable IPv6 for listening and dialing. By default only IPv4 TCP is used

```
-envflag.enable
```

Whether to enable reading flags from environment variables additionally to command line. Command line flag values have priority over values from environment vars. Flags are read only from command line if this flag isn't set

```
-envflag.prefix string
```

Prefix for environment variables if -envflag.enable is set

```
-finalMergeDelay duration
```

The delay before starting final merge for per-month partition after no new data is ingested into it. Final merge may require additional disk IO and CPU resources. Final merge may increase query speed and reduce disk space usage in some cases. Zero value disables final merge

```
-forceFlushAuthKey string
```

authKey, which must be passed in query string to /internal/force_flush pages

```
-forceMergeAuthKey string
```

authKey, which must be passed in query string to /internal/force_merge pages

```
-fs.disableMmap
```

Whether to use pread() instead of mmap() for reading data files. By default mmap() is used for 64-bit arches and pread() is used for 32-bit arches, since they cannot read data files bigger than 2^32 bytes in memory. mmap() is usually faster for reading small data chunks than pread()

```
-http.connTimeout duration
```

Incoming http connections are closed after the configured timeout. This may help spreading incoming load among a cluster of services behind load balancer. Note that the real timeout may be bigger by up to 10% as a protection from Thundering herd problem (default 2m0s)

```
-http.disableResponseCompression
```

Disable compression of HTTP responses for saving CPU resources. By default compression is enabled to save network bandwidth

```
-http.idleConnTimeout duration
```

Timeout for incoming idle http connections (default 1m0s)

```
-http.maxGracefulShutdownDuration duration
```

The maximum duration for graceful shutdown of HTTP server. Highly loaded server may require increased value for graceful shutdown (default 7s)

```
-http.pathPrefix string
```

An optional prefix to add to all the paths handled by http server. For example, if '-http.pathPrefix=/foo/bar' is set, then all the http requests will be handled on '/foo/bar/...' paths. This may be useful for proxied requests. See https://www.robustperception.io/using-external-urls-and-proxies-with-prometheus

```
-http.shutdownDelay duration
```

Optional delay before http server shutdown. During this dealy the servier returns non-OK responses from /health page, so load balancers can route new requests to other servers

```
-httpListenAddr string
```

Address to listen for http connections (default ":8482")

```
-loggerDisableTimestamps
```

Whether to disable writing timestamps in logs

```
-loggerErrorsPerSecondLimit int
```

Per-second limit on the number of ERROR messages. If more than the given number of errors are emitted per second, then the remaining errors are suppressed. Zero value disables the rate limit

```
-loggerFormat string
```

Format for logs. Possible values: default, json (default "default")

```
-loggerLevel string
```

Minimum level of errors to log. Possible values: INFO, WARN, ERROR, FATAL, PANIC (default "INFO")

```
-loggerOutput string
```

Output for the logs. Supported values: stderr, stdout (default "stderr")

```
-loggerWarnsPerSecondLimit int
```

Per-second limit on the number of WARN messages. If more than the given number of warns are emitted per second, then the remaining warns are suppressed. Zero value disables the rate limit

```
-memory.allowedBytes value
```

Allowed size of system memory VictoriaMetrics caches may occupy. This option overrides -memory.allowedPercent if set to non-zero value. Too low value may increase cache miss rate, which usually results in higher CPU and disk IO usage. Too high value may evict too much data from OS page cache, which will result in higher disk IO usage
Supports the following optional suffixes for values: KB, MB, GB, KiB, MiB, GiB (default 0)

```
-memory.allowedPercent float
```

Allowed percent of system memory VictoriaMetrics caches may occupy. See also -memory.allowedBytes. Too low value may increase cache miss rate, which usually results in higher CPU and disk IO usage. Too high value may evict too much data from OS page cache, which will result in higher disk IO usage (default 60)

```
-precisionBits int
```

The number of precision bits to store per each value. Lower precision bits improves data compression at the cost of precision loss (default 64)

```
-retentionPeriod value
```

Data with timestamps outside the retentionPeriod is automatically deleted
The following optional suffixes are supported: h (hour), d (day), w (week), y (year). If suffix isn't set, then the duration is counted in months (default 1)

```
-rpc.disableCompression
```

Disable compression of RPC traffic. This reduces CPU usage at the cost of higher network bandwidth usage

```
-search.maxTagKeys int
```

The maximum number of tag keys returned per search (default 100000)

```
-search.maxTagValueSuffixesPerSearch int
```

The maximum number of tag value suffixes returned from /metrics/find (default 100000)

```
-search.maxTagValues int
```

The maximum number of tag values returned per search (default 100000)

```
-search.maxUniqueTimeseries int
```

The maximum number of unique time series each search can scan (default 300000)

```
-smallMergeConcurrency int
```

The maximum number of CPU cores to use for small merges. Default value is used if set to 0

```
-snapshotAuthKey string
```

authKey, which must be passed in query string to /snapshot* pages

```
-storageDataPath string
```

Path to storage data (default "vmstorage-data")

```
-tls
```

Whether to enable TLS (aka HTTPS) for incoming requests. -tlsCertFile and -tlsKeyFile must be set if -tls is set

```
-tlsCertFile string
```

Path to file with TLS certificate. Used only if -tls is set. Prefer ECDSA certs instead of RSA certs, since RSA certs are slow

```
-tlsKeyFile string
```

Path to file with TLS key. Used only if -tls is set

```
-vminsertAddr string
```

TCP address to accept connections from vminsert services (default ":8400")

```
-vmselectAddr string
```

TCP address to accept connections from vmselect services (default ":8401")

Grafana

Grafana - это опенсорс-решение для визуализации метрик, плюс опционально - для генерации и рассылки алертов.

Установка в Kubernetes

Ставить лучше всего через Helm.

Сначала добавить дистрибутив:

```
helm repo add grafana https://grafana.github.io/helm-charts
```

Затем установка (в примере- с кастомным файлом для values):

```
helm install grafana grafana/grafana --namespace=monitoring --values grafana-values-1.yaml
```

Что интересного в файле `values.yaml`:

`ingress: annotations:` - в этой секции надо прописать настройки для basic-auth.

`adminUser` + `adminPassword` - креды для админа (если не закрывать аутентификацию секретом)

`grafana.ini` - настройки самого приложения - где хранятся файлы, дашборды, параметры авторизации и т.д.

!ВАЖНО параметр `root_url` нужно обязательно выставить, иначе инвайты новым пользователям будут приходить со ссылками типа http://localhost

В лабе есть 5 разных файлов с именами `grafana-values-{1..5}.yaml`.

Установка с помощью первого файла:

```
helm install --wait --atomic grafana grafana/grafana --values grafana-values-1.yaml -n monitoring
```

Grafana - веб-интерфейс, который отображает всевозможные циклические метрики. Давайте же приступим к установке этой программы.

Заходим на master-1 (XXXXXX - номер вашего логина).

```
ssh master-1.sXXXXXX
```

Повышаем права до root.

```
sudo -i
```

Запустим утилиту watch для постоянного отслеживания результата следующих команд.

```
watch kubectl get po,ing,secrets,svc -n monitoring
```

У нас уже стоит prometheus. Он находится по такому адресу

```
...
NAME                                    CLASS    HOSTS
ADDRESS   PORTS   AGE
ingress.extensions/prometheus-server   <none>   prometheus.sXXXXXX.edu.slurm.io
          80      25m
...
```

Ещё есть готовый секрет для basic-авторизации.

```
...
NAME                                                TYPE
       DATA   AGE
secret/basic-auth                                   Opaque
       1      26m
...
```

Запустим ещё одно окно терминала, т.к. предыдущее уже занято утилитой watch.

Выводим helm список.

```
helm repo list

NAME                    URL
stable                  https://charts.helm.sh/stable
prometheus-community    https://prometheus-community.github.io/helm-charts
grafana                 https://grafana.github.io/helm-charts
```

Перейдем в папку srv и там будем работать.

```
cd /srv
```

Клонируем репозиторий нашего курса. Если у вас ещё его нет. Соответственно вводим логин и пароль, который находится в шапке курса тут.

```
git clone https://gitlab.slurm.io/edu/lmk8s.git
```

Перейдем в репозиторий с которым будем работать на текущем уроке.

```
cd lmk8s/5.grafana/
```

Правим файл.

```
vim grafana-values-1.yaml
```

Меняем на ваш номер из логина студента в двух местах.

```
...
  hosts:
    - grafana.s<ваш номер логина>.edu.slurm.io
...
  server:
    root_url: http://grafana.s<ваш номер логина>.edu.slurm.io
...
```

Сохраняем и выходим из файла.

Запустим установку grafana.

```
helm install --wait --atomic grafana grafana/grafana --values grafana-values-
1.yaml -n monitoring
```

Переходим в первый терминал и смотрим результат работы. У нас должен запуститься под grafana...

```
...
NAME                                                READY   STATUS    RESTARTS
AGE
pod/grafana-56dd55f874-nlmwf                        1/1     Running   0
68s
...
```

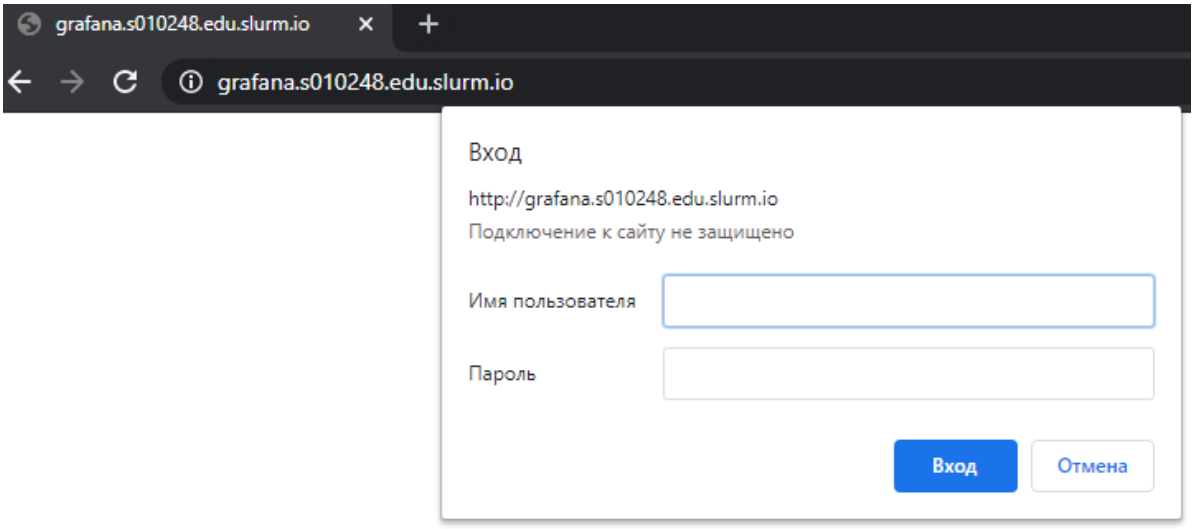
и под ingress с grafana.

```
...
NAME                                  CLASS    HOSTS
ADDRESS   PORTS   AGE
ingress.extensions/grafana            <none>   grafana.sXXXXXX.edu.slurm.io
          80      110s
...
```

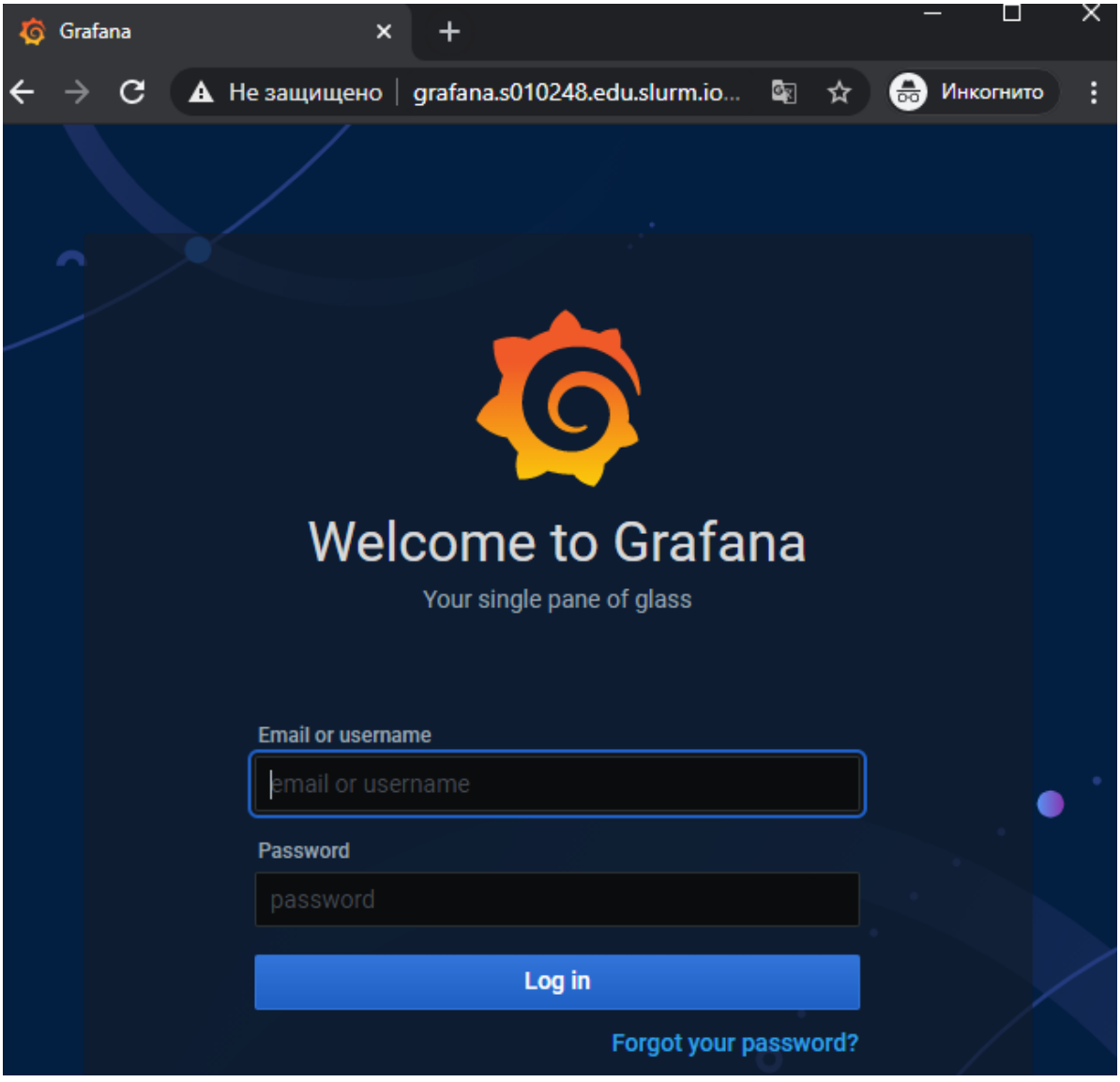
Копируем HOSTS от ingress с grafana и переходим в браузере по этой ссылке.

> Открывайте приложение в режиме "инкогнито", т.к. в обычном режиме браузер блокирует доступ по http. В Google Chrome к примеру, можно запустить режим с помощью комбинации "Ctrl+Shift+N".

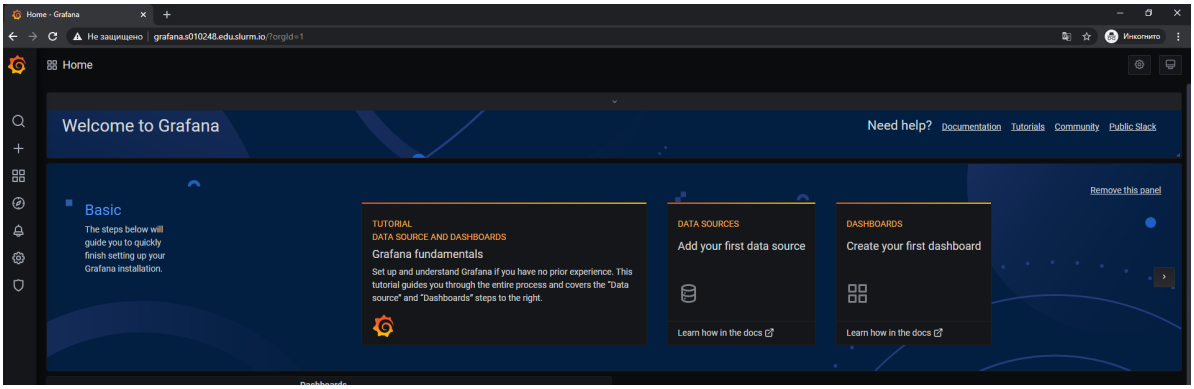
basic авторизация. Вводите ваш логин студента личного кабинета и от него пароль.



grafana авторизация. Тут вводим admin и пароль, который задавали ранее в файле grafana-values-1.yaml



И нас приветствует Grafana. Мы это сделали.



Источники данных (data source)

По сути, Grafana - это фронтенд любого стека мониторинга.

Она не обрабатывает и не хранит метрики.

Вместо этого - она забирает их с бекендов - это могут быть БД, системы для хранения логов, системы мониторинга (в т.ч. Прометей).

Data Source - это коннектор между инстансом метрик и Графаной.

Атрибуты data source:

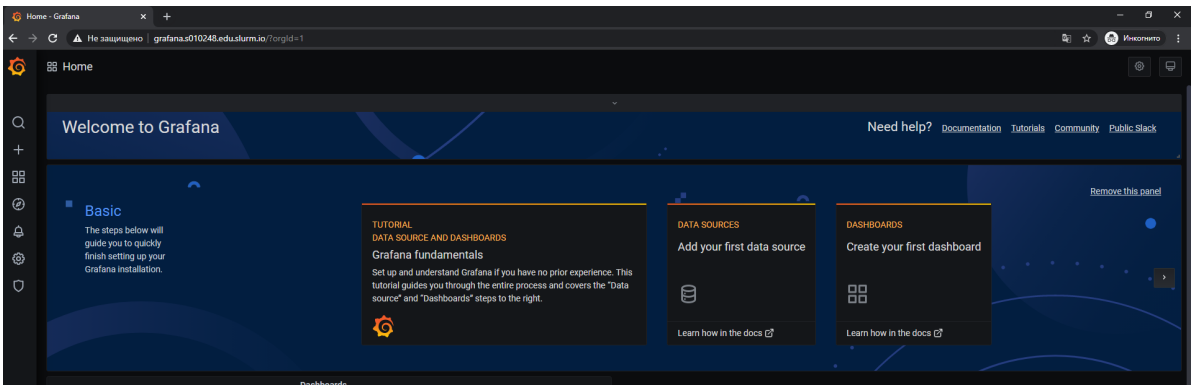
- имя (по умолчанию `default`)
- тип (Prometheus, SQL, Clickhouse, Influx, Jaeger/Zipkin etc)
- кастомные настройки (параметры подключения, логины/пароли итд)

Большинство data source на практике - это HTTP-источники, куда Графана будет обращаться по REST API.

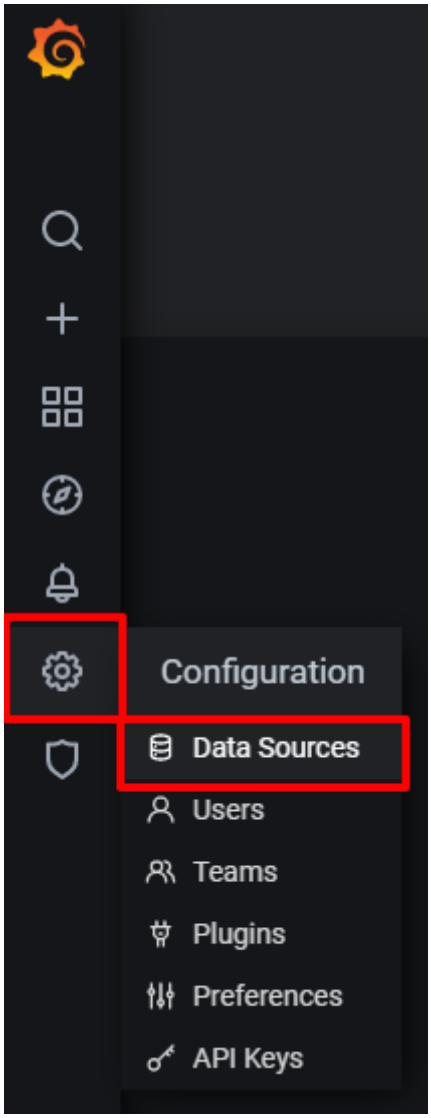
Если Графана и Прометей в одном кластере, то URL до Прометея будет вида `http://<имя сервиса>.<имя неймспейса>.svc.cluster.local`

В разделе Explore можно проверить работу data source, руками введя PromQL-запрос в метриках.

