

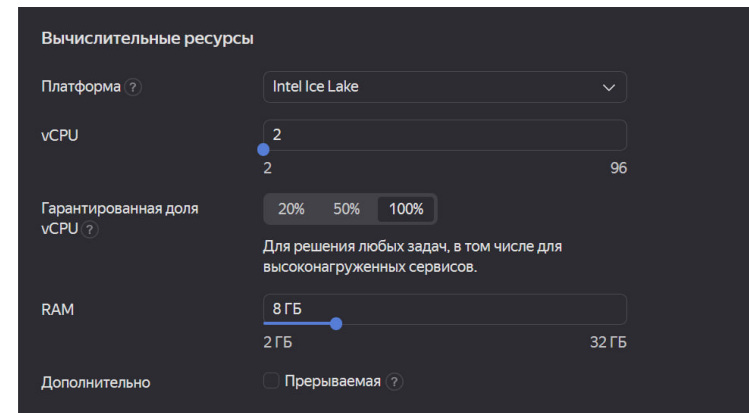
Домашнее задание. Сервисы централизованного логирования для компонентов Kubernetes и приложений

Подготовка Kubernetes кластера

Любым способом (вручную через web-интерфейс, консольным клиентом, утилитой Terraform) разверните managed Kubernetes кластер в YC.

Для выполнения домашнего задания понадобится:

- Как минимум 1 нода типа `intel ice lake, 2vCPU, 8 GB RAM` в default-pool
- Как минимум 3 ноды типа `intel ice lake, 2vCPU, 8 GB RAM` в infra-pool



Вычислительные ресурсы

Платформа ?	Intel Ice Lake
vCPU	2
Гарантированная доля vCPU ?	20% 50% 100%
RAM	8 ГБ
Дополнительно	<input type="checkbox"/> Прерываемая ?

Для решения любых задач, в том числе для высоконагруженных сервисов.

Подготовка Kubernetes кластера

Подготовить кластер можно из инструкции: <https://cloud.yandex.ru/docs/managed-kubernetes/quickstart?from=int-console-empty-state>

Создание и конфигурация нод: <https://cloud.yandex.ru/docs/managed-kubernetes/operations/node-group/node-group-create>

Подготовка Kubernetes кластера

Как можно догадаться из названия, мы планируем отдать три из четырех нод кластера под инфраструктурные сервисы.

Присвоим этим нодам определенный **taint**, чтобы избежать запуска на них случайных pod.

```
kubectl taint nodes node1 node-role=infra:NoSchedule
```

Подготовка Kubernetes кластера

В результате должна получиться следующая конфигурация кластера:

```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
node0-default-pool	Ready	<none>	11m27s	v1.22.4
node1-infra-pool	Ready	<none>	55s	v1.22.4
node2-infra-pool	Ready	<none>	56s	v1.22.4
node3-infra-pool	Ready	<none>	55s	v1.22.4

Установка HipsterShop

Для начала, установим в Kubernetes кластер уже знакомый нам HipsterShop.

Самый простой способ сделать это - применить подготовленный манифест:

```
kubectl create ns microservices-demo  
kubectl apply -f https://raw.githubusercontent.com/express42/otus-platform-snippets  
/master/Module-02/Logging/microservices-demo-without-resources.yaml -n  
microservices-demo
```

Проверьте, что все pod развернулись на ноде из default-pool:

```
kubectl get pods -n microservices-demo -o wide
```

Установка EFK стека | Helm charts

В данном домашнем задании мы будем устанавливать и использовать различные решения для логирования различными способами.

Начнем с "классического" набора инструментов (**ElasticSearch**, **Fluent Bit**, **Kibana**) и "классического" способа его установки в Kubernetes кластер (**Helm**).

Рекомендуемый репозиторий с Helm chart для ElasticSearch и Kibana на текущий момент - <https://github.com/elastic/helm-charts>

Добавим его:

```
helm repo add elastic https://helm.elastic.co
```

Установка EFK стека | Helm charts

И установим нужные нам компоненты, для начала - без какой-либо дополнительной настройки:

```
kubectl create ns observability
```

```
# Elasticsearch
```

```
helm upgrade --install elasticsearch elastic/elasticsearch --namespace observability
```

```
# Kibana
```

```
helm upgrade --install kibana elastic/kibana --namespace observability
```

```
# Fluent Bit
```

```
helm upgrade --install fluent-bit stable/fluent-bit --namespace observability
```

Установка EFK стека | Helm charts

Если посмотреть, как установленные нами сервисы распределились по нодам, можно догадаться, что что-то пошло не по плану, все сервисы с переменным успехом попытались запуститься только на одной ноде из default-pool.

Попробуем исправить это и запустить каждую реплику Elasticsearch на своей, выделенной ноде из infra-pool.

Создайте в директории `kubernetes-logging` файл `elasticsearch.values.yaml`, будем указывать в этом файле нужные нам values.

Установка EFK стека | Elasticsearch

Для начала, обратимся к файлу `values.yaml` в репозитории и найдем там ключ `tolerations`. Мы помним, что ноды из infra-pool имеют taint `node-role=infra:NoSchedule`. Давайте разрешим Elasticsearch запускаться на данных нодах:

```
tolerations:  
  - key: node-role  
    operator: Equal  
    value: infra  
    effect: NoSchedule
```

Обновите установку:

```
helm upgrade --install elasticsearch elastic/elasticsearch --namespace observability  
-f elasticsearch.values.yaml
```

Установка EFK стека | Elasticsearch

Теперь Elasticsearch **может** запускаться на нодах из infra-pool, но это не означает, что он **должен** это делать.

Исправим этот момент и добавим в `elasticsearch.values.yaml` NodeSelector, определяющий, на каких нодах мы можем запускать наши pod.