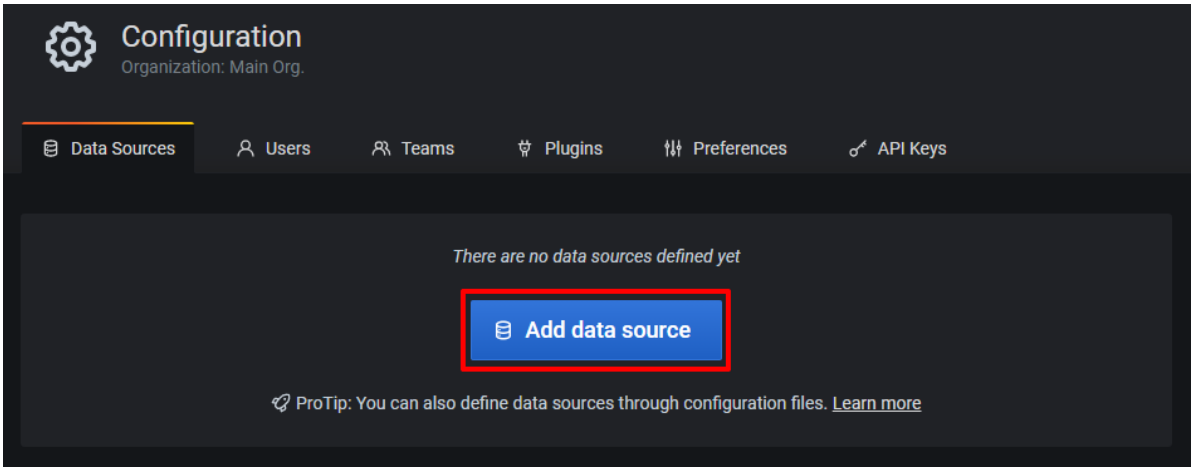
Заходим в нашу grafana.



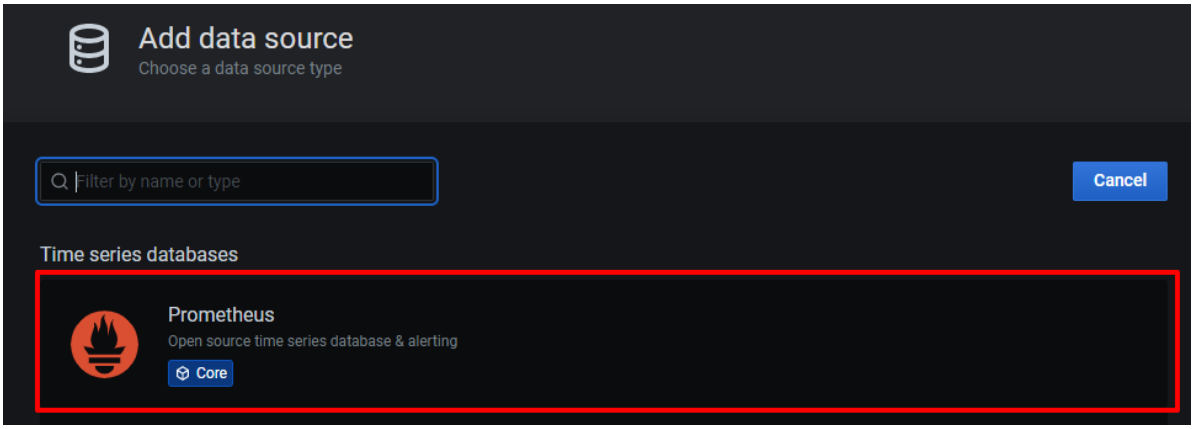
Давайте же создадим наш первый Data Source. **Menu** > **Configuration** > **Data Sources**



Пока у нас нет ни одного Data Source. Нажимаем **Add data source.**



Выбираем тип - **Prometheus**



Посмотрим какие есть сервисы у нас. Смотрим вывод команды...

```
watch kubectl get po,ing,secrets,svc -n monitoring
```

Видим svc - service/prometheus-server.

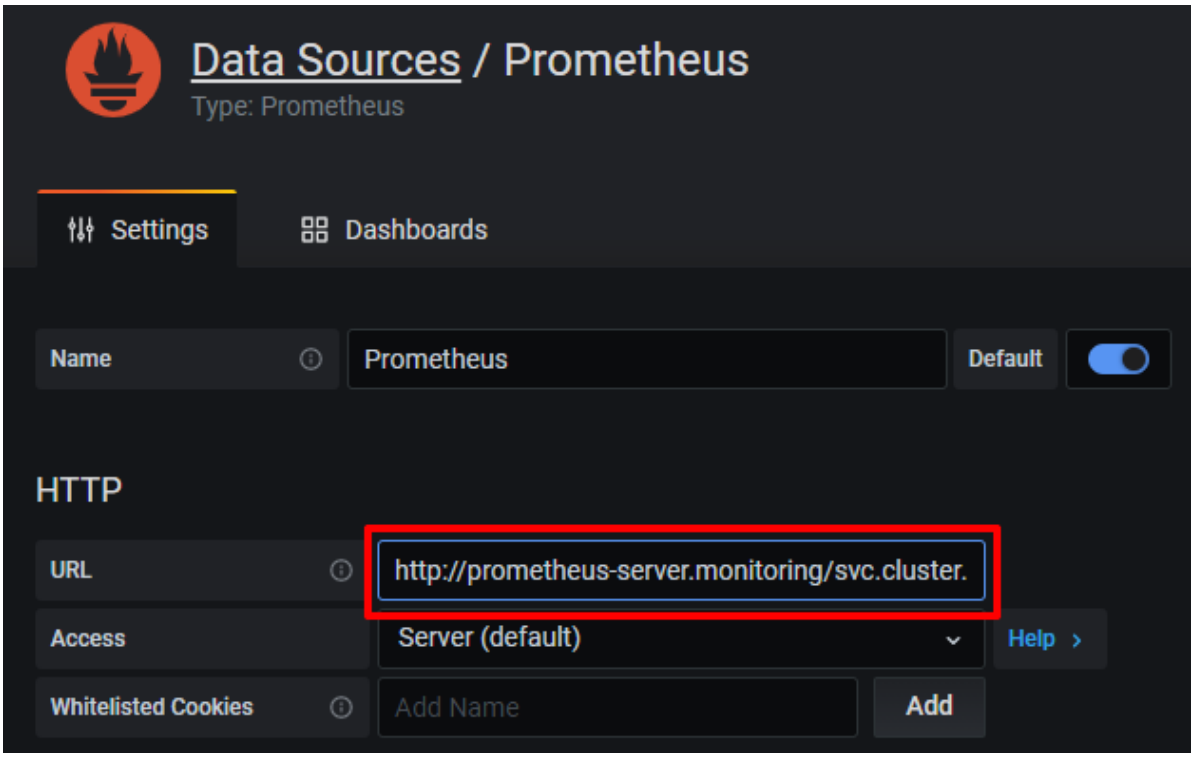
```
...
NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP
PORT(S)    AGE
...
service/prometheus-server           ClusterIP   10.105.50.222   <none>
80/TCP     117m
```

Заполняем поле **HTTP** > **URL.** Вводим полный путь - http://prometheus-server.monitoring.svc.cluster.local

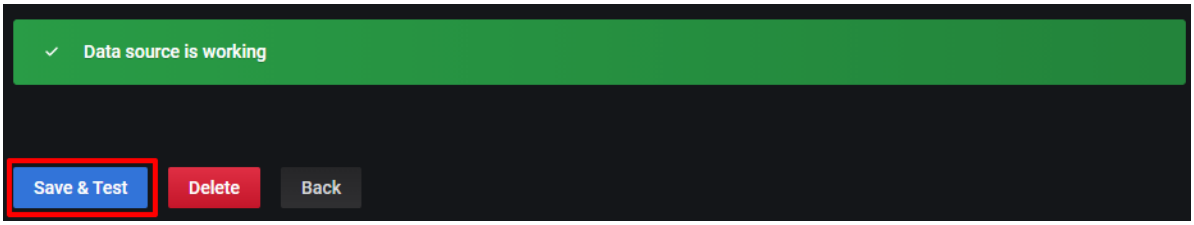
prometheus-server - наименование сервиса

monitoring - namespace

svc.cluster.local - стандартная аннотация

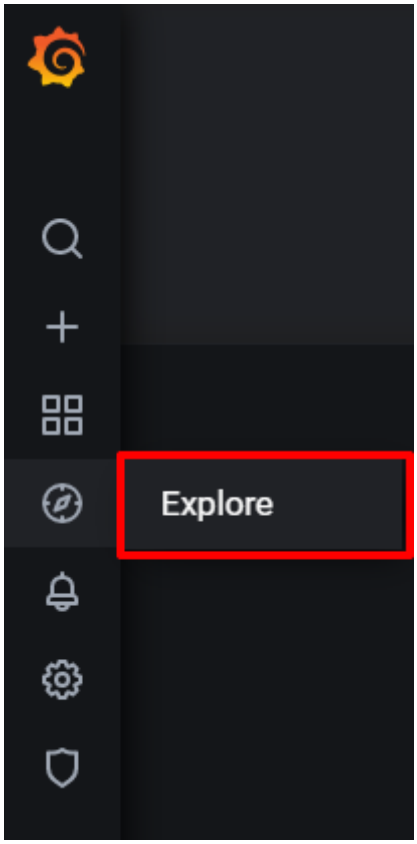


Нажимаем **Save & Test**. Создается Data source и сразу проверяется connection endpoint.

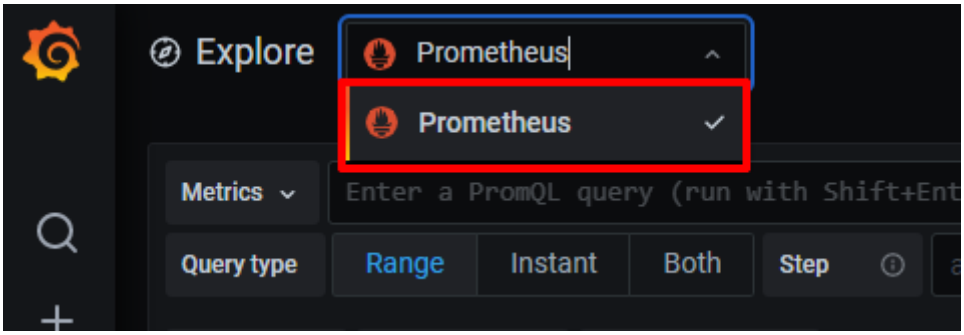


> Домашнее задание: попробуйте настроить Data source типа Prometheus через basic авторизацию.

Перейдем в раздел **Explore.**



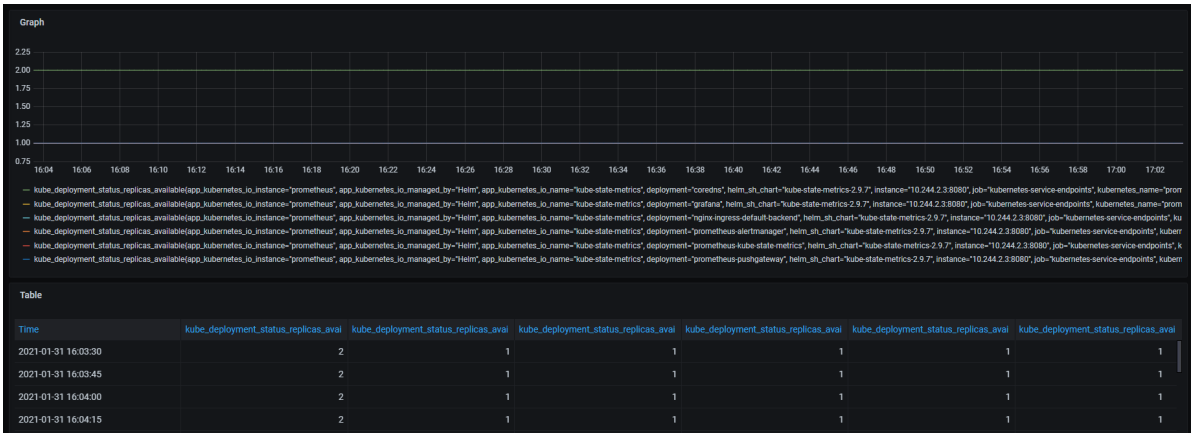
На текущий момент у нас только одно исследование. Выбираем - **Prometheus**.



Проверим доступные реплики деплойментов.

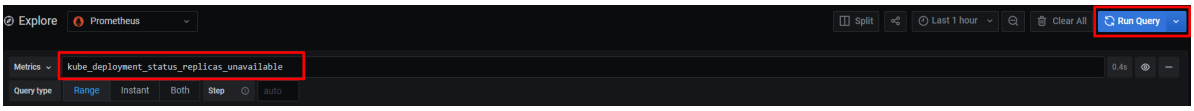
Вводим в поле **Metrics - kube_deployment_status_replicas_available** и нажимаем **Run Query**

Можете изучить вывод.



Проверим недоступные реплики деплоиментов.

Вводим в поле **Metrics - kube_deployment_status_replicas_unavailable** и нажимаем **Run Query**



Видим, что всё хорошо, ноль недоступных.

Dashboards. Готовые и собственные

Атомарная единица любого дашборда - это панель (**Panel**).

Свойства панели:

- queries - набор запросов
- visualization - тип визуализации (график, диаграмма, текст, таблица итд)
- params (параметры для кастомизации панели - threshold, mapping)

Темплейты (templating) - ввод переменных (сортировка, отброс ненужных метрик, наследование метрик друг от друга, фильтрация)

Любой дашборд - это **JSON-объект**.

Есть куча готовых дашбордов в **Grafana Store**: http://dashboards.grafana.com/dashboards

Плагины

Плагины бывают 3 типов:

- Panel - дополнительная визуализация
- Data Source - источник данных для конкретной системы
- App - дополнительные страницы или разделы в Графане

Плагины бывают FrontEnd и Backend.

Самый популярный - плагин для снятия скриншотов.

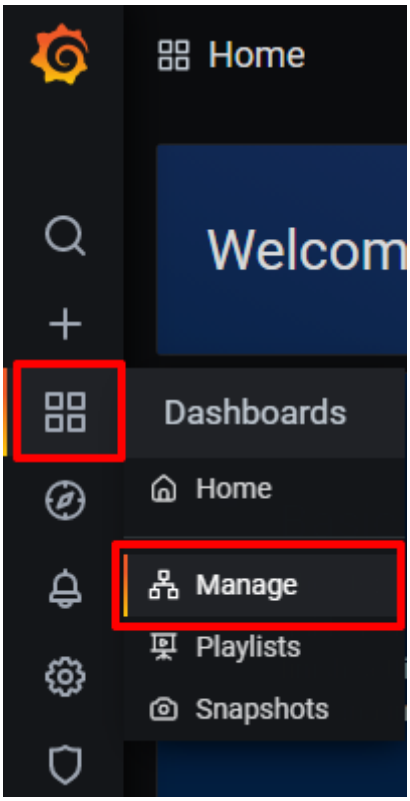
Бывают еще плагины с кастомными дашбордами, которых нет в Grafana Store.

Например, плагины для Zabbix, Percona, Kubernetes.

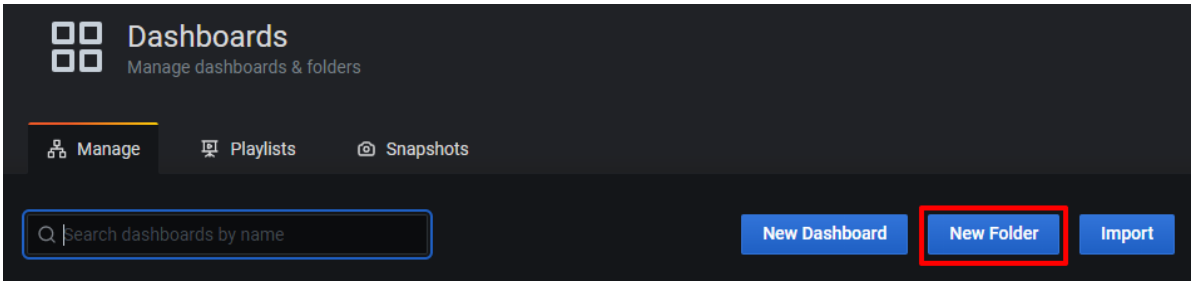
Практика

Заходим в нашу grafana. Давайте же нарисуем наш с вами первый дашборд.

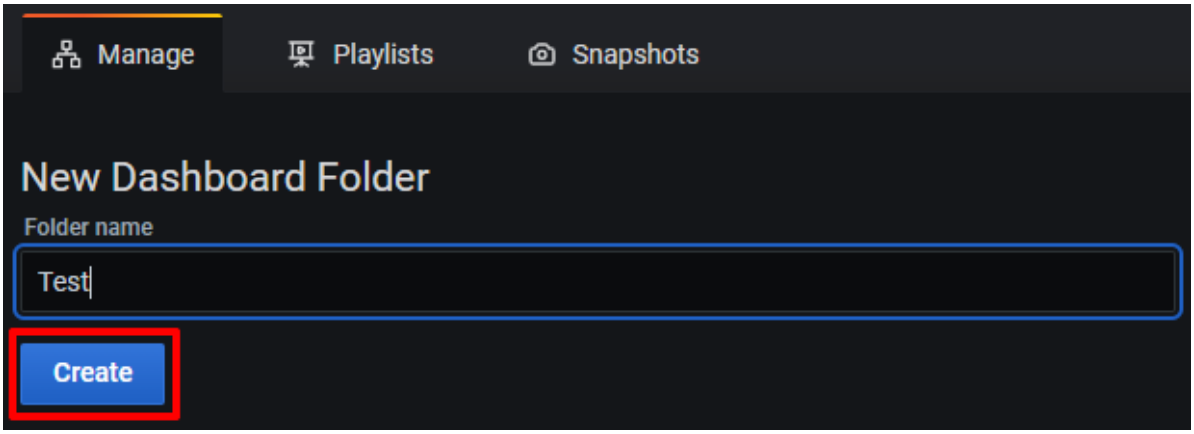
Переходим в **Dashboards** > **Manage**



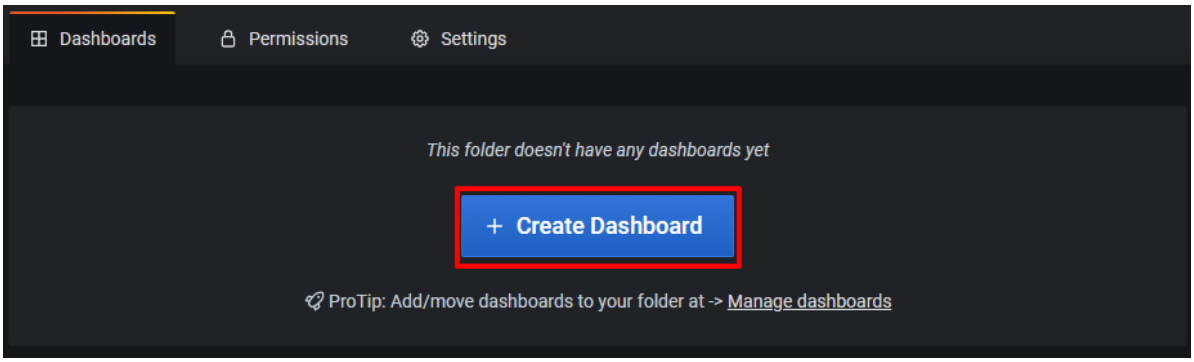
Создаем папку, где будет храниться наш дашборд. Нажимаем **New Folder**.



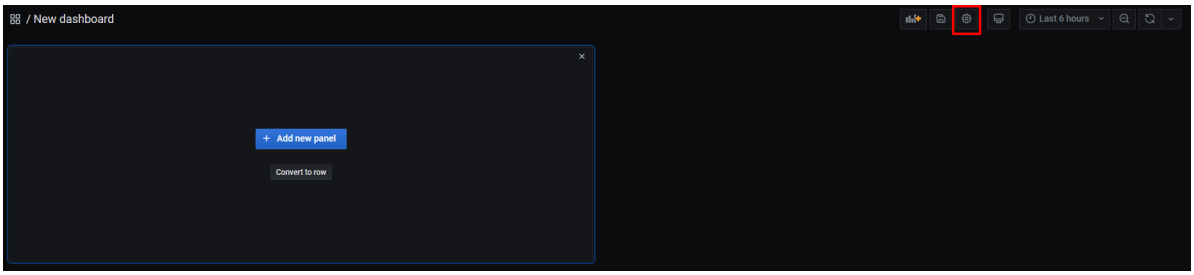
Задаем имя папки и нажимаем **Create**.



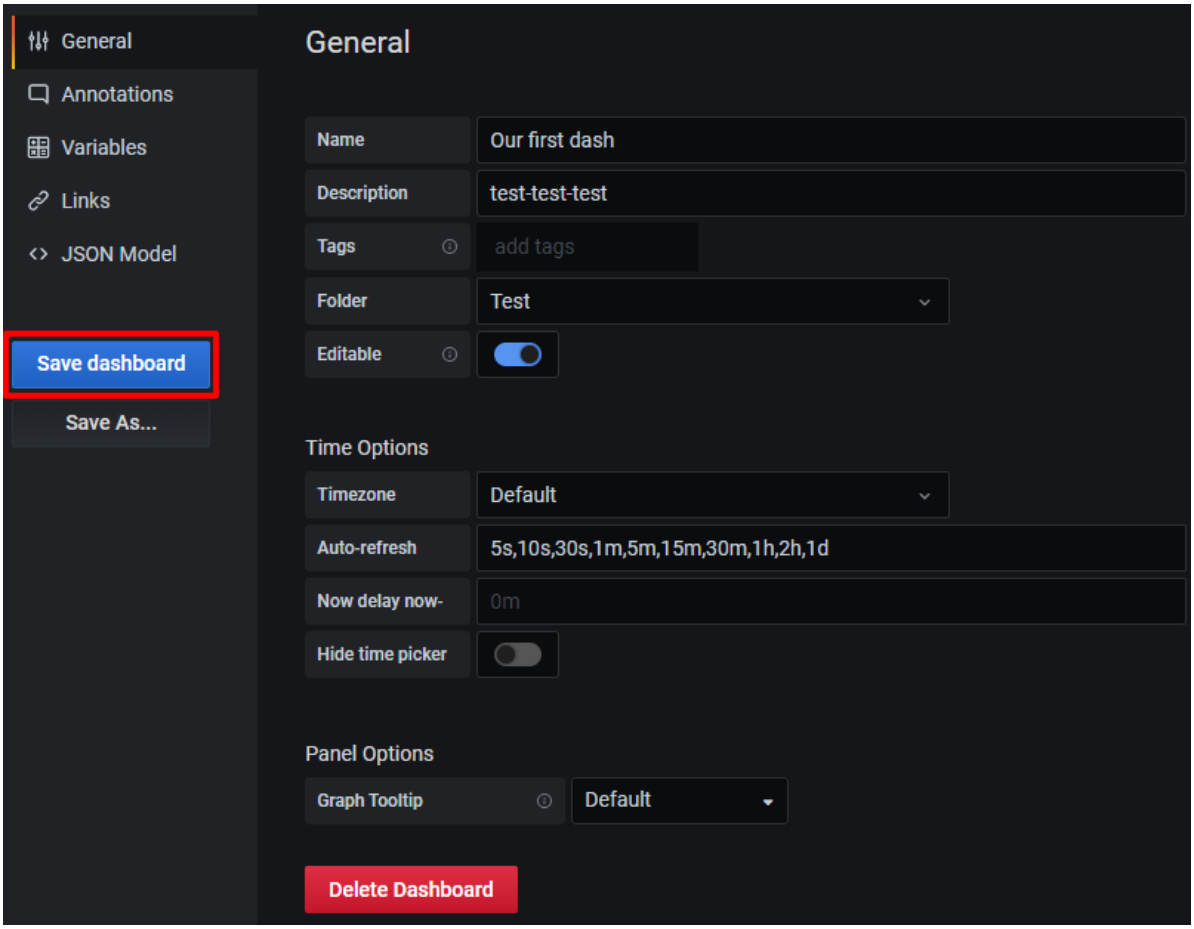
Создаем дашборд нажимаем **+ Create Dashboard**.



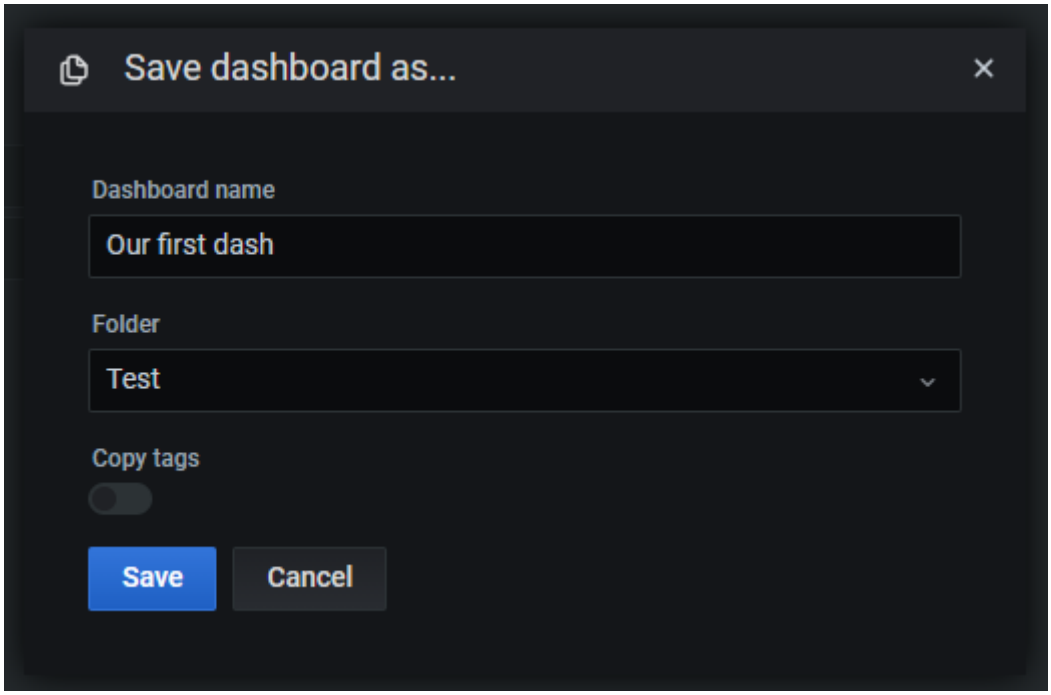
Прежде чем добавим панель, сначала давайте произведём настройку дашборда. Нажимаем на иконку "**механизм**".



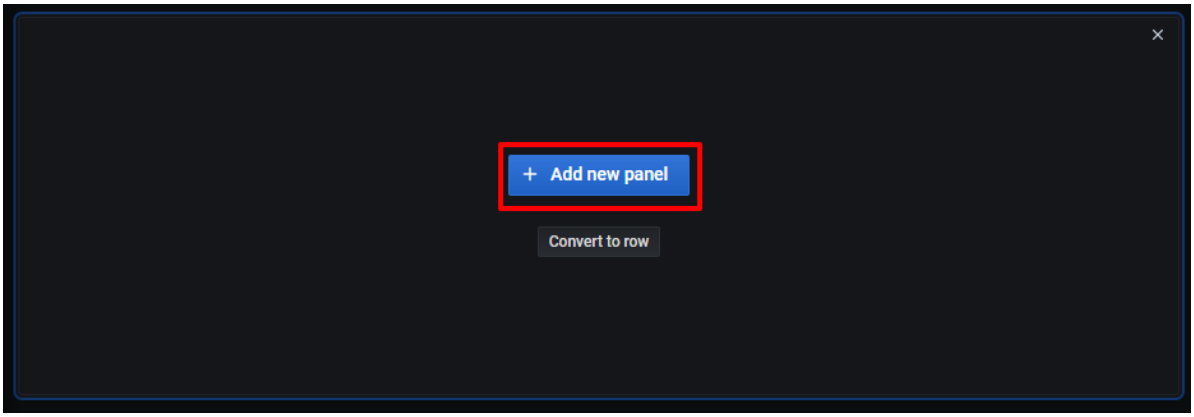
Заполним **General**. **Name**: Our first dash, **Description**: test-test-test, и нажимаем **Save dashboard**.



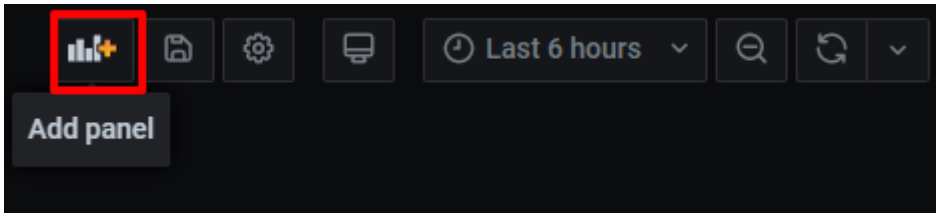
Корректируем имя дашборда и нажимаем **Save**.



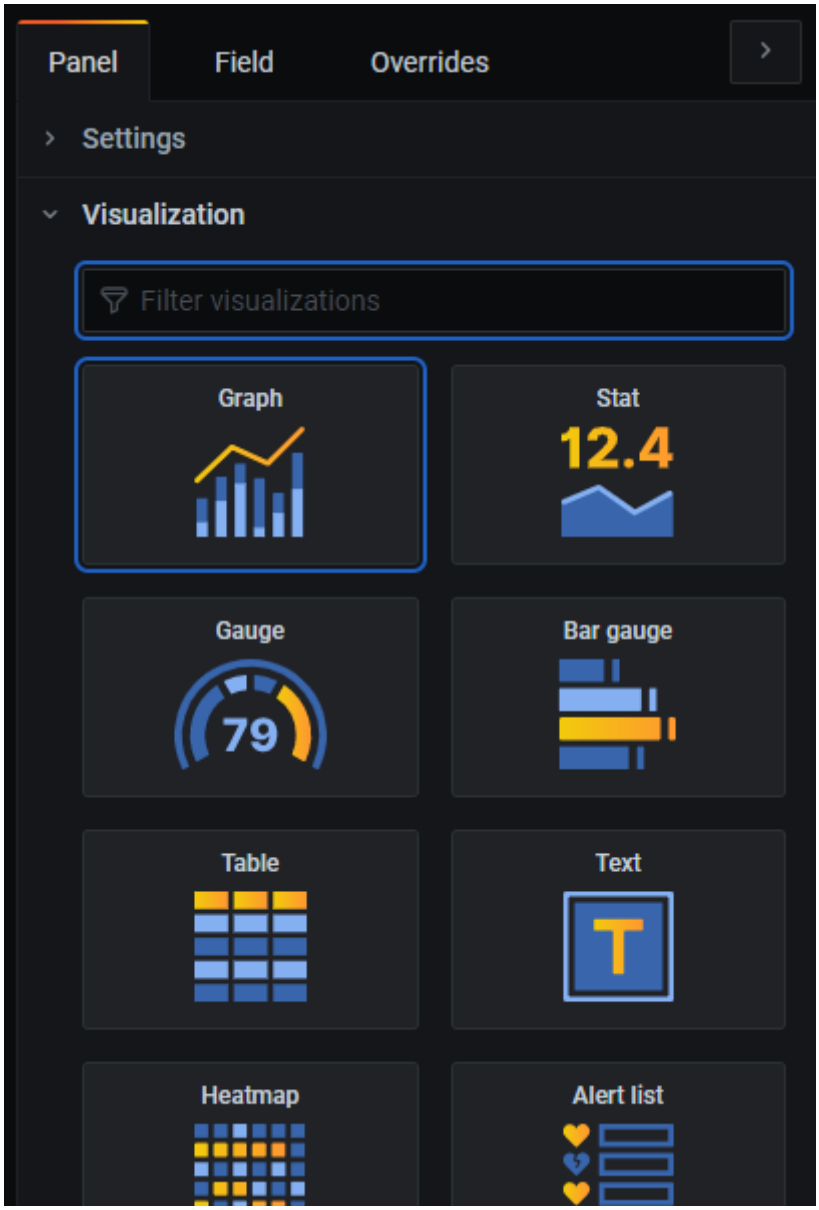
Наконец-то добрались до места, где можем приступить к созданию нашей первой панели. Нажимаем **+ Add new panel.**



Если вдруг не будет **+ Add new panel**, то нажмите на эту иконку.



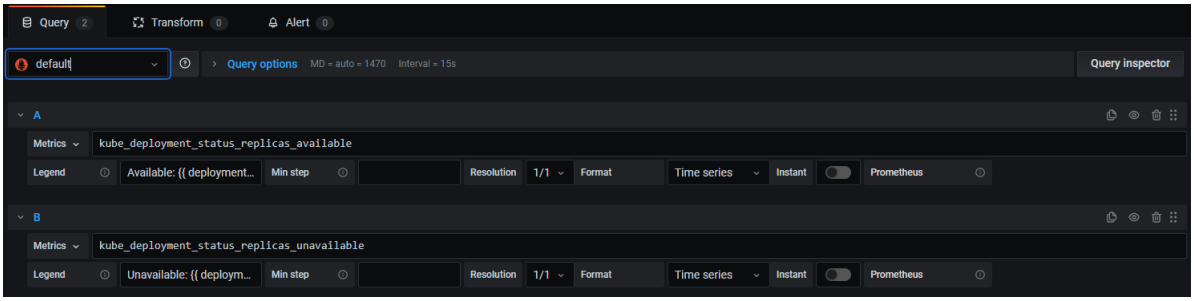
Справа на экране можно увидеть разного рода настройки отображения.



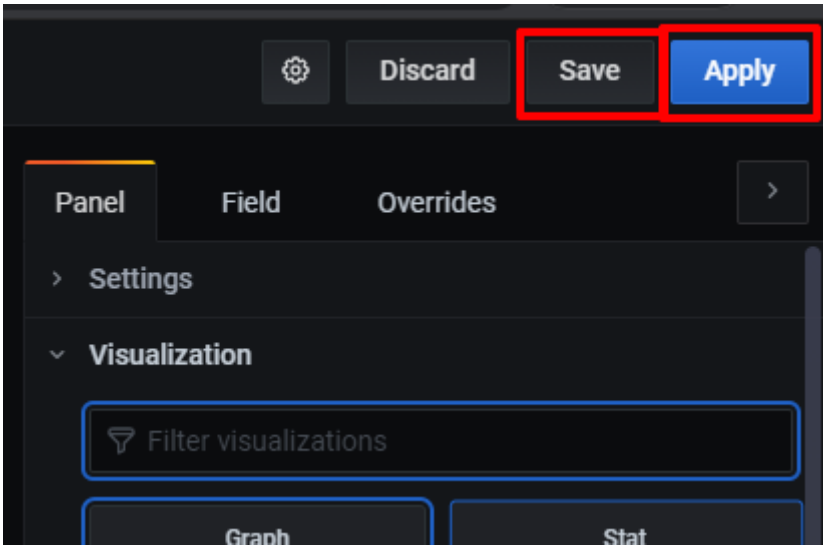
Задаем метрики и легенды.

A: Metrics: **kube_deployment_status_replicas_available** Legend: **Available: {{ deployment }}**

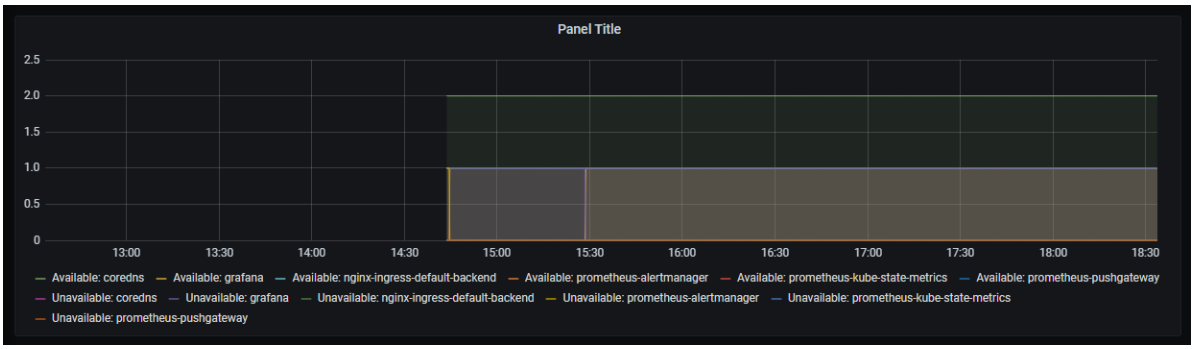
B: Metrics: **kube_deployment_status_replicas_unavailable** Legend: **Unavailable: {{ deployment }}**



Сохраняем **Save** и применяем **Apply.**



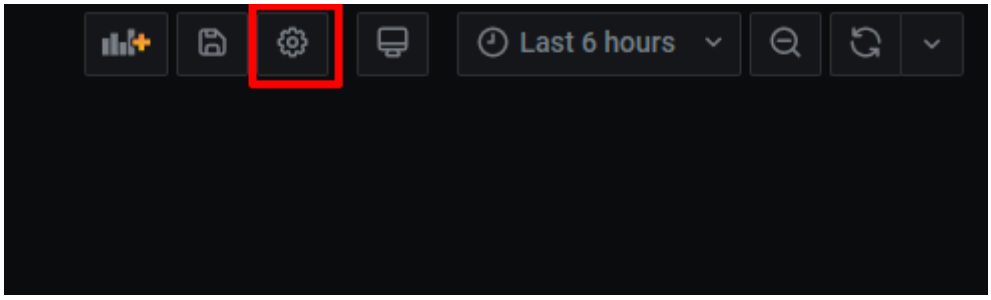
Смотрим какая красота у нас с вами получилась.



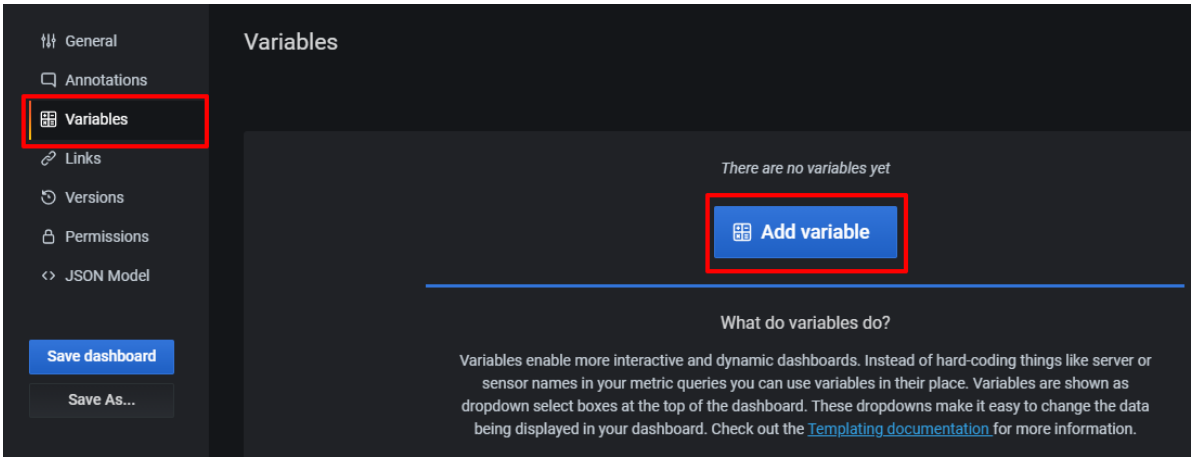
Можете поиграться с настройками отображения легенд, а также создать ещё один дашборд только с одной метрикой. Измените её отображение, чтобы она отличалась от предыдущего дашборда, который вы сделали недавно.

Теперь приступим к настройке нашей первой фильтрации. Создадим переменную, по которой можно будет фильтровать метрики по namespace.

Заходим в настройки дашборда.