```
nodeSelector:  
  cloud.yandex.ru/yandex-nodepool: infra-pool
```

Другой, и, на самом деле, более гибкий способ осуществить задуманное - `nodeAffinity`

Установка EFK стека

Если все выполнено корректно, то через некоторое время мы сможем наблюдать следующую картину:

```
kubectl get pods -n logging -o wide -l chart=elasticsearch
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
elasticsearch-master-0 infra-pool	1/1	Running	0	1m22s	10.40.3.4	node1-
elasticsearch-master-1 infra-pool	1/1	Running	0	2m34s	10.40.2.4	node2-
elasticsearch-master-2 infra-pool	1/1	Running	0	4m22s	10.40.1.4	node3-

Пока остановимся на этом и перейдем к следующему компоненту инсталляции

Установка nginx-ingress | Самостоятельное задание

Для того, чтобы продолжить установку EFK стека и получить доступ к Kibana, предварительно потребуется развернуть ingress-controller.

В предыдущих ДЗ мы рассматривали процесс в деталях, так что данное задание предлагается выполнить самостоятельно.

Установите nginx-ingress. **Должно быть развернуто три реплики controller, по одной, на каждую ноду из infra-pool**

Если с соответствием указанным требованиям возникают какие-либо проблемы, можно воспользоваться [примером values.yaml](#)

Установка EFK стека | Kibana

По традиции создадим файл `kibana.values.yaml` в директории `kubernetes-logging` и добавим туда конфигурацию для создания ingress:

```
ingress:
  enabled: true
  annotations: {
    kubernetes.io/ingress.class: nginx
  }
  path: /
  hosts:
    - kibana.<YOUR_IP>.xip.io
```

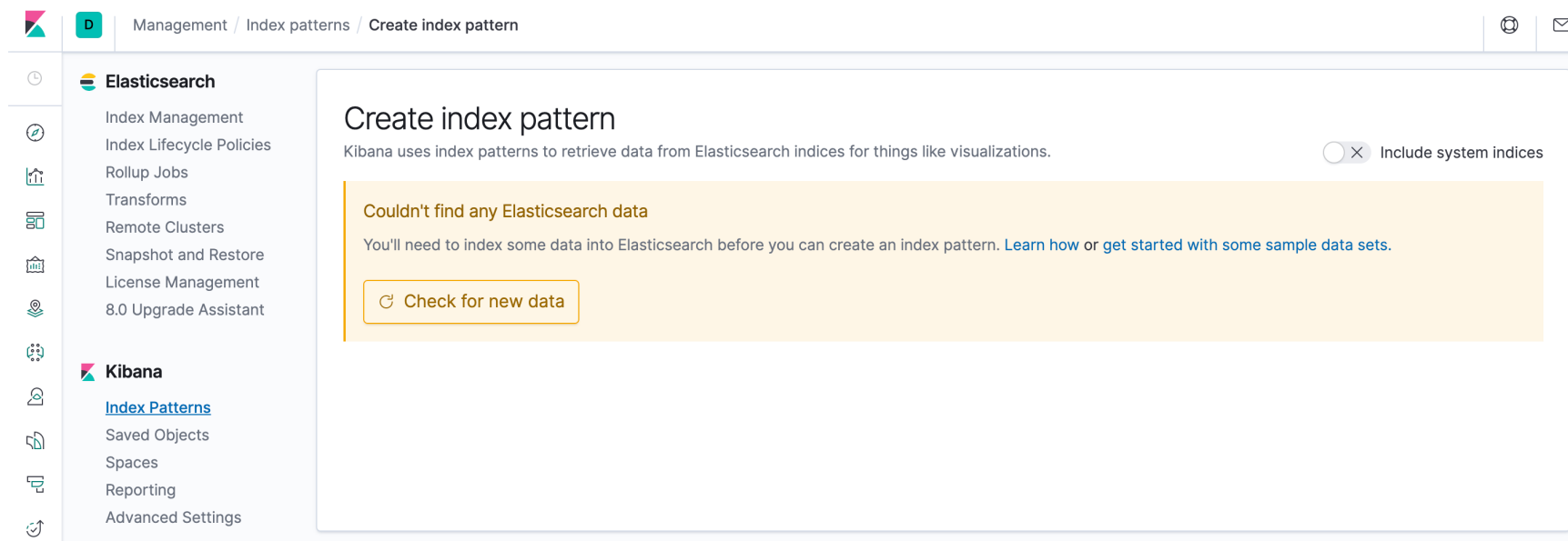
Обновим релиз:

```
helm upgrade --install kibana elastic/kibana --namespace observability -f
kibana.values.yaml
```

Установка EFK стека

После прохождения всех предыдущих шагов у вас должен появиться доступ к Kibana по URL `kibana.<YOUR_IP>.xip.io`

Попробуем создать `index pattern`, и увидим, что в ElasticSearch пока что не обнаружено никаких данных:



Установка EFK стека

Посмотрим в логи решения, которое отвечает за отправку логов (Fluent Bit) и увидим следующие строки:

```
kubectl logs fluent-bit-4bsmz -n logging --tail 2
[2020/02/10 14:33:15] [ warn] net_tcp_fd_connect: getaddrinfo(host='fluentd'): Name
or service not known
[2020/02/10 14:33:15] [error] [out_fw] no upstream connections available
```

Установка EFK стека

Попробуем исправить проблему. Создадим файл `fluent-bit.values.yaml` и добавим туда:

```
backend:  
  type: es  
  es:  
    host: elasticsearch-master
```

Описание того, что мы сделали, можно найти в документации и в оригинальном файле `values.yaml`

Установка EFK стека

Попробуем повторно создать `index pattern`. В этот раз ситуация изменилась, и какие-то индексы в Elasticsearch уже есть

Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

☐ Include system indices

Step 1 of 2: Define index pattern

Index pattern

kubernetes_cluster-*

You can use a * as a wildcard in your index pattern.
You can't use spaces or the characters \, /, ?, ", <, >, |.

> Next step

✓ **Success!** Your index pattern matches **1 index**.

kubernetes_cluster-2020.02.10

Rows per page: 10 ▾

Установка EFK стека

После установки можно заметить, что в Elasticsearch попадают далеко не все логи нашего приложения.

Причину можно найти в логах pod с Fluent Bit, он пытается обработать JSON, отдаваемый приложением, и находит там дублирующиеся поля `time` и `timestamp`

| [GitHub issue](#), с более подробным описанием проблемы

Установка EFK стека

Вариантов решения проблемы, озвученной ранее, несколько:

- Полностью отключить парсинг JSON внутри лога ключом `filter.mergeJSONLog=false`
- Складывать содержимое лога после парсинга в отдельный ключ (в нашем случае для некоторых микросервисов возникнет проблема `illegal_argument_exception` с полем `time`)
- Изменить имя ключа (`time_key`) с датой, добавляемое самим Fluent Bit, на что-то, отличное от `@timestamp`. Это не решит проблему с тем, что останется поле `time`, которое также будет помечено дублирующимся
- "Вырезать" из логов поля `time` и `@timestamp`

Установка EFK стека

Мы пойдем сложным путем и воспользуемся фильтром **Modify**, который позволит удалить из логов "лишние" ключи

Пример итогового `fluent-bit.values.yaml`:

```
backend:
  type: es
  es:
    host: elasticsearch-master
rawConfig: |
  @INCLUDE fluent-bit-service.conf
  @INCLUDE fluent-bit-input.conf
  @INCLUDE fluent-bit-filter.conf
  @INCLUDE fluent-bit-output.conf

[FILTER]
  Name      modify
  Match     *
  Remove    time
  Remove    @timestamp
```

Установка EFK стека | Задание со

- Детально изучите возникшую ситуацию с дублирующими полями
- Предложите более элегантное решение в контексте конкретной ситуации и реализуйте его
- Все артефакты решения, в частности, файл `fluent-bit.values.yaml` приложите к PR

Мониторинг Elasticsearch

Помимо установки Elasticsearch, важно отслеживать его показатели и вовремя понимать, что пора предпринять какие-либо действия. Для мониторинга Elasticsearch будем использовать следующий **Prometheus exporter**

- Установите **prometheus-operator** в namespace **observability** любым удобным вам способом
- Установите упомянутый выше exporter:

```
helm upgrade --install elasticsearch-exporter stable/elasticsearch-exporter --set  
es.uri=http://elasticsearch-master:9200 --set serviceMonitor.enabled=true  
--namespace=observability
```

Мониторинг Elasticsearch

Импортируйте в Grafana один из популярных **Dashboard** для Elasticsearch exporter, содержащий визуализацию основных собираемых метрик:

Importing Dashboard from [Grafana.com](https://grafana.com)

Published by	Kristian Jensen
Updated on	2018-01-16 10:21:48

Options

Name	ElasticSearch Overview	✓
Folder	General ▾	
Unique identifier (uid)	<i>i</i> auto-generated	change
Prometheus	<i>i</i> Prometheus	▾ ✓

Import

Cancel

Мониторинг Elasticsearch

Проверим, что метрики действительно собираются корректно. Сделайте drain одной из нод infra-pool

```
kubectl drain <NODE_NAME> --ignore-daemonsets
```

Статус Cluster Health остался зеленым, но количество нод в кластере уменьшилось до двух штук. При этом, кластер сохранил полную работоспособность.

Попробуем сделать drain второй ноды из infra-pool, и увидим что **PDB** не дает этого сделать.

Про PDB мы поговорим на следующих лекциях, а пока вручную удалите pod, находящийся на ноде, для которой мы пытались сделать drain.

Мониторинг Elasticsearch

После данного действия можно заметить следующее:

- Оставшийся pod с Elasticsearch перешел в статус "Not Ready"
- Kibana потеряла подключение к кластеру
- Метрики Prometheus перестали собираться, так как у сервиса, к которому подключается exporter, пропали все endpoint

Сделаем вывод - узнавать о проблемах с Elasticsearch в нашем сценарии (**replication factor = 1: 1 shard + 1 replica** на индекс) желательно на этапе выхода из строя первой ноды в кластере.

Мониторинг Elasticsearch

В определении проблем с участниками кластера Elasticsearch может помочь следующий Prometheus alert ([источник](#)):

```
ALERT ElasticsearchTooFewNodesRunning
  IF elasticsearch_cluster_health_number_of_nodes < 3
  FOR 5m
  LABELS {severity="critical"}
  ANNOTATIONS {description="There are only {{$value}} < 3 Elasticsearch nodes
running", summary="ElasticSearch running on less than 3 nodes"}
```

Не забудьте вернуть ноды в строй (`kubectl uncordon`
`<NODE_NAME>`)

Мониторинг Elasticsearch

Рассмотрим некоторое количество ключевых метрик, которые рекомендуется отслеживать при эксплуатации Elasticsearch:

- **unassigned_shards** - количество shard, для которых не нашлось подходящей ноды, их наличие сигнализирует о проблемах
- **jvm_memory_usage** - высокая загрузка (в процентах от выделенной памяти) может привести к замедлению работы кластера
- **number_of_pending_tasks** - количество задач, ожидающих выполнения. Значение метрики, отличное от нуля, может сигнализировать о наличии проблем внутри кластера

Больше метрик с их описанием можно найти [здесь](#)

EFK | nginx ingress

Попробуем найти в Kibana логи nginx-ingress (например, полнотекстовым поиском по слову `nginx`) и обнаружим, что они отсутствуют.