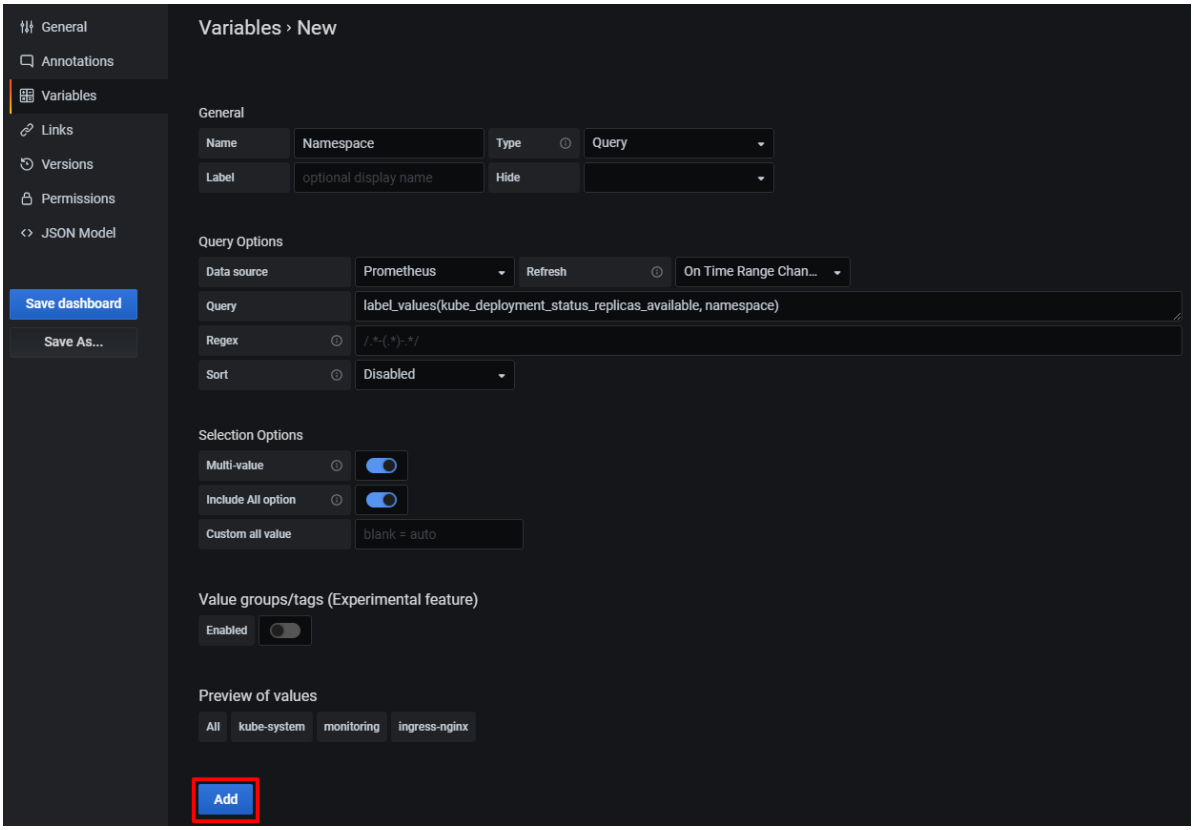


Заходим в **Variables** и нажимаем **Add variable**.

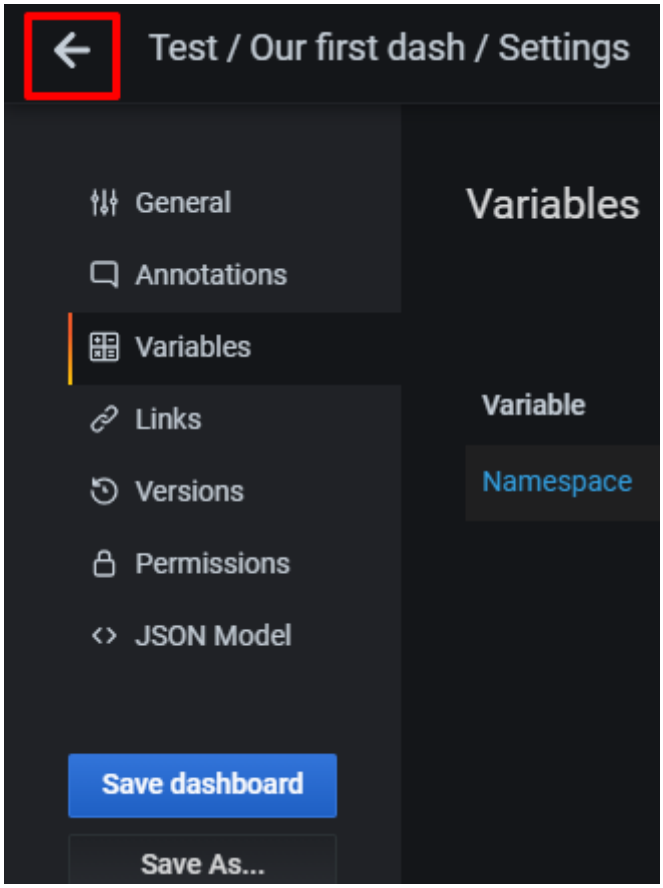


Настройте у себя так, как на скриншоте и нажмите **Add**.

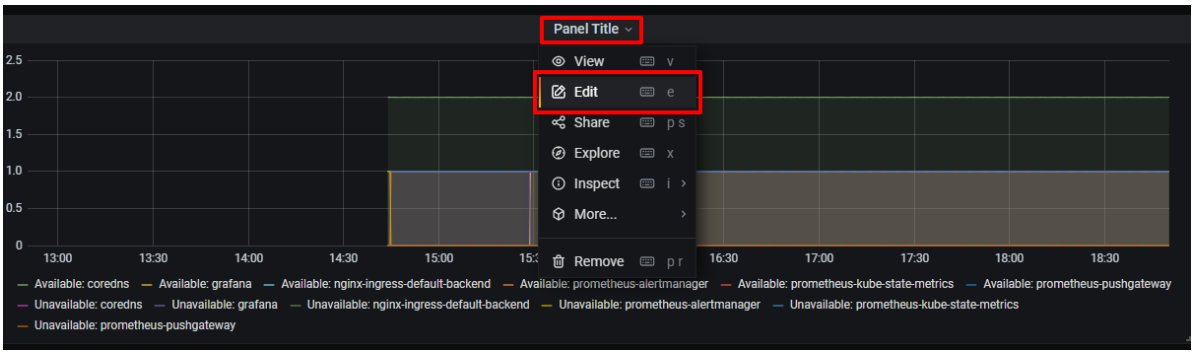
Query: **label_values(kube_deployment_status_replicas_available, namespace)**



Вернемся назад.

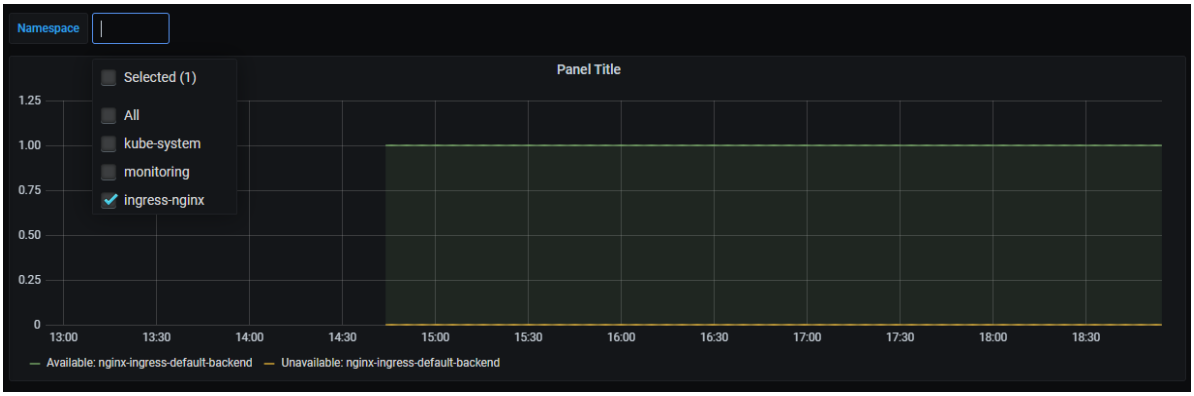


Добавим эту фильтрацию в настройки метрик панели. Заходим в **Panel** > **Edit**.



Добавим в конец Metrics это - **{namespace=~"$Namespace"}**

Нажимаем **Save** потом **Apply** и проверяем.

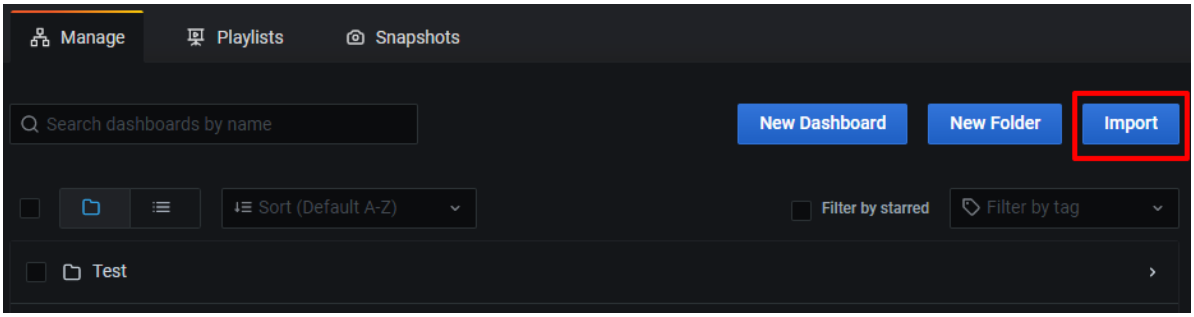


А теперь сами добавьте в конец строки Legend (через запятую) - **ns: {{namespace}}** . Теперь у нас задается namespace для каждой метрики.

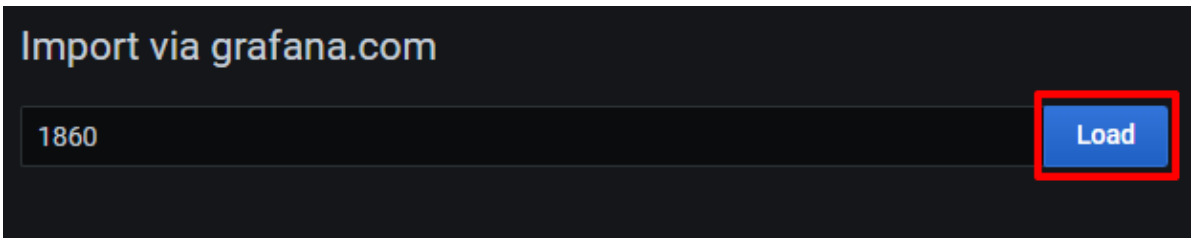
Поставим самый популярный дашборд.

Можете перейти на сайт - https://grafana.com/grafana/dashboards и изучить дашборды.

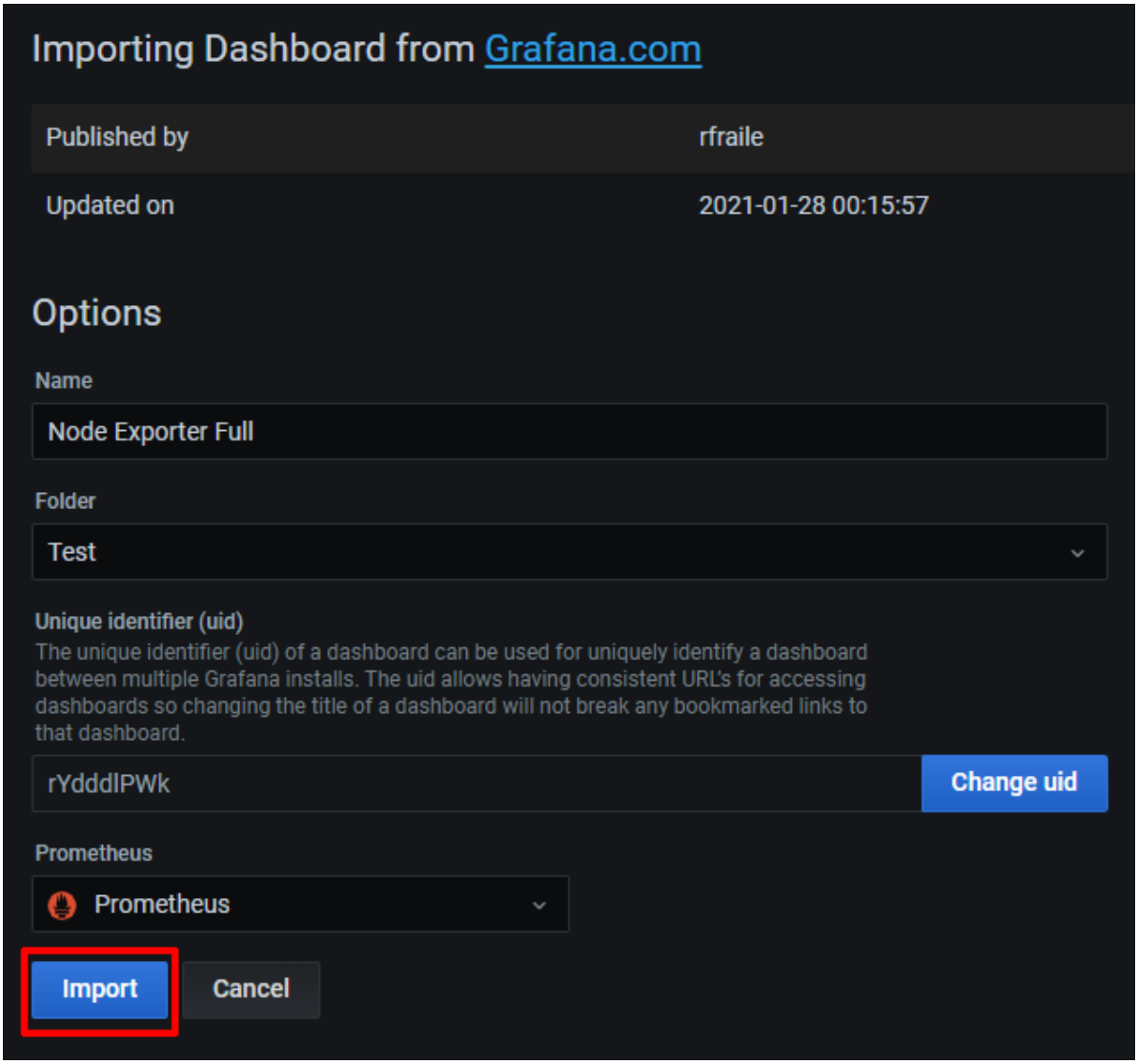
Переходим в **Dashboards** > **Manage** и нажимаем **Import**



Вводим ID дашборда - **1860** и нажимаем **Load**.



Указываем настройки какие хотите, нажимаем **Import** и ...



...любуемся.

А теперь перейдём к плагинам.

Plugins

Можете перейти на сайт и просмотреть какие есть плагины - https://grafana.com/grafana/plugins. У каждого плагина есть инструкция по установке. Мы будем устанавливать плагин - Pie Chart. Давайте же приступим.

Посмотрим вывод команды

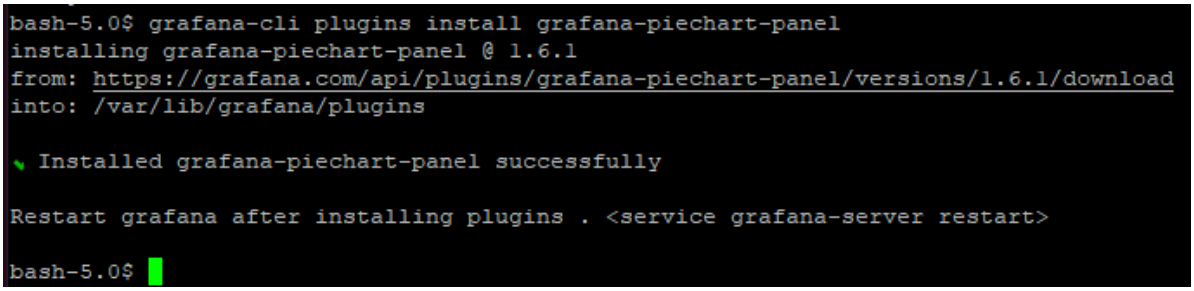
```
watch kubectl get po,ing,secrets,svc -n monitoring
```

Возьмем имя пода с grafana'ой, у меня это - **pod/grafana-56dd55f874-nlmwf** и зайдем в этот под.

```
k exec -ti pod/grafana-56dd55f874-nlmwf bash -n monitoring
```

Вводим команду на установку плагина - Pie Chart

```
grafana-cli plugins install grafana-piechart-panel
```

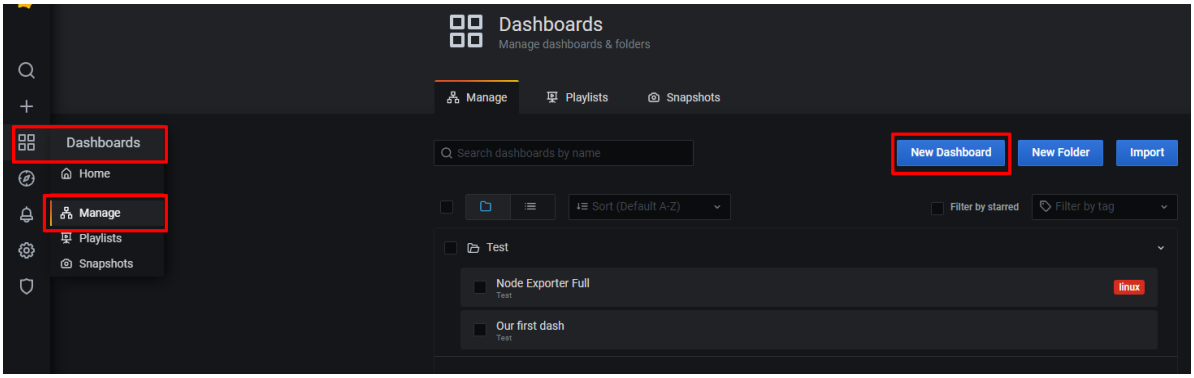


Выйдем из пода.

Удалим под, чтобы новый под уже был с нашим плагином.

```
k delete pod/grafana-56dd55f874-nlmwf -n monitoring
```

Добавим новый дашборд и панель. **Dashboards** > **Manage** > **New Dashboard > New panel**



Справа во вкладке - **Visualization** будет **Pie Chart**.

Выбираем его. В Metrics вставляем - **kube_deployment_status_replicas_available** и готово.

Я надеюсь вам было понятно. Можете ниже написать ваши впечатления от пройденного на текущий момент материала.

Организация пользователей и доступов (org, teams, LDAP)

Базово в Графане есть 3 вида ролей для пользователей:

- viewer - право только смотреть
- editor - редактировать и создавать дашборды и дата сорсы

- admin - право на всё

Можно группировать пользователей в **teams** (команды) и давать права на команду.

Если закрыть пользователю в составе команды право на дашборд или что-то еще **на уровне команды** - то если у него права были на уровне пользователя, они автоматически пропадут - запрет имеет приоритет выше.

Организация (organization) - самая верхняя иерархия в Графане, один пользователь может быть в нескольких организациях.

Из коробки поддерживаются **OAuth**-сервисы. Например, авторизация через GitHub.

Есть возможность сходить в LDAP.

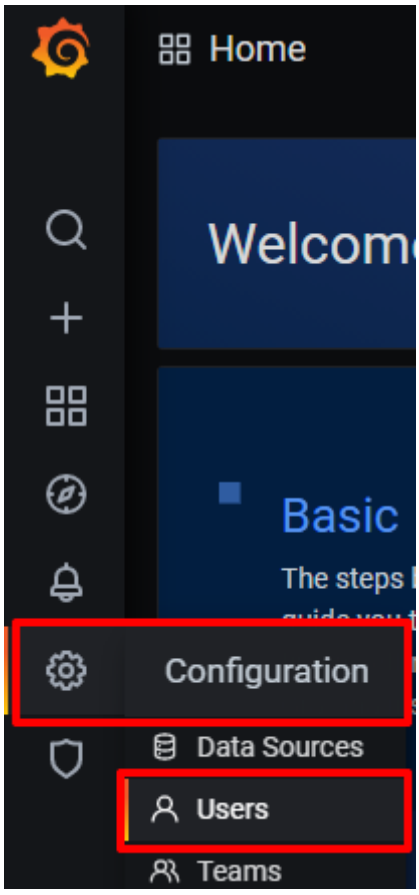
Для этого надо сходить в `grafana.ini` и раскомментировать соответствующие разделы.

Практика

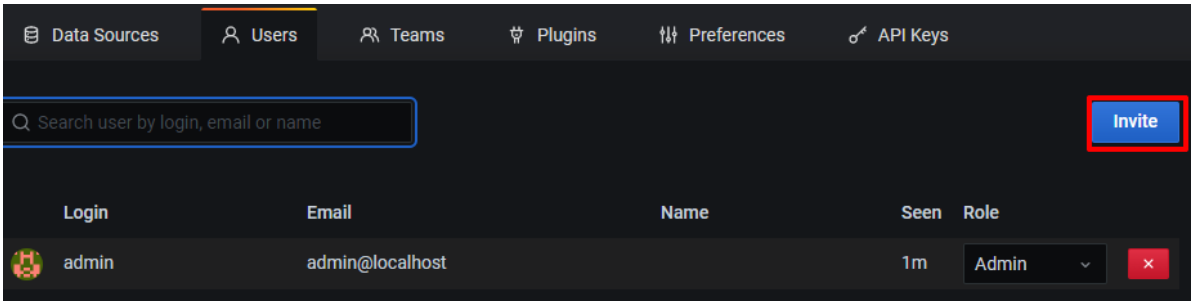
Users

Давайте приступим к созданию пользователя и дадим права ему.

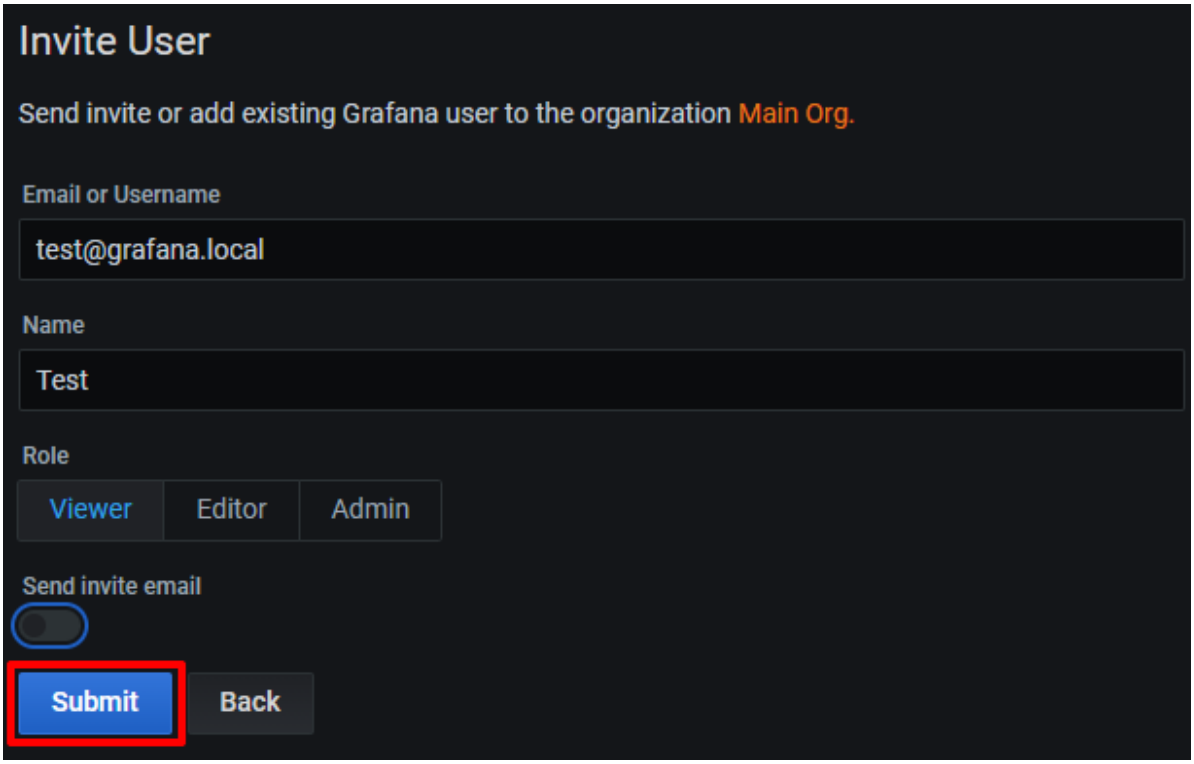
Перейдем в **Configuration** > **Users**



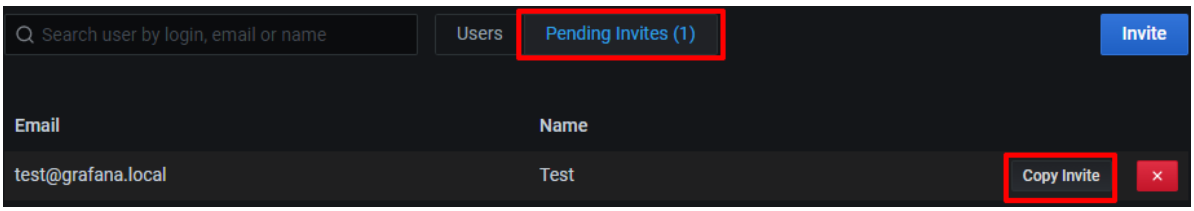
Создадим 2-ого пользователя после нашего админа. Нажимаем **Invite**.



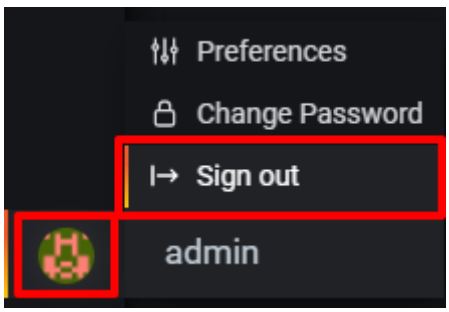
Вводим данные и нажимаем **Submit**.



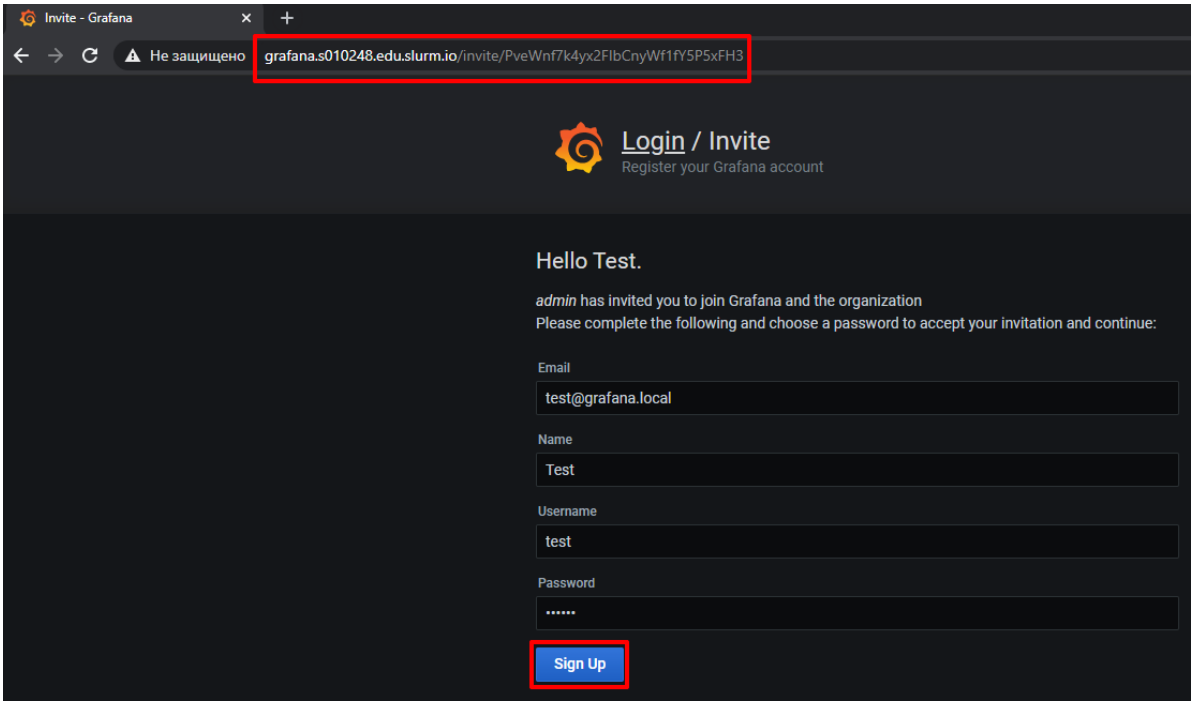
Переходим в **Pending Invites** и копируем ссылку **Copy Invite**.



Выходим из grafana.



Вставляем скопированную ссылку в адресную строку браузера. Вводим данные, задаём любой пароль и нажимаем **Sign Up**.

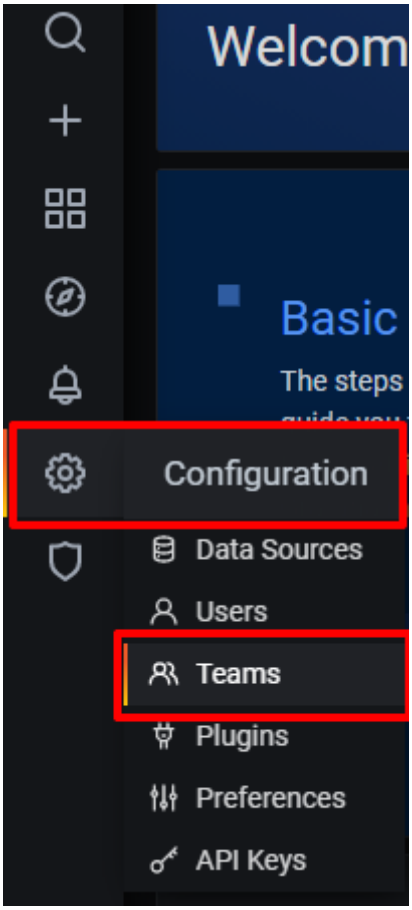


К примеру, заходим на любой наш дашборд и смотрим, что у нас права только на просмотр. **Dashboards** > **Manage** > папка **Test** > название дашборда **Our first dush** (или другой, если вы выполняли доп. задание) > **Panel title** и смотрим что нет строки **Edit**.

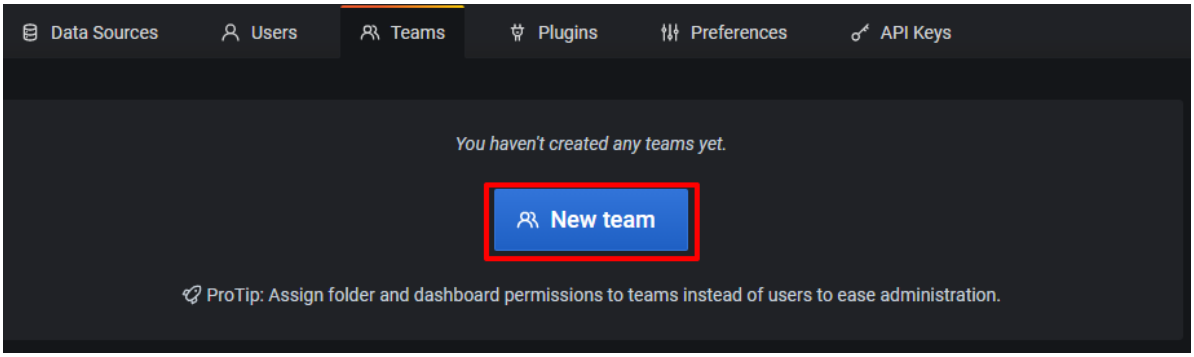
Возвращаемся в grafana под админом.

Team

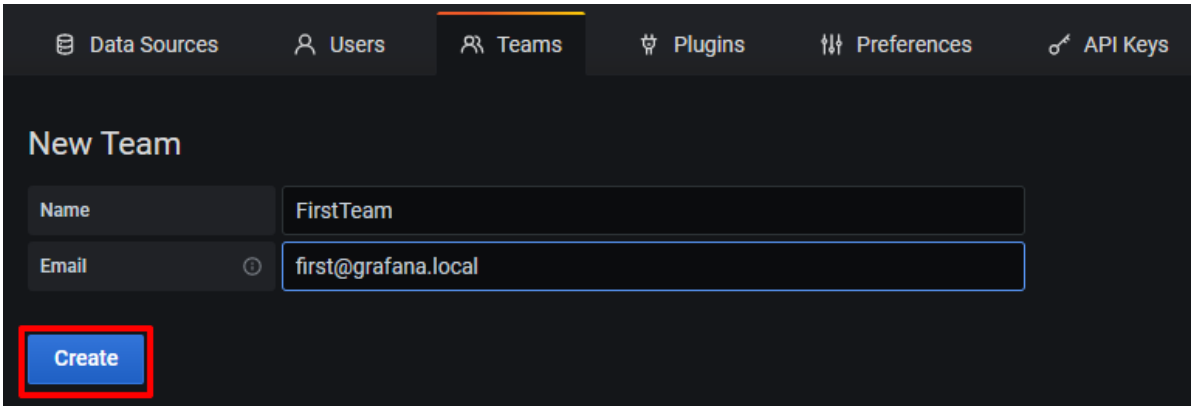
Теперь давайте создадим первую Team.



Нажимаем **New team**.

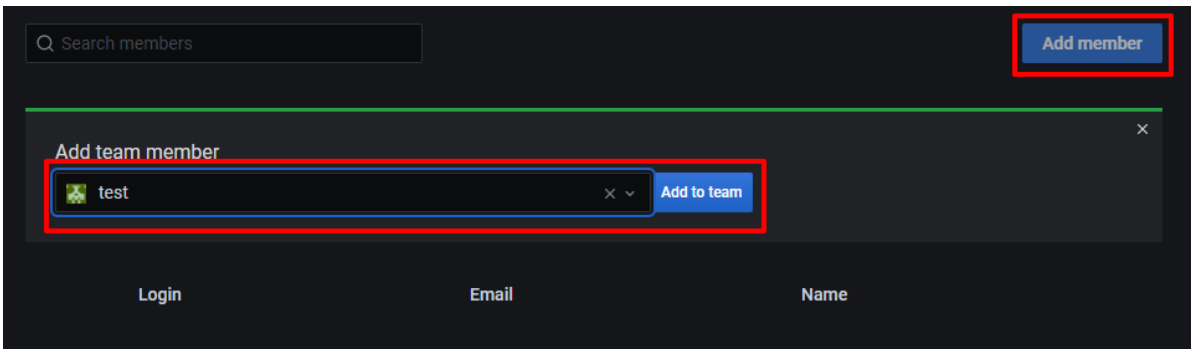


Заполняем поля и нажимаем **Create**.



Добавим пользователя в team. Нажимаем **Add member** > выбираем пользователя в **Add team member** > нажимаем **Add to team**

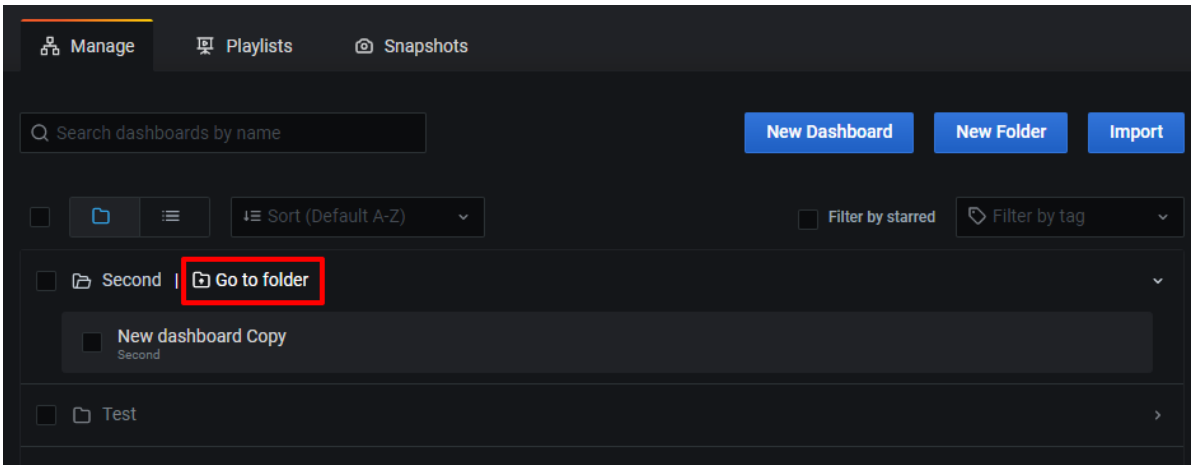
> Имейте в виду, если пользователь не принял приглашение, то его нельзя добавить в team.



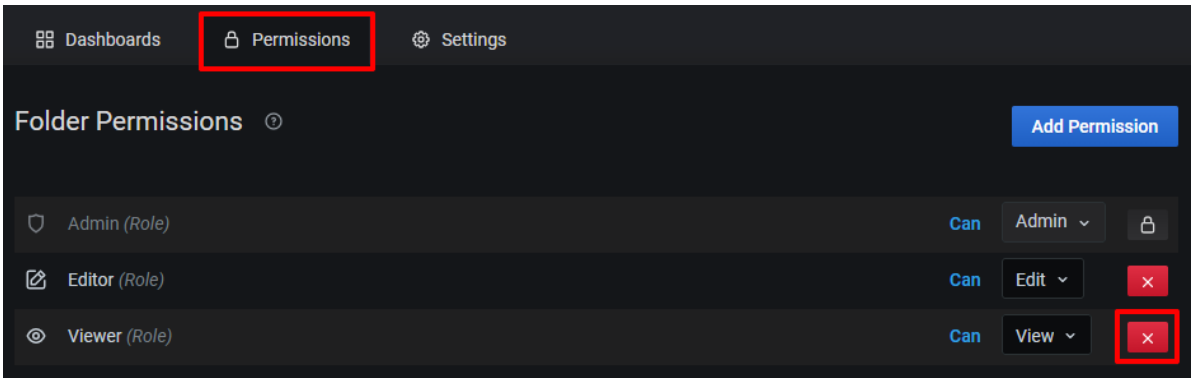
Создадим новую директорию и в ней панель. **Dashboards** > **Manage** > **New folder** >название **Second** > **Create** > **+ Create Dashboard** > **+ Add new panel**

Заведем туда любую метрику, к примеру - **kube_deployment_created** и нажимаем **Save.**

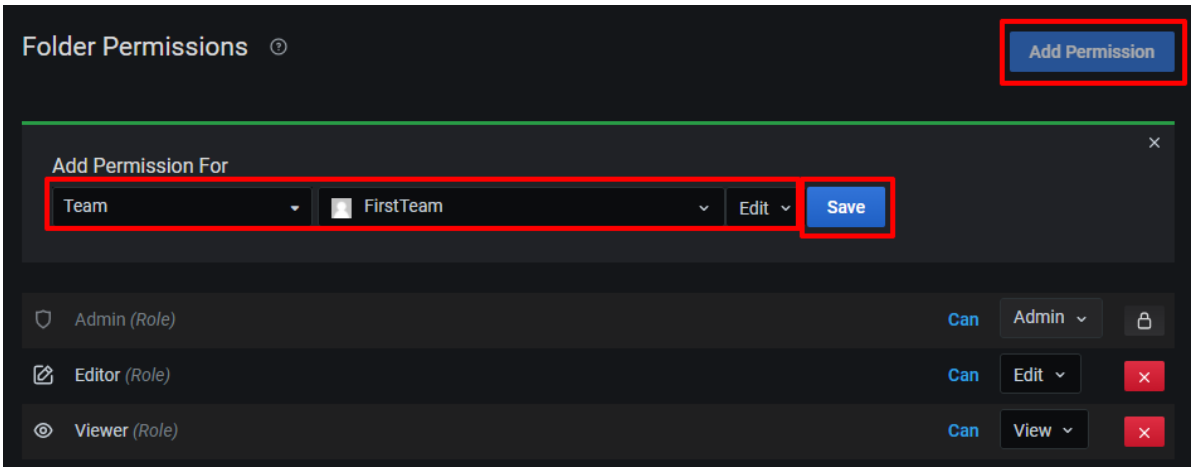
Давайте сходим в директорию **Second**.



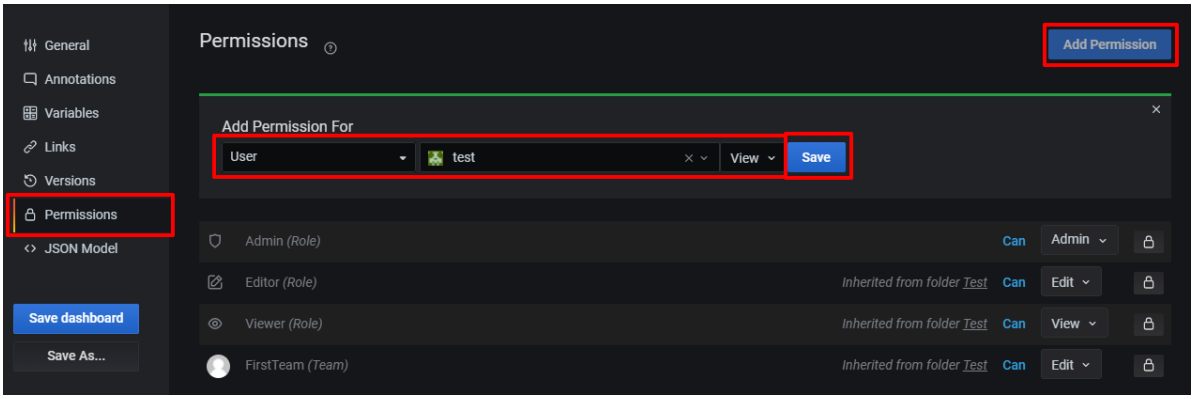
Переходим во вкладку **Permissions** и уберем права на просмотр папки у **Viewer**.



Переходим в папку Test - **Go to folder** и добавляем team - **FirstTeam** права на редактирование **Edit** и жмём **Save**.