Добейтесь того, чтобы эти логи появились

EFK | nginx ingress

После появления логов nginx у нас возникнет следующая проблема:

```
"_source": {  
  "log": "10.44.2.1 - - [11/Feb/2020:15:26:56 +0000] \"POST /elasticsearch  
/kubernetes_cluster-*/_search?rest_total_hits_as_int=true&ignore_unavailable=true&  
ignore_throttled=true&preference=1581433758087&timeout=30000ms HTTP/1.1\" 499 0  
\"http://kibana.35.184.239.20.xip.io/app/kibana\" \"Mozilla/5.0 (Macintosh; Intel  
Mac OS X 10_15_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.87  
Safari/537.36\" 1535 0.303 [observability-kibana-kibana-5601] [] 10.44.0.70:5601 0  
0.303 - 8de867470c348bd5598e554b5b36f8d9\\n",  
  ...  
}
```

EFK | nginx ingress

Сейчас лог представляет из себя строку, с которой сложно работать.

Мы можем использовать полнотекстовый поиск, но лишены возможности:

- Задействовать функции **KQL**
- Полноценно проводить аналитику
- Создавать Dashboard по логам
- ...

EFK | nginx ingress

Вспомним что у nginx есть возможность выбора формата и отформатируем лог в JSON

Отвечает за это ключ `log-format-escape-json`, добавьте его в `nginx-ingress.values.yaml`

Также не забудьте про опцию `log-format-upstream`

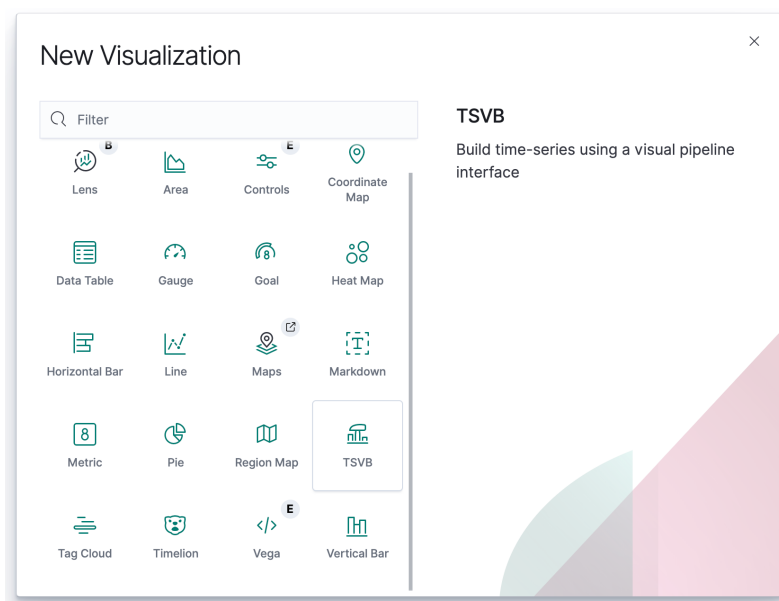
Если все сделано корректно, формат логов должен измениться на следующий:

```
"_source": {  
  "x-forward-for": "10.128.0.35",  
  "request_id": "bfcee33afe75c099f7887d7e70b1ab00",  
  "bytes_sent": 19087,  
  "request_time": 1.168,  
  "status": 200,  
  ...  
}
```

EFK | nginx ingress

Теперь, когда мы научились собирать логи с nginx-ingress и смогли их структурировать, можно опробовать возможности Kibana для визуализации.

Перейдите на вкладку **Visualize** и создайте новую визуализацию с типом **TSVB**



EFK | nginx ingress

Для начала, создадим визуализацию, показывающую общее количество запросов к nginx-ingress. Для этого нам понадобится применить следующий KQL фильтр:

```
kubernetes.labels.app : nginx-ingress
```

Добавьте данный фильтр в **Panel options** нашей визуализации:

The screenshot shows the 'Panel options' tab in a Kibana visualization configuration. The interface is divided into two main sections: 'Data' and 'Panel filter'.

Data section:

- Index pattern:** A text input field containing 'kubernetes_cluster-*'. Below it, a note states: 'Default index pattern is used. To query all indexes use *'.
- Time field:** A dropdown menu showing '@timestamp'.
- Interval:** A text input field containing 'auto'. Below it, examples are listed: 'Examples: auto, 1m, 1d, 7d, 1y, >=1m'.
- Drop last bucket?:** Radio buttons for 'Yes' (selected) and 'No'.

Panel filter section:

- Panel filter:** A text input field containing 'kubernetes.labels.app : nginx-ingress'.
- Ignore global filter?:** Radio buttons for 'Yes' and 'No' (selected).
- KQL:** A blue button with the text 'KQL' next to the filter input.

EFK | nginx ingress

Самостоятельно создайте визуализации для отображения запросов к nginx-ingress со статусами:

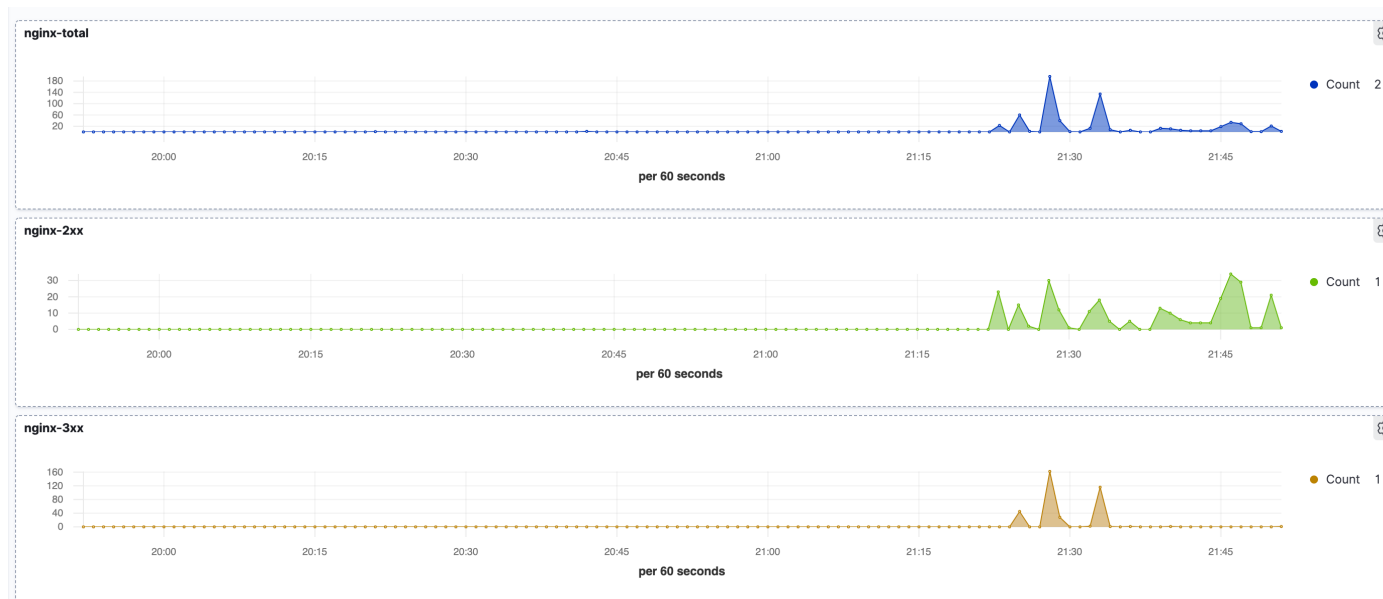
- 200-299
- 300-399
- 400-499
- 500+

EFK | nginx ingress

Пришло время объединить созданные нами визуализации в Dashboard.

Создайте Dashboard (соответствующая вкладка в меню) и добавьте на него свои визуализации.

Итоговый результат должен выглядеть примерно так:



EFK | nginx ingress

Экспортируйте получившиеся визуализации и Dashboard и добавьте файл `export.ndjson` в директорию `kubernetes-logging` вашего репозитория

Saved Objects

[Export 8 objects](#) [Import](#) [Refresh](#)

From here you can delete saved objects, such as saved searches. You can also edit the raw data of saved objects. Typically objects are only modified via their associated application, which is probably what you should use instead of this screen.

Search...		Type ▾	Delete	Export ▾
<input type="checkbox"/>	Type	Title	Actions	
<input type="checkbox"/>	⋮	Advanced Settings [7.5.2]	⋮	
<input checked="" type="checkbox"/>	📄	nginx-dashboard	⋮	
<input type="checkbox"/>	👤	kubernetes_cluster-*	⋮	
<input checked="" type="checkbox"/>	🏠	nginx-total	⋮	
<input checked="" type="checkbox"/>	🏠	nginx-4xx	⋮	
<input checked="" type="checkbox"/>	🏠	nginx-3xx	⋮	
<input checked="" type="checkbox"/>	🏠	nginx-2xx	⋮	
<input checked="" type="checkbox"/>	🏠	nginx-5xx	⋮	

Loki


Перейдем к еще одному решению для сбора и визуализации логов Kubernetes кластера

Задания:

- Установите Loki в namespace `observability`, используя любой из доступных способов. Должны быть установлены непосредственно Loki и Promtail
- Модифицируйте конфигурацию prometheus-operator таким образом, чтобы datasource **Loki** создавался сразу после установки оператора
- Итоговый файл `prometheus-operator.values.yaml` выложите в репозиторий в директорию `kubernetes-logging`

Loki | Datasource

Результат

 **Data Sources / Loki**
Type: Loki

Settings

This datasource was added by config and cannot be modified using the UI. Please contact your server admin to update this datasource.