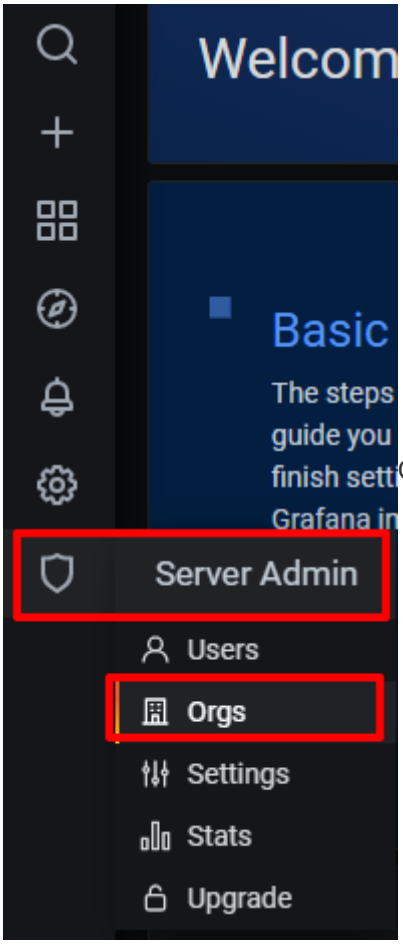
Идём в дашборд - **Our first dash** в папке - **Test.** Заходим в настройки и **Permissions**. Добавим пользователя и права **Add Permission** > Выберем юзера **test** > права **View** > нажмем **Save**.



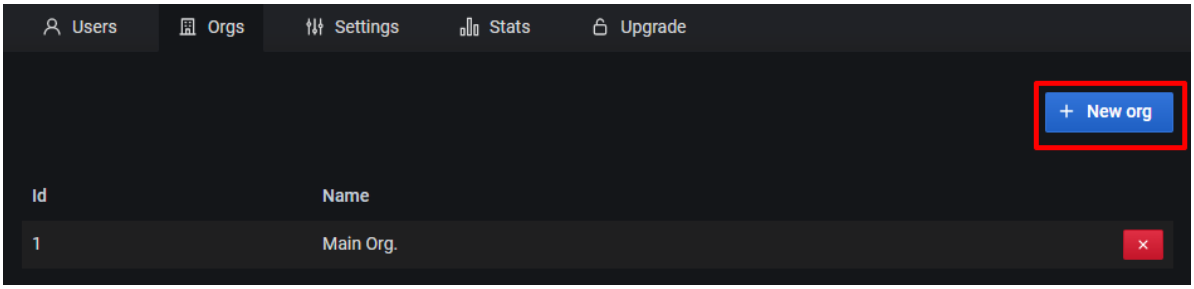
Теперь логинимся с учетной записью **test@grafana.local** и смотрим на что имеет права он.

Orgs

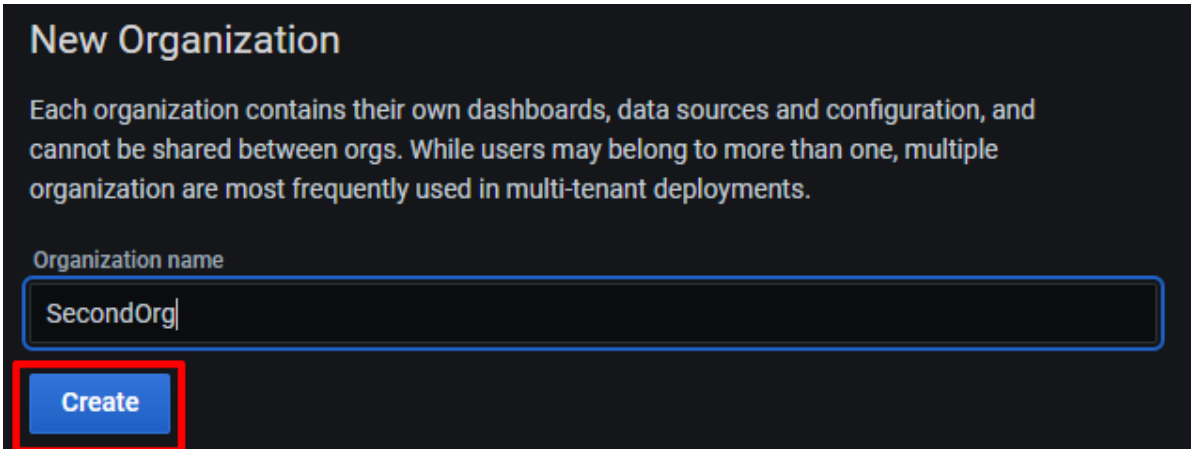
Приступим к изучению Orgs **Server Admin** > **Orgs**.



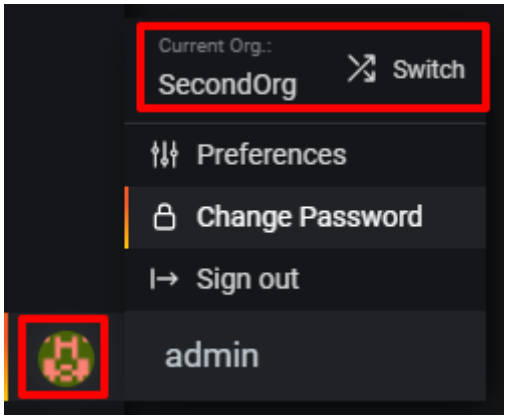
Создаем новую организацию. **+ New org**



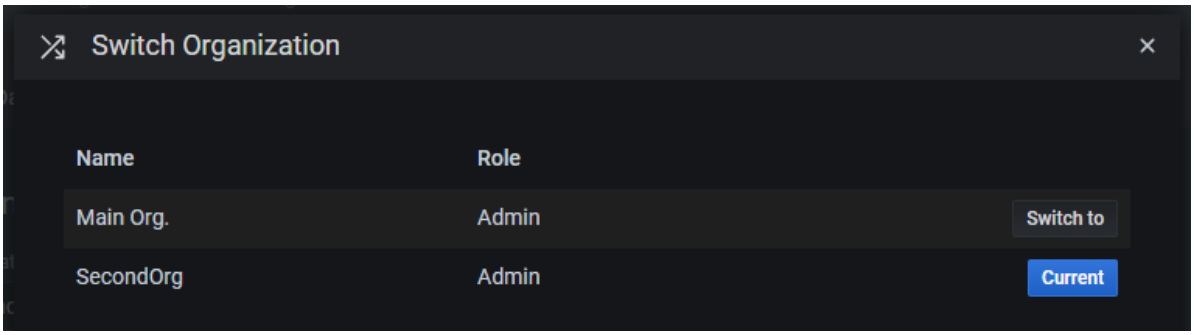
Задаем имя и нажимаем **Create**.



Организация - это высокоуровневая сущность над всеми (users, team, dashboard) и можно просто переключаться между ними.



И выбираем на какую организацию переключится. Все сущности (users, team, dashboard) которые были в панели у вас исчезнуть, и будут заменены на те сущности, которые привязаны к организации на которую вы переключились.



И это ещё не все варианты организации доступа.

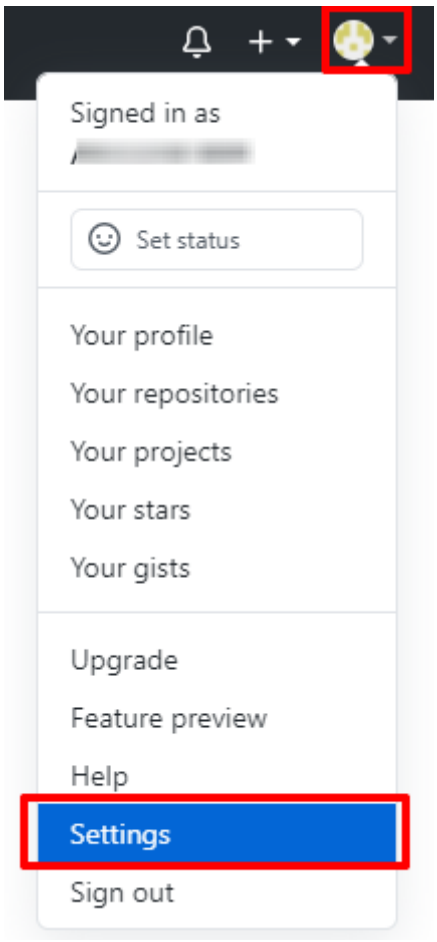
Подготовка GitHub

Прежде чем приступим к следующему шагу, нам нужно настроить GitHub.

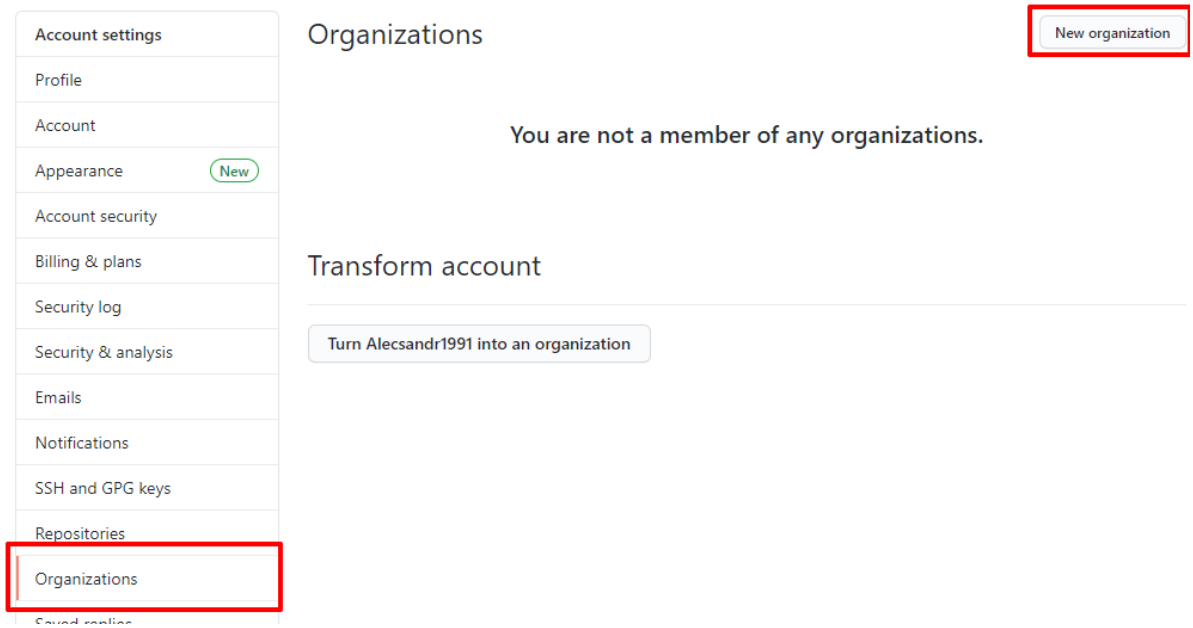
Заходим и регистрируемся - https://github.com/

Создадим организацию.

Заходим в настройки профиля.

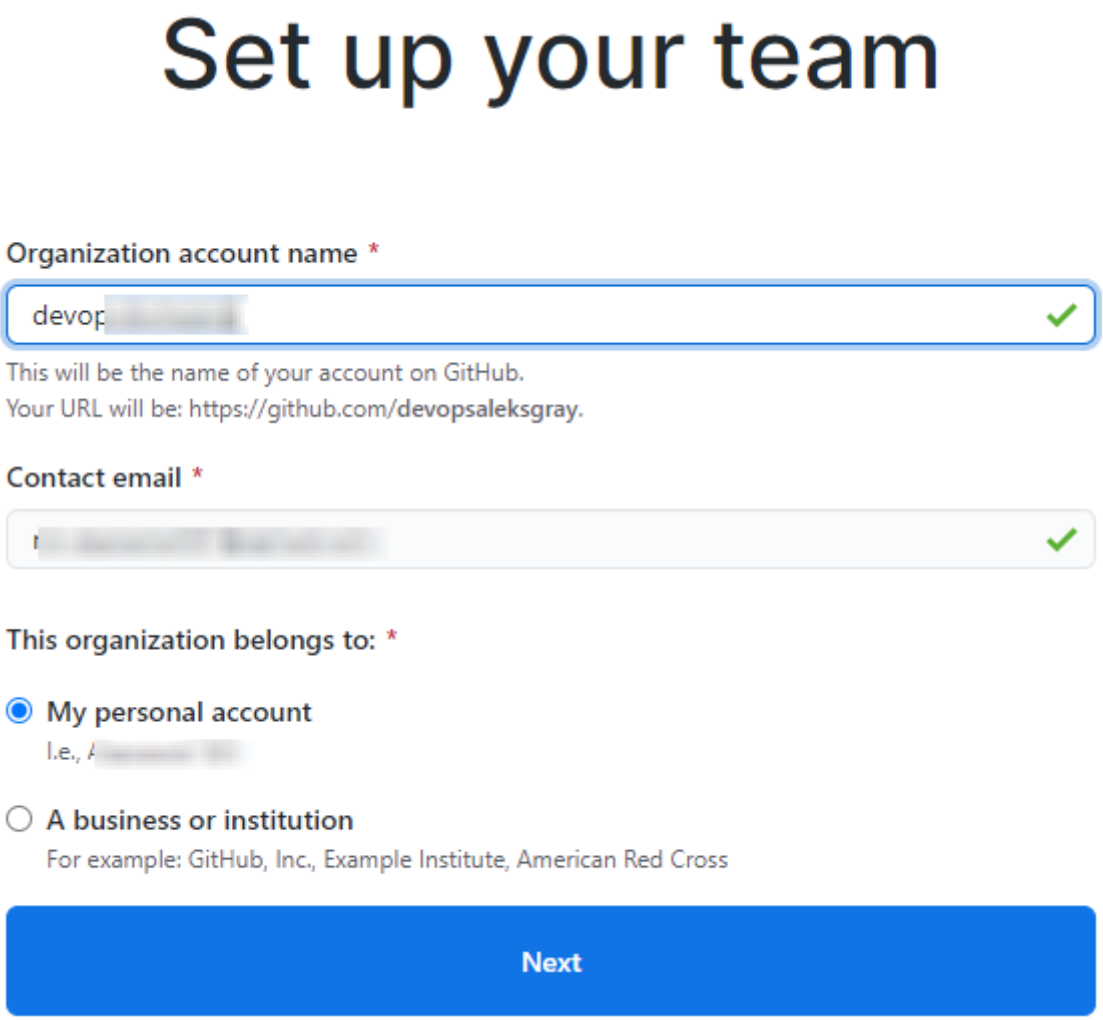


Ogranizations > **New organization**

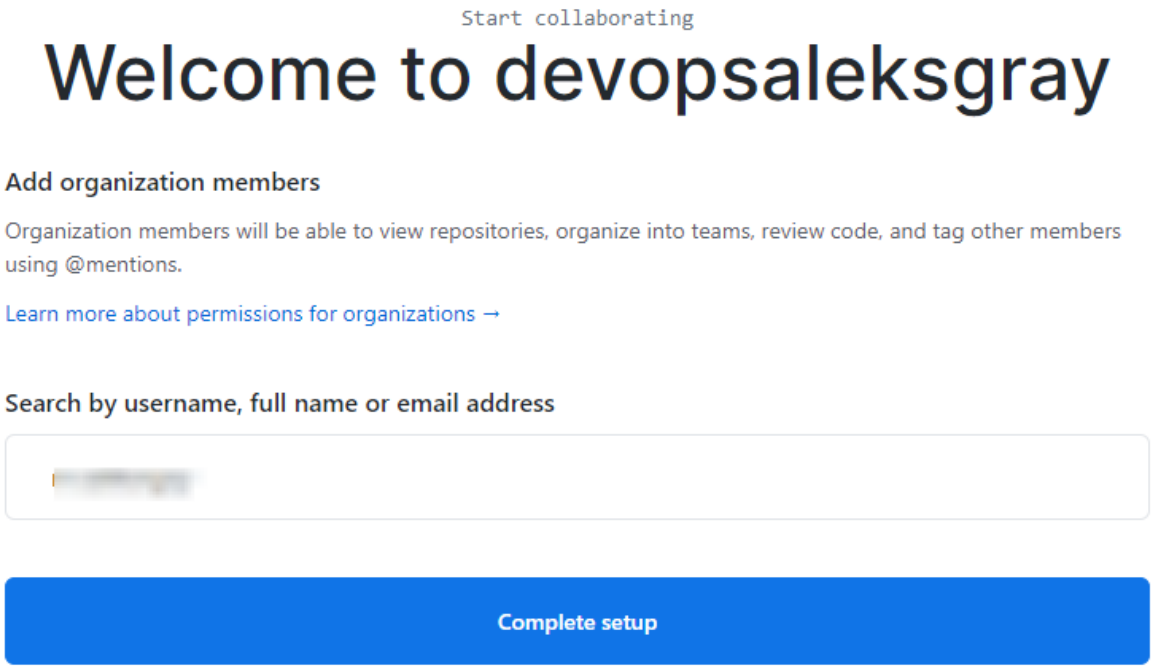


Выбираем план для своей team - **Join for free**

Вводим потом наименование организации **Organization account name** и email **Contact email**.

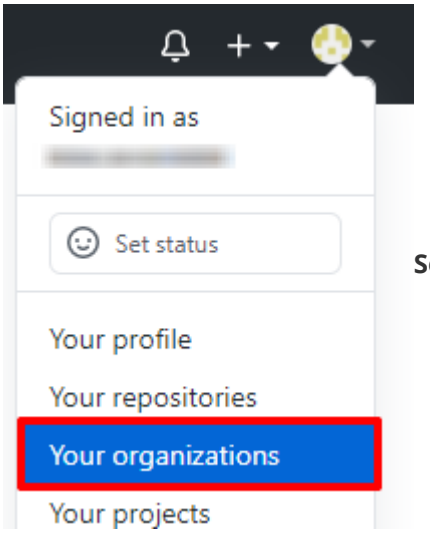


Вписывает свой любой username к примеру и **Complete setup** и далее **Submit** просто**.** Готово!

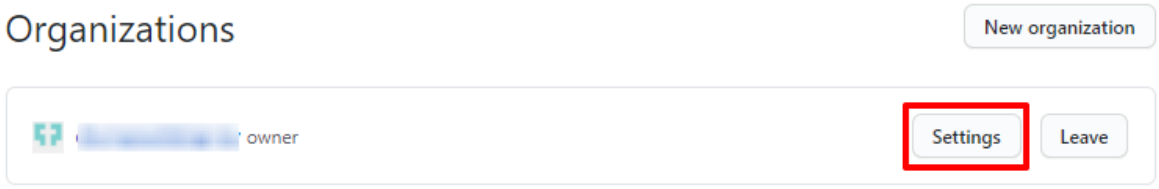


Аутентификация Oauth Apps

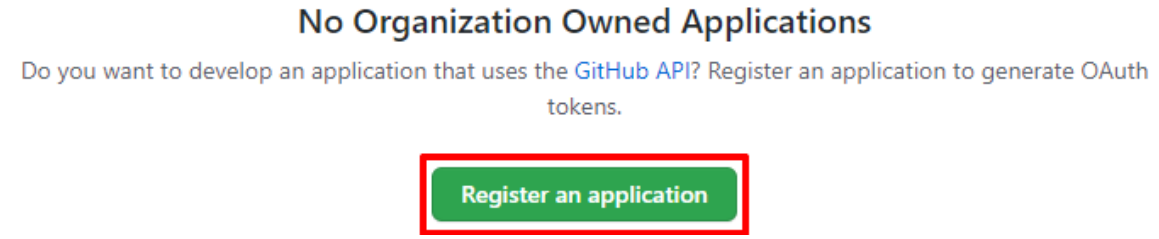
Теперь добавим к организации авторизацию на нашу grafana. Перейдем в профиль и **Your organizations**.



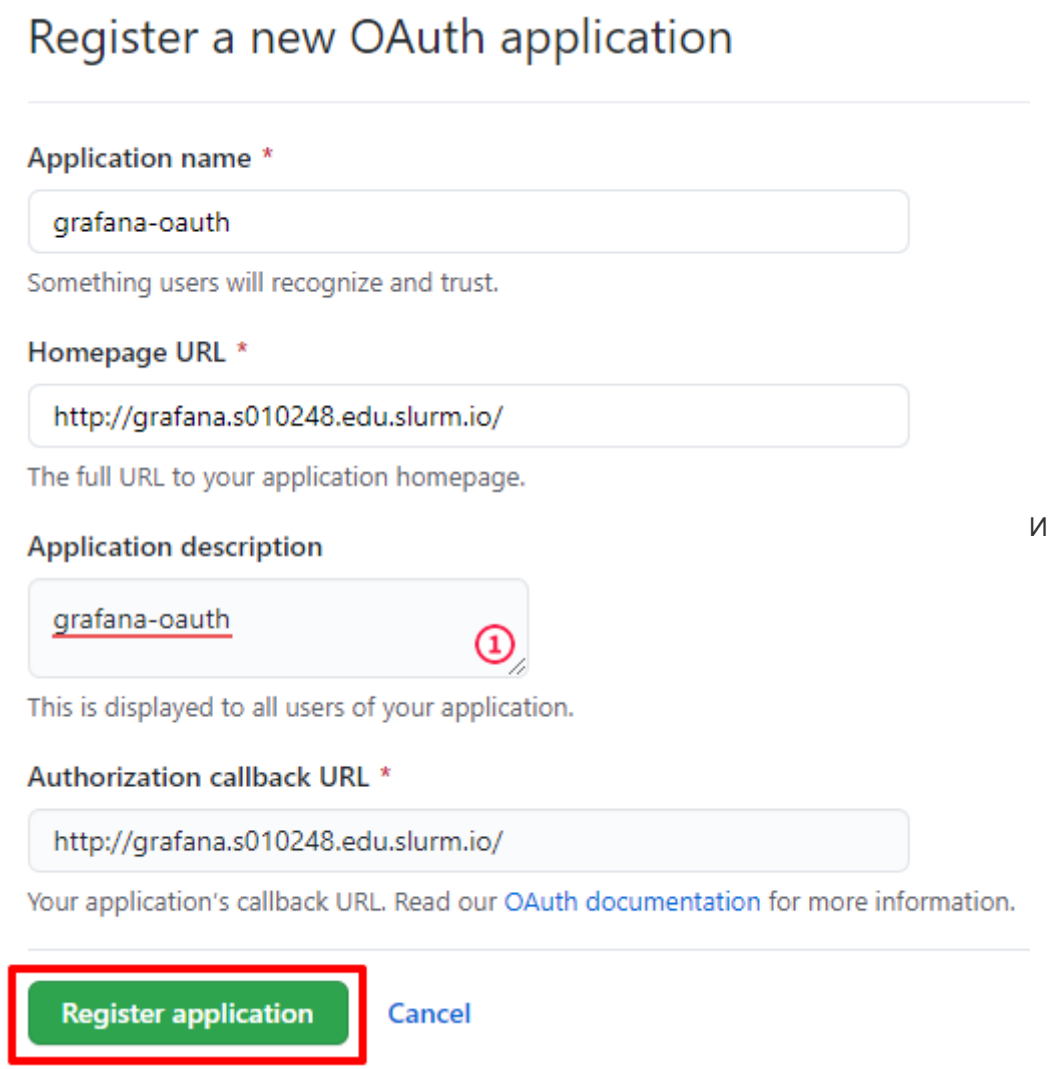
Settings организации, которой вы создали.



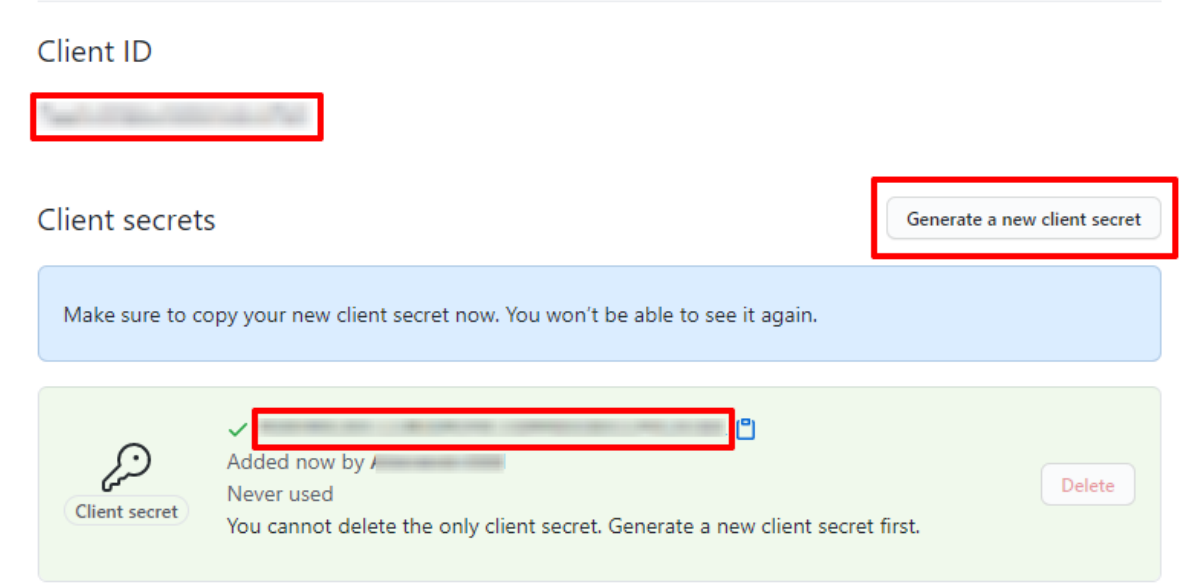
Заходим в **Developer settings** и кликаем **Register an application.**



Заполняем поля и **Register application**.



И вы увидите **Client ID**. Нажмите на **Generate a new client secre**t, чтобы получить **Client secret**. Сохраните эти данные для следующего шага.



Теперь переходим в следующий шаг и произведем настройку аутентификации через GitHub.

Организация пользователей и доступов - Oauth

Настроим аутентификацию с github. Идём в консоль нашу и папку - **lmk8s/5.grafana**

Открываем файл

```
vim grafana-values-2.yaml
```

Корректируем на ваш логин (XXXXXX номер ваш логина).

```
...
  hosts:
    - grafana.sXXXXXX.edu.slurm.io
...
grafana.ini:
  server:
    root_url: http://grafana.sXXXXXX.edu.slurm.io
...
```

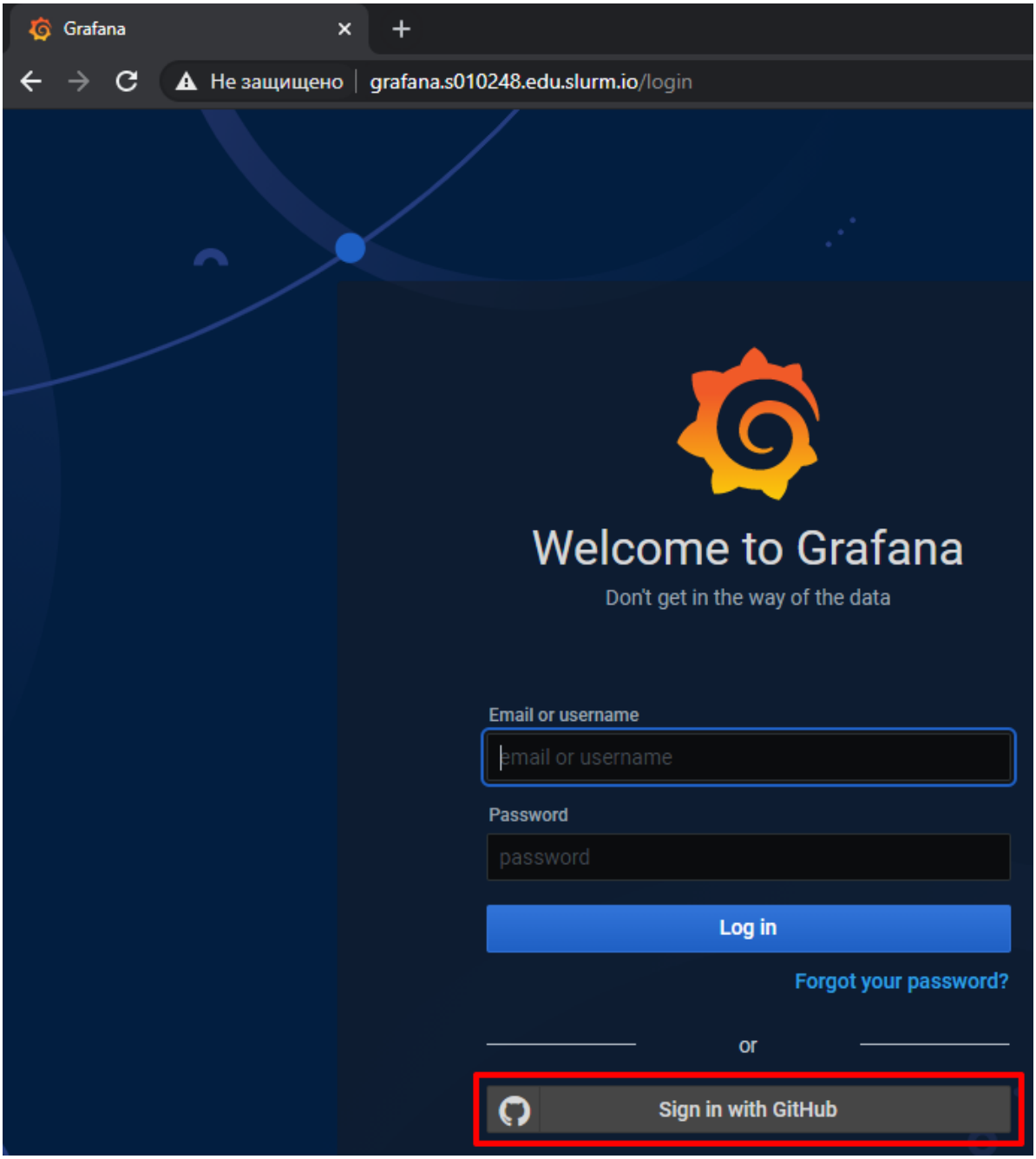
Введём данные, которые вы подготовили в предыдущем шаге.

```
allowed_organizations: название организации
client_id: ******
client_secret: ******
```

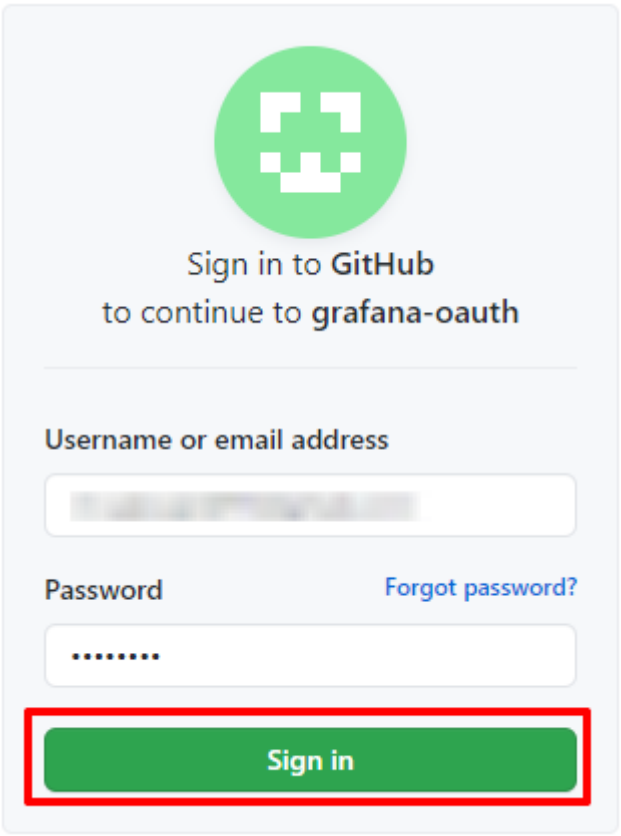
Запускаем обновление grafana.

```
helm upgrade --wait --atomic grafana grafana/grafana -n monitoring --values
./grafana-values-2.yaml
```

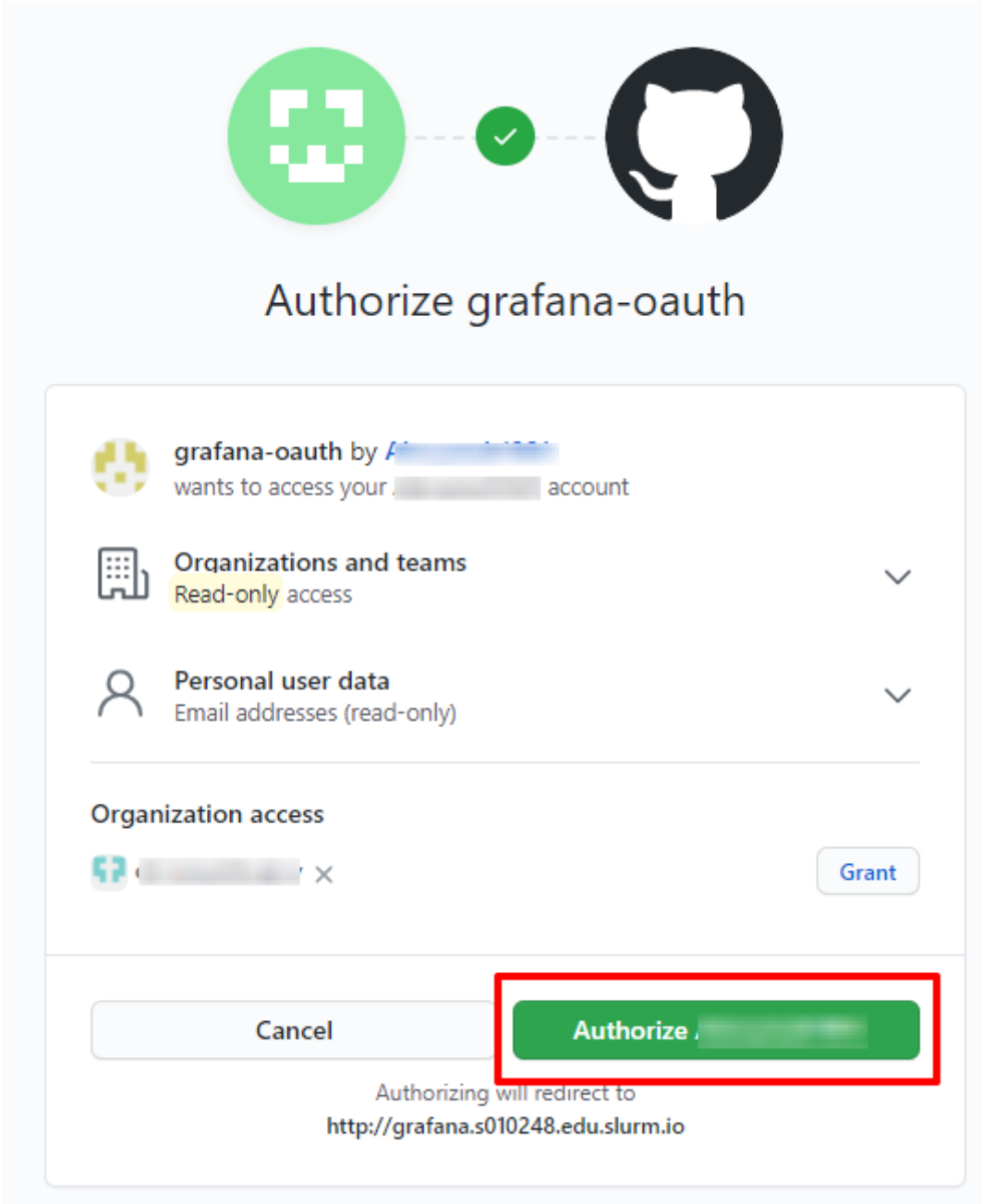
Заходим в grafana и видим, что появилась кнопка логина через GitHub.



Вводим доступ свой от GitHub и **Sign in**.



Нажимаем **Authorize <название организации>**



И мы логинимся в нашу grafana под учетной записью GitHub. Поздравляем вас!

Provisioning/Grafana Enterprise

Как принято работать в современном мире? (всё как код)

Provisioning Directory - директория, через которую можно подкинуть набор JSON-файлов (дашборды, дата сорсы).

Если работать в **K8S** - все можно описывать через файл `values.yaml` и далее раскатывать в кластер через CI/CD конвейер.

IaaC - подход: Terraform, Pulumi.

Grafana Enterprise - это саппорт, Extra datasources, White labeling, и много фич, которых нет в бесплатной версии:

- раздача прав на data sources

- расширенная интеграция LDAP
- Enterprise-плагины типа Oracle, Dynatrace

Практика

Установим Data source.

Настроим наш первый provisioning. Идём в консоль нашу и папку - **lmk8s/5.grafana**

Открываем файл

```
vim grafana-values-3.yaml
```

Корректируем на ваш логин (XXXXXX номер ваш логина)

```
...
  hosts:
    - grafana.sXXXXXX.edu.slurm.io
adminPassword: admin123
...
  server:
    root_url: http://grafana.sXXXXXX.edu.slurm.io
datasources:
...
```

В файле описано, что мы добавляем data source, это аналогично тому что делали ранее вручную.

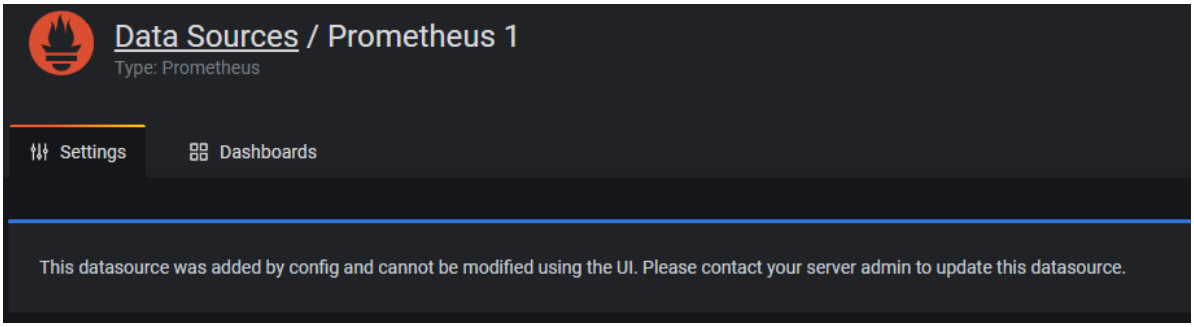
```
datasources:
  "datasources.yaml" :
    apiVersion: 1
    datasources:
      - name: Prometheus 1
        type: prometheus
        url: http://prometheus-server.monitoring.svc.cluster.local
        access: proxy
        isDefault: true
```

Запускаем обновление grafana.

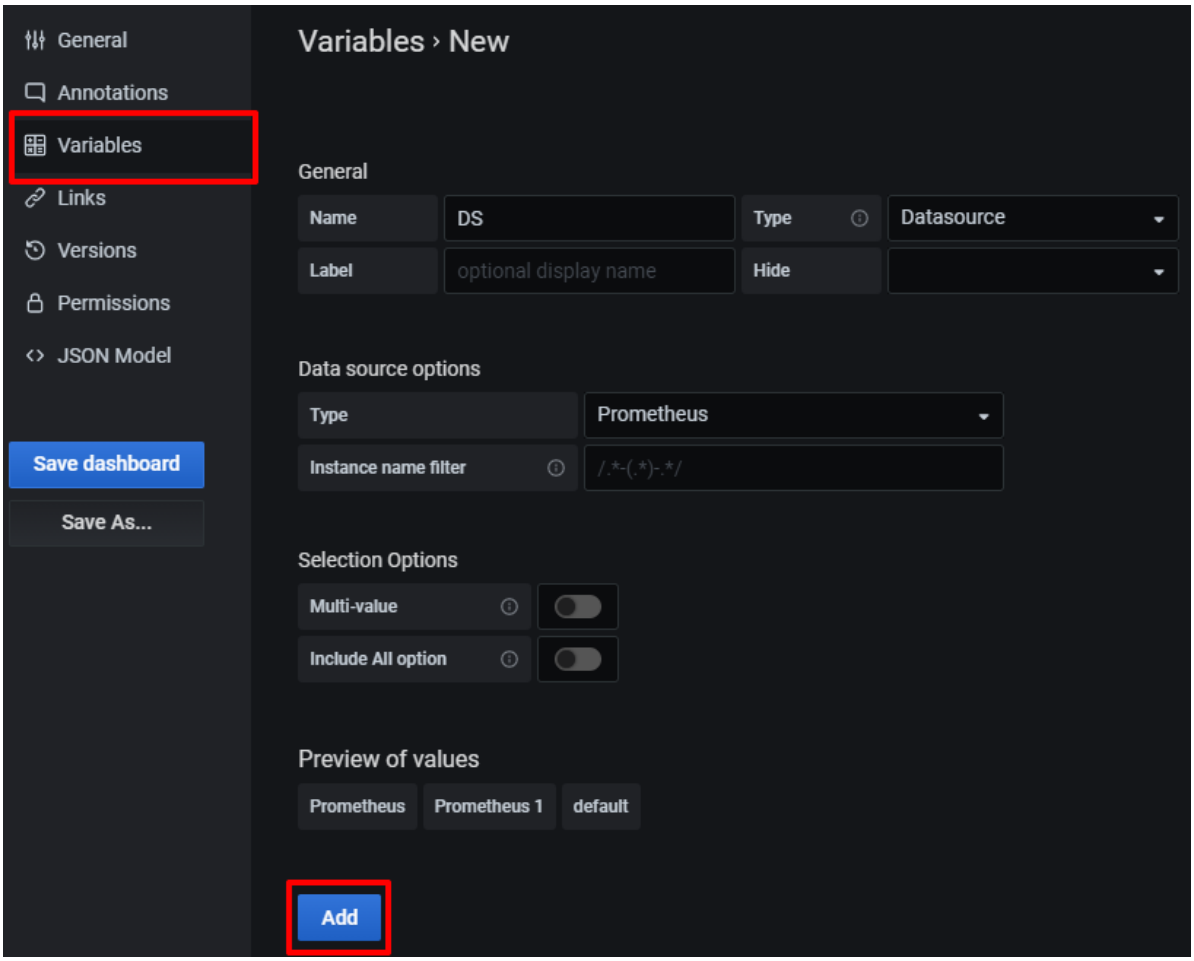
```
helm upgrade --wait --atomic grafana grafana/grafana -n monitoring --values
./grafana-values-3.yaml
```

Заходим в Data source и смотрим, что добавился новый - **Prometheus 1**.