Name

Loki

Default

☐

HTTP

URL

http://loki:3100/

Whitelisted Cookies

Add Name

Add

Auth

Basic auth

☐

With Credentials

☐

TLS Client Auth

☐

With CA Cert

☐

Skip TLS Verify

☐

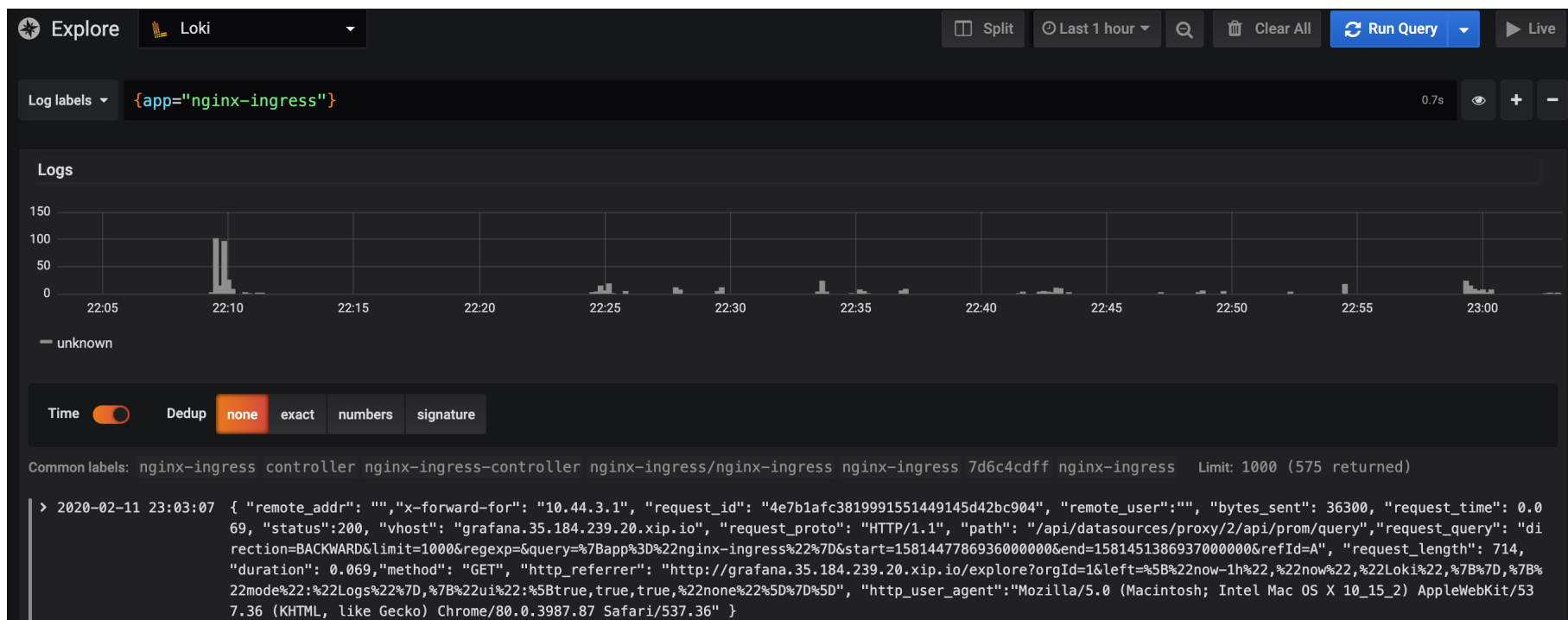
Forward OAuth Identity

☐

Maximum lines

1000

До недавнего времени, единственным способом визуализации логов в Loki была функция **Explore**. Например, логи nginx-ingress с ее помощью можно отобразить следующим образом:



Loki, аналогично Elasticsearch умеет разбирать JSON лог по ключам, но, к сожалению, фильтрация по данным ключам на текущий момент не работает:

Log Labels:			
...	Q	Q	app nginx-ingress
...	Q	Q	component controller
...	Q	Q	container_name nginx-ingress-controller
...	Q	Q	filename /var/log/pods/nginx-ingress/nginx-ingress-controller-7d6c4cdf-qcg7b_41d211b3-8569-49f6-893b-c00ce01bc5d1/nginx-ingress-controller/0.log
...	Q	Q	instance nginx-ingress-controller-7d6c4cdf-qcg7b
...	Q	Q	job nginx-ingress/nginx-ingress
...	Q	Q	namespace nginx-ingress
...	Q	Q	pod_template_hash 7d6c4cdf
...	Q	Q	release nginx-ingress
...	Q	Q	stream stdout
Parsed fields:			
...		ts	2020-02-11T20:09:51.110199227Z
...		remote_addr	""
...		x-forward-for	"10.128.0.35"
...		request_id	"1a09b97432ff9d3dcda108ae318692d3"
...		remote_user	""
...		bytes_sent	201
...		request_time	0.004
...		status	200

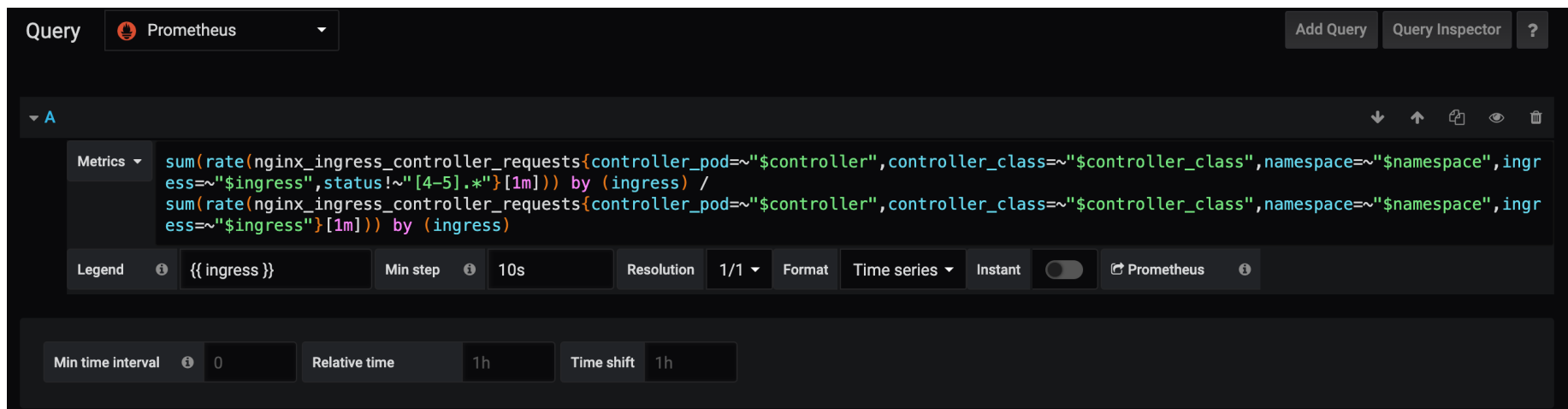
Создадим Dashboard, на котором одновременно выведем метрики nginx-ingress и его логи

- Убедитесь, что вместе с nginx-ingress устанавливается serviceMonitor, и Prometheus "видит" его
- Создайте в Grafana новый Dashboard
- Добавьте для него следующие **переменные** (взяты из **официального Dashboard** для nginx-ingress):