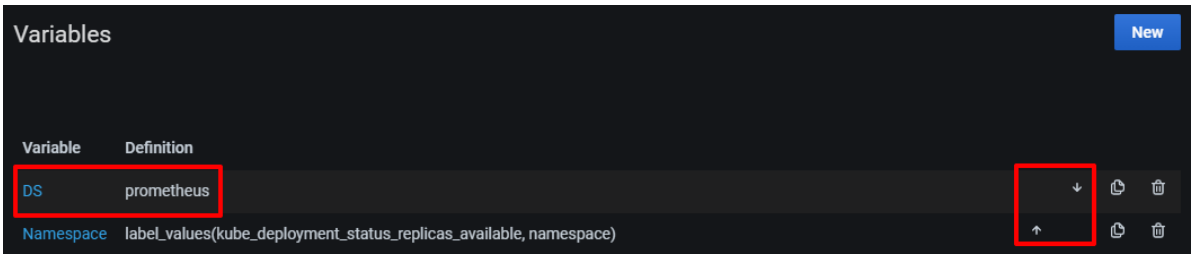
Имейте в виду, что - этот источник данных был добавлен конфигурацией и не может быть изменен с помощью пользовательского интерфейса. Пожалуйста, свяжитесь с администратором вашего сервера, чтобы обновить этот источник данных.



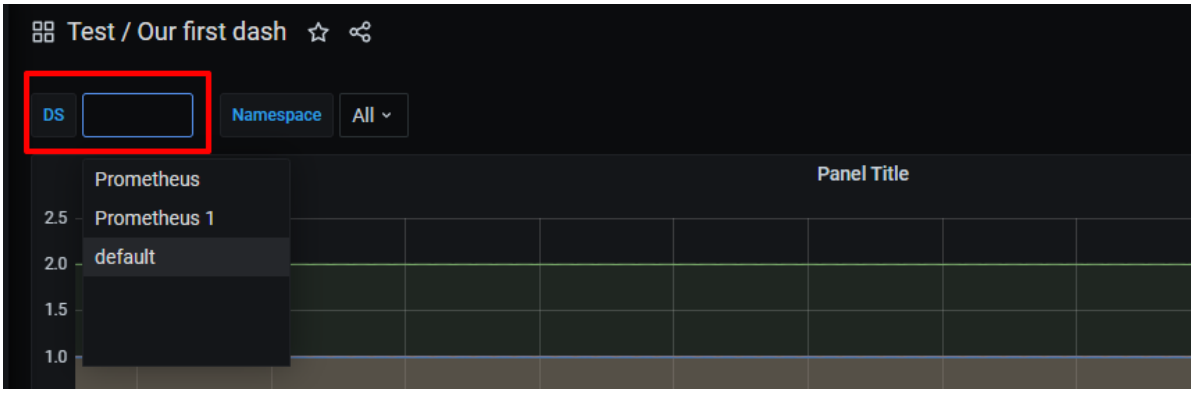
Добавим переменную в дашборд, которая будет давать возможность смену data source типа - Prometheus. Заходим в настройки дашборда - **Our first dash**. **Variables** > **New** > заполним поля > **Add**.



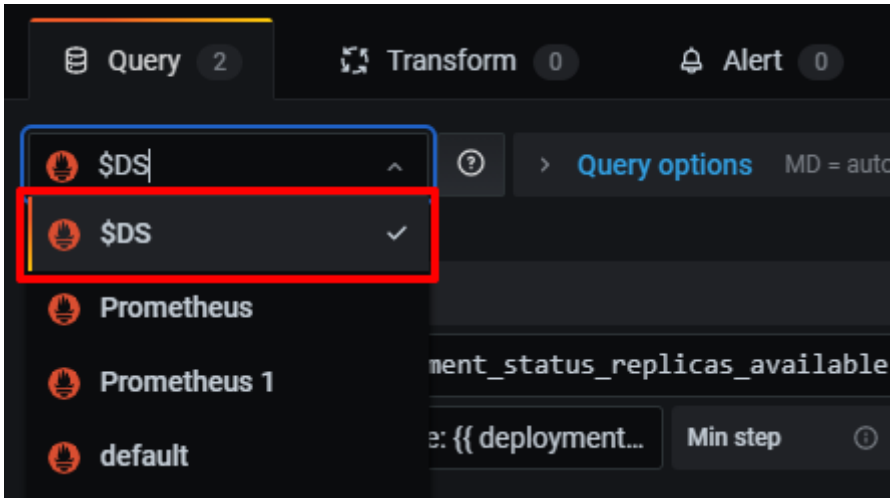
Поменяем порядок переменных, ставим вверх **DS**. Кликаем **Save dashboard**.



У панели в дашборде - **Our first dash**, появился теперь фильтр по Data source.



Изменим выбор Data source в **Edit** панели на **$DS**, и теперь можем в панели менять Data source.



Установим plugin.

Открываем файл

```
vim grafana-values-4.yaml
```

Корректируем на ваш логин (XXXXXX номер ваш логина)

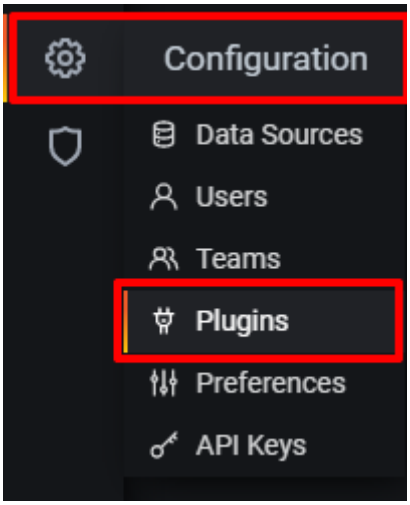
```
...
  hosts:
    - grafana.sXXXXXX.edu.slurm.io
adminPassword: admin123
...
  server:
    root_url: http://grafana.sXXXXXX.edu.slurm.io
datasources:
...
```

Ищем плагин **TrafficLight** тут - https://grafana.com/grafana/plugins и копируем его название с инструкций. Вставляем в файл в конец `grafana-values-4.yaml`

Запускаем обновление grafana.

```
helm upgrade --wait --atomic grafana grafana/grafana -n monitoring --values
./grafana-values-4.yaml
```

Переходив в веб-интерфейс grafana и **Configure** > **Plugins**. В списке у вас будет плагин - **TrafficLight.**



Установим dashboard.

Открываем файл

```
vim grafana-values-5.yaml
```

Корректируем на ваш логин (XXXXXX номер ваш логина)

```
...
  hosts:
    - grafana.sXXXXXX.edu.slurm.io
adminPassword: admin123
...
  server:
    root_url: http://grafana.sXXXXXX.edu.slurm.io
datasources:
...
```

Далее описание, что мы устанавливаем дашборд провайдер с именем - monitoring, далее дашборд с ID 1860, что ранее ставили мы. Далее Data source с именем - Prometheus 1 и еще дашборд с именем - NewDash.

Запускаем обновление grafana.

```
helm upgrade --wait --atomic grafana grafana/grafana -n monitoring --values
./grafana-values-5.yaml
```

Заходим в **Dashboards** > **Manage** и смотрим что создали через **Provisioning**.

Поздравляю вас, вы прошли эту тему полностью!

Prometheus Operator

Общие сведения и установка

CRD - расширение API Kubernetes, чтобы можно было создавать новые абстракции в кластере Kubernetes (`kind:...` в YAML-декларациях создания ресурсов).

Новые **абстракции** - это набор полей. CRD - это описание этих полей.

Экземпляры абстракций - это собственно **Custom Resource (CR)**

Штука, которая мониторит и приводит в соответствие спекам в YAML-ах CR-ы - это **Custom Controller**.

А **оператор** управляет Custom Controller-ами.

Prometheus Operator состоит из нескольких частей:

- **Prometheus** - описывает установку Прометея
- **Alertmanager** - описывает установку Алертменеджера
- **ServiceMonitor** - собственный Service Discovery - какие сервисы надо мониторить, на основании этого ресурса оператор генерит конфиг, по которому будет работать скрейпинг. Если в сервисмониторе засвечивается новый сервис, то под `prometheus-operator` обновляет секрет с конфигом Прометея, добавляет туда настройки скрейпинга. После этого работает контейнер `prometheus-config-reloader`, который отслеживает изменения секретов с конфигами Прометея и обновляет собственно конфиг Прометея.
- **PodMonitor** - то же самое, но просто для подов, для которых не создавались сервисы.
- **Probe** - описание списка ингрессов для добавления в мониторинг, например для BlackBox Exporter
- **PrometheusRule** - описывает набор правил, которые будут добавлены в Prometheus, работает тоже по такому же принципу, что и ServiceMonitor.
- **AlertManagerConfig** - описывает набор алертов, которые будут добавлены в Prometheus.

Рекомендации

- Прометей ставится в кластер как `StatefulSet`, надо не забывать, что при выходе из строя нода из STS, под не перезапускается на другой ноде. Поэтому ставить надо минимум в 2 репликах.
- В новой версии появилась `basic auth` на ингрессе + `Network Policy`, рекомендуется их использовать.
- Retention Size - лимит на количество хранимых данных (по объему) - тоже появилось в новых версиях, и это надо включать. Поскольку по умолчанию Прометей пишет в `EmptyDir`.
- **AlertManager** - то же самое со `StatefulSet`, но надо минимум 3 копии, так как нужен кластер с кворумом и нечетное число реплик.
- `Basic Auth` также надо включать для AlertManager-а.
- **Grafana** - не надо плодить лишних дашбордов, подтягиваться должны только из Git и применять их надо отдельно из `ConfigMap`-ов.!

Установка Prometheus Operator

Цели практики

Целью данной практической работы является демонстрация процесса установки Prometheus Operator в кластер. Установка будет производиться с помощью Helm Chart.

С чем будем работать

- Helm chart **kube-prometheus-stack** - необходим для установки Prometheus Operator в кластер

Где будем выполнять практику

ВНИМАНИЕ: Перед тем, как работать со стендом, его необходимо запустить из Личного кабинета

- Подключаемся по SSH к sbox.slurm.io:

`ssh s<номер вашего логина>@sbox.slurm.io`

- Подключаемся к своему кластеру Kubernetes:

`ssh master-1.s<ваш номер логина>`

- Становимся root:

`sudo -s`

Практика

Подготовительная работа

1. Клонируем манифесты

Для выполнения практической части потребуются манифесты. Их необходимо складировать с gitlab, сделать это можно выполнив команды:

```
cd ~
git clone git@gitlab.slurm.io:edu/lmk8s.git
```

2. Подключаем репо prometheus-community:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

`helm repo update`

3. Создаем namespace

В рамках данного урока Prometheus Operator будет устанавливаться в отдельный namespace: prometheus-operator. Для его создания необходимо выполнить команду:

```
kubectl create ns prometheus-operator
```

4. Создаем Secret для basic-auth

Из соображений безопасности доступ к Prometheus и Alertmanager необходимо ограничить с помощью basic-auth. Для этого создадим секрет, в котором будут данные для basi auth. Имя пользователя и пароль рекомендовано установить такими же, как и для личного кабинета. Для создания секрета необходимо выполнить команды.

`htpasswd -c auth <имя студента, например s0000000>`````

```
kubectl create secret generic admin-basic-auth --from-file=auth -n prometheus-operator
```

5. Подготовка values

Для установки Prometheus будет использоваться уже подготовленные значения values: `~/lmk8s/6.prometheus_operator/operator_values.yml`

Но в них необходимо заменить адрес Ingress для Prometheus, это можно сделать выполнив команду:

```
sed -i 's/<student number>/Ваш номер студента, например s000000/g'
~/lmk8s/6.prometheus_operator/operator_values.yml
```

Установка Prometheus в кластер

Теперь все готово для установки и достаточно выполнить команду:

```
helm upgrade -i prom-operator prometheus-community/kube-prometheus-stack -n
prometheus-operator -f ~/lmk8s/6.prometheus_operator/operator_values.yml
```

В результате будут установлены следующие компоненты:

- Prometheus operator
- Prometheus Server
- Alertmanager
- Grafana
- Kube-state-metrics metrics
- Node exporter
- Servicemonitor для основных компонентов control-plane
- Набор правил alerts для k8s
- Набор dashboards для основных компонентов k8s

Проверка работоспособности

После установки, необходимо проверить, что все работает корректно:

- Открываем в браузере в анонимном режиме `http://prometheus.<номер студента>.edu.slurm.io`

  username: s<номер студента>

  password: Ваш пароль

- Открываем в браузере в анонимном режиме `http://alertmanager.<номер студента>.edu.slurm.io`

  username: s<номер студента>

  password: Ваш пароль

- Открываем в браузере в анонимном режиме `http://grafana.<номер студента>.edu.slurm.io`

  username: admin

  password: prom-operator

Если у вас в браузере 503 ошибка, проверьте что создался секрет с basic-auth. Для этого необходимо выполнить команду: `kubectl get secrets -n prometheus-operator admin-basic-auth`

Ключевые изменения в values.yml

Обратите внимание, что в values.yml для чарта были сделаны изменения, наиболее ключевые из них:

- **Заданы ресурсы для все объектов** - это общее хорошее правило для всего, что запускается в Kubernetes кластере. Во первых, заданные request и limits позволяют scheduler корректно распределять Pod по node. Во вторых, это важно для избегания ситуации утечки ресурсов, а в случае с Prometheus это особенно актуально, так как он является достаточно "прожорливым".
- Задан **podDisruptionBudget** - данная настройка влияет на то, сколько Pod может быть одновременно выключено (распространяется только на evection api). Она позволяет гарантировать, что при обслуживании кластера Kubernetes не будут выключены все Pod с Prometheus.
- **podAntiAffinity** - данная настройка позволяет гарантировать, что Pod из statefulset не будут запущены на одной node. Без данной настройки возможна ситуация, когда все Pod с Prometheus и Alertmanager будут запущены на одной node, и в случае выхода из строя этой node, кластер останется без мониторинга (Pod для statefulset не перезапускаются автоматически)
- **retentionSize** - эта настройка позволяет ограничить хранимую историю по размеру. Обратите внимание, что мы запрашиваем pvc на 1Gb, а размер ограничен 700Mb, это связано с тем, что фактический размер pv будет немного меньше 1Gb.
- **serviceMonitorSelectorNilUsesHelmValues** и **podMonitorSelectorNilUsesHelmValues** - это настройка влияет на то, в какие метки должны быть у ServiceMonitor и PodMonitor. При значение true, они должны совпадать с labels, которые выставляет Helm на prometheus operator, как правило, это неудобно.

Настройка мониторинга ресурсов

Настройка Servicemonitor и Podmonitor

Цели практики

Целью данной практической работы является знакомство со структурой custom resource Servicemonitor и Podmonitor.

Servicemonitor

Servicemonitor - это абстракция, с помощью которой реализован механизм Service discovery в Prometheus Operator. В данной абстракции описывается, из каких Service необходимо получить список Endpoint, и задаются правила, по которым необходимо производить scraping. В общем случае манифест Servicemonitor выглядит примерно так:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prom-kube-prometheus-nodexporter
spec:
  endpoints:
  - interval: 1m
    path: /metrics
    port: metrics-port
  namespaceSelector:
```

```
    matchNames:
      - prometheus-operator
  selector:
    matchLabels:
      app: example
```

Назначение полей:

`spec.endpoints` - содержит настройки, по которым должен производиться scraping, такие как: частота опроса, endpoint, может быть указан аутентификация, настройки tls и так далее.

`spec.endpoints.port` - задается порт, по которому будет производиться scraping. Важной особенностью является то, что Servicemonitor работает только с именованными портами, то есть указать port: 443 нельзя.

`spec.namespaceSelector` - задает, в каком namespace будет производиться scraping. Возможные значения: matchNames и any: true/false. В случае использования any: true будет производиться по всем namespace.

`spec.selector` - содержит настройку, какие labels должна содержать служба, для которой будет получаться список endpoints. Может содержать следующие параметры:

`matchLabels` - список labels, которые должны быть у службы.

`matchExpressions` - регулярное выражение для discovery Service на основании наличия или отсутствие labels. Формат регулярных выражений:

- key: serviceapp
  operator: Exists

Полный список полей можно посмотреть здесь: https://docs.openshift.com/container-platform/4.4/rest_api/monitoring_apis/servicemonitor-monitoring-coreos-com-v1.html

PodMonitor

PodMonitor - это абстракция, с помощью которой реализован механизм Service discovery в Prometheus Operator. В данной абстракции описываются, какие Pod необходимо добавить в Prometheus и правила, по которым необходимо производить scraping. В общем случае, манифест PodMonitor выглядит примерно так:

```
apiVersion: monitoring.coreos.com/v1`
`kind: PodMonitor`
`metadata:`
` name: example-app`
`spec:`
` namespaceSelector:`
`  matchNames:`
`    - prometheus-operator  `
` selector:`
`  matchLabels:`
`    app: example`
` podMetricsEndpoints:`
` - port: web`
`  honorLabels: true
```

`spec.namespaceSelector` - задает, в каком namespace будет производиться scraping. Возможные значения: `matchNames` и `any: true/false`. В случае использования `any: true` будет производиться по всем namespace.

`spec.selector` - содержит настройку, какие метки должна содержать служба, для которой будет получаться список endpoints. Может содержать следующие параметры: `matchNames` и `any: true/false`. В случае использования `any: true` будет производиться по всем namespace.

`podMetricsEndpoints` - содержит описание, какие порты должны использоваться для scraping, а также другие настройки, относящиеся к scraping.

Полный список полей можно посмотреть здесь: https://docs.openshift.com/container-platform/4.4/rest_api/monitoring_apis/podmonitor-monitoring-coreos-com-v1.html

Создайте Servicemonitor для Nginx ingress с именем: `prom-kube-prometheus-nginx`.

Также сервис должен содержать следующие labels:

```
app: nginx-ingress`
`component: controller`
`release: nginx-ingress
```

Порт для scraping называется: `metrics`. Интервал для scraping: `1m`, а поиск службы должен производиться по всем namespace.

Настройка Prometheus rules

Целью данной практической работы является знакомство со структурой custom resource: PrometheusRule.

PrometheusRule

Custom resource PrometheusRule достаточно простой, он содержит стандартные поля Kubernetes абстракции, а в поле spec описываются Rules с таким же синтаксисом, как и у Prometheus. В общем случае, PrometheusRule выглядит примерно так:

```
apiVersion: monitoring.coreos.com/v1`
`kind: PrometheusRule`
`metadata:`
` name: prometheus-example-rules`
`spec:`
` groups:`
` - name: node.rules`
`  rules:`
`  - expr: sum(min(kube_pod_info{node!=""}) by (cluster, node))`
`   record: ':kube_pod_info_node_count:'`
`  - expr: |-`
`    topk by(namespace, pod) (1,`
`     max by (node, namespace, pod) (`
`      label_replace(kube_pod_info{job="kube-state-metrics",node!=""}, "pod",
"$1", "pod", "(.*)")`
`    ))`
`   record: 'node_namespace_pod:kube_pod_info'
```

Создайте PrometheusRule с именем: `prom-ingress.rules`,

который вычисляет выражение: `rate(nginx_ingress_controller_requests[5m])`

и сохраняет его с именем: `nginx_ingress_controller_requests_per_second`.

Имя группы: `Ingress`

Логирование

Error Reporting

Tracing

Мониторинг и логирование Kubernetes в Production