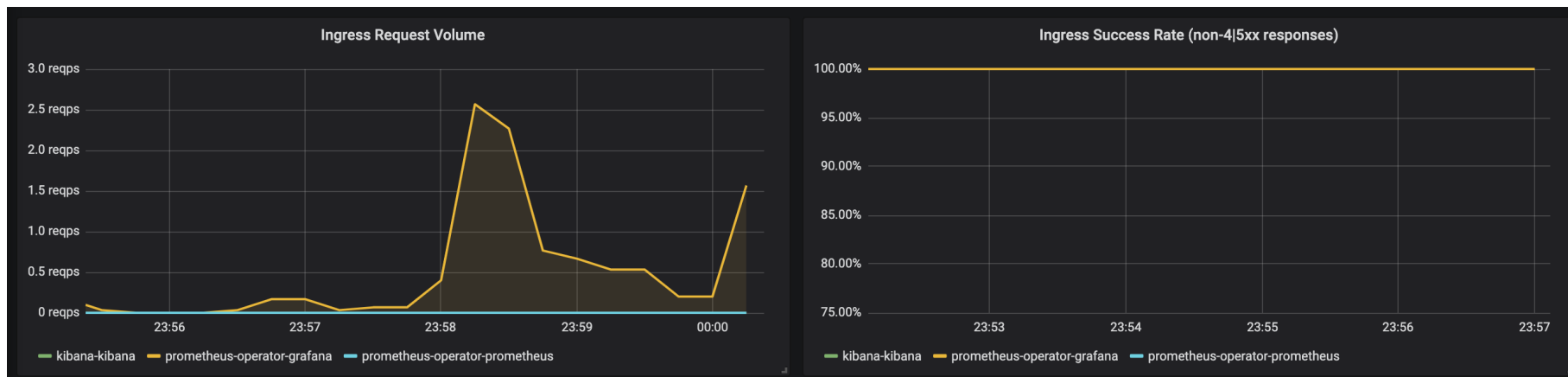
Variable	Definition		
<code>\$namespace</code>	<code>label_values/nginx_ingress_controller_config_hash, controller_namespace)</code>	↓	<button>Duplicate</button> <button>×</button>
<code>\$controller_class</code>	<code>label_values/nginx_ingress_controller_config_hash{namespace=~"\$namespace"}, controller_class)</code>	↑ ↓	<button>Duplicate</button> <button>×</button>
<code>\$controller</code>	<code>label_values/nginx_ingress_controller_config_hash{namespace=~"\$namespace",controller_class=~"\$controller_class"}, con...</code>	↑ ↓	<button>Duplicate</button> <button>×</button>
<code>\$ingress</code>	<code>label_values/nginx_ingress_controller_requests{namespace=~"\$namespace",controller_class=~"\$controller_class",controlle...</code>	↑	<button>Duplicate</button> <button>×</button>

Создайте новую панель и добавьте туда следующую [query](взято из официального Dashboard для nginx-ingress):

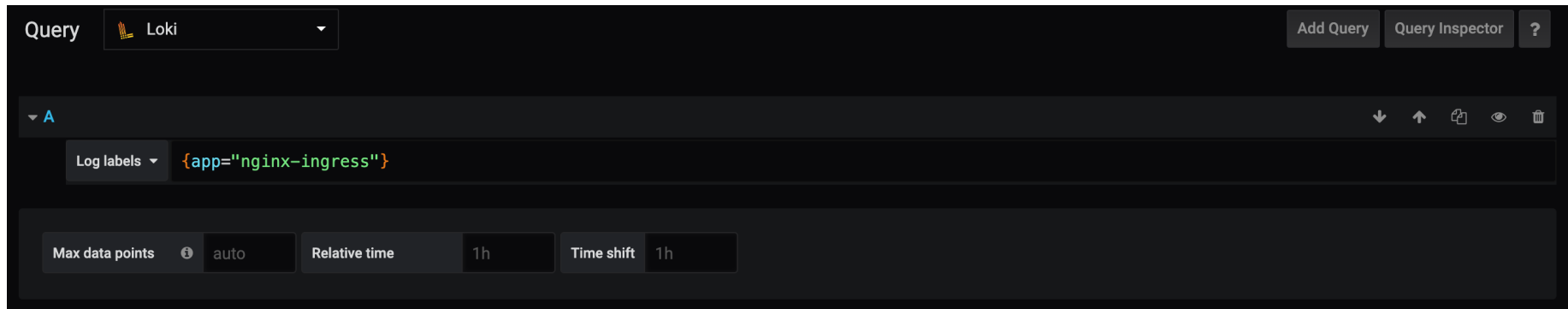
```
sum(rate/nginx_ingress_controller_requests{controller_pod=~"$controller",controller_class=~"$controller_class",namespace=~"$namespace",ingress=~"$ingress",status!~"[4-5].*"}[1m])) by (ingress) /  
sum(rate/nginx_ingress_controller_requests{controller_pod=~"$controller",controller_class=~"$controller_class",namespace=~"$namespace",ingress=~"$ingress"}[1m])) by (ingress)
```



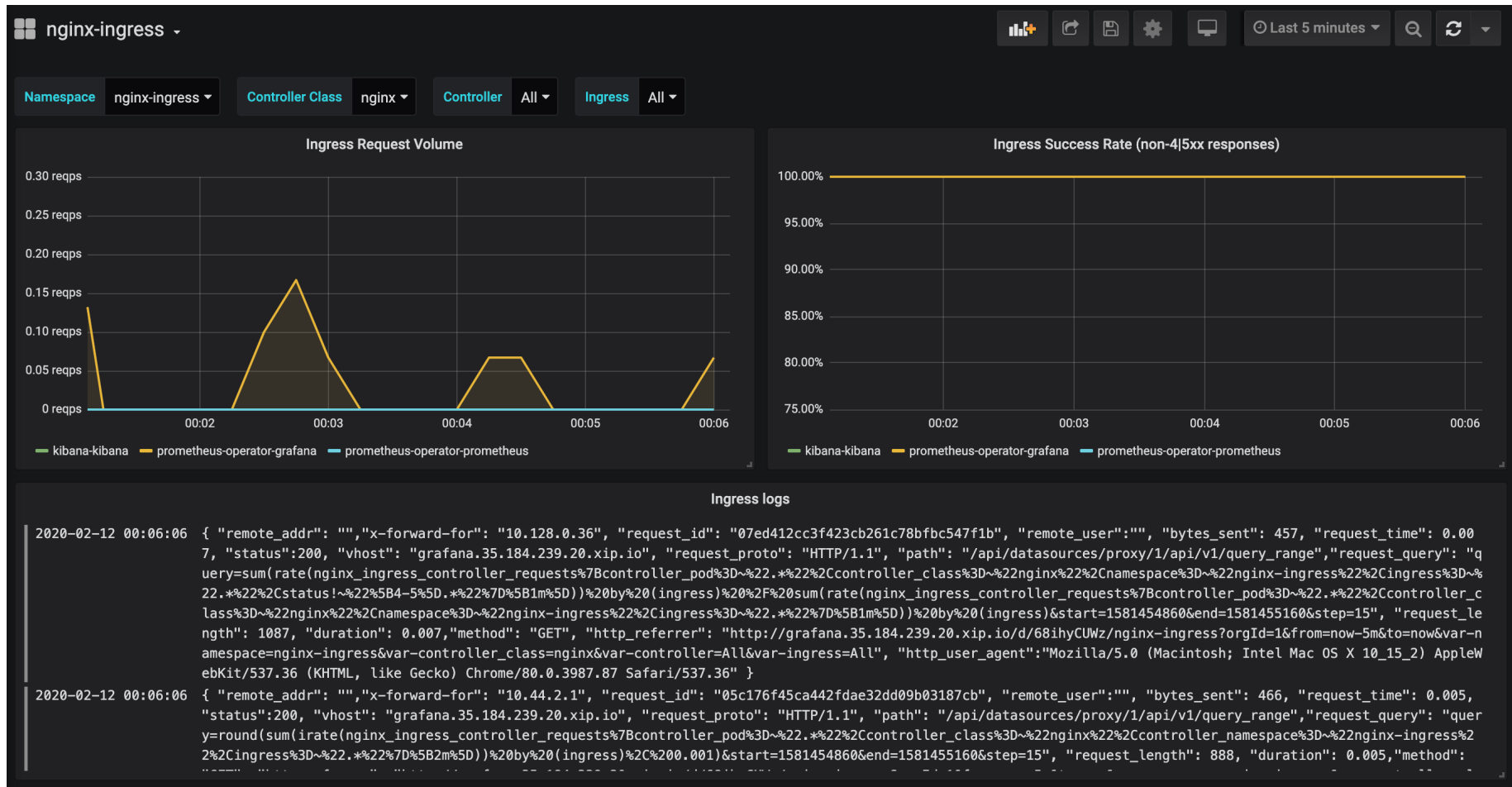
- Аналогичным образом добавьте панель, позволяющую оценить количество запросов к nginx-ingress в секунду
- Настройками в пункте **Visualization** приведите панели к следующему виду:



Добавим панель с логами и укажем для нее следующие настройки
Query:



Итоговый Dashboard должен выглядеть примерно так:



Loki | Визуализация

- Добавьте в Dashboard дополнительные панели с метриками, отслеживание которых кажется вам важным
- Выгрузите из Grafana JSON с финальным Dashboard и поместите его в файл `kubernetes-logging/nginx-ingress.json`


Хотим обратить внимание на еще одну небольшую, но очень полезную **утилиту**, позволяющую получить и сохранить event'ы Kubernetes в выбранном решении для логирования

Например, event, говорящий нам о том, что liveness probe у prometheus-operator выполнилась неуспешно

```
Parsed fields:
...|      ts                2020-02-11T21:27:52.019925745Z
...|      metadata          {"name":"prometheus-prometheus-operator-prometheus-0.15f2764847d2e553","namespace":"observability","selfLink":"/api/v1/namespaces/observability/
                              events/prometheus-prometheus-operator-prometheus-0.15f2764847d2e553","uid":"5497905b-4a45-4d5b-9a2e-f8501311bd93","resourceVersion":"3011","crea
                              tionTimestamp":"2020-02-11T21:27:51Z"}
...|      involvedObject    {"kind":"Pod","namespace":"observability","name":"prometheus-prometheus-operator-prometheus-0","uid":"505d7940-8589-44e0-9121-78d61f349d6b","api
                              Version":"v1","resourceVersion":"171888","fieldPath":"spec.containers{prometheus}"}
...|      reason            "Unhealthy"
...|      message            "Liveness probe failed: Get http://10.44.0.116:9090-/healthy: dial tcp 10.44.0.116:9090: connect: connection refused"
...|      source             {"component":"kubelet","host":"gke-logging-hw-default-pool-1c98687f-pfbn"}
...|      firstTimestamp     "2020-02-11T21:27:51Z"
...|      lastTimestamp      "2020-02-11T21:27:51Z"
...|      count              1
...|      type                "Warning"
...|      eventTime           null
...|      reportingComponent  ""
...|      reportingInstance   ""
```

Audit logging | Задание со

Еще один важный тип логов, который рекомендуется собирать и хранить - логи **аудита**

- Получить их в YC сложнее, чем в self-hosted кластерах из-за того, что доступ к master нодам, на которых запущен kube-apiserver, отсутствует
- Поэтому, в качестве задания со , предлагаем вам любым удобным образом развернуть self-hosted кластер и настроить сбор аудит логов
- Опишите результат в PR и выложите получившиеся в процессе выполнения задания артефакты в директорию `kubernetes-logging` вашего репозитория

Host logging | Задание со

- На текущий момент мы лишены возможности централизованного просмотра логов с виртуальных машин, на которых запущен Kubernetes
- Модернизируйте конфигурацию fluent-bit таким образом, чтобы данные логи отображались в Elasticsearch или реализуйте это при помощи другого решения для логирования (Fluentd, etc...)
- Опишите результат в PR и выложите получившиеся в процессе выполнения задания артефакты в директорию `kubernetes-logging` вашего репозитория

Проверка ДЗ

- Результаты вашей работы должны быть добавлены в ветку **kubernetes-logging** вашего GitHub репозитория `<YOUR_LOGIN>_platform`
- В **README.md** рекомендуется внести описание того, что сделано
- Создайте Pull Request к ветке **master** (описание PR рекомендуется заполнять)
- Добавьте метку `kubernetes-logging` к вашему PR

Проверка ДЗ

Данное задание будет проверяться в полуавтоматическом режиме. Не пугайтесь того, что тесты в итоге завершатся неуспешно.

При этом смотрите в лог **Travis**, чтобы понять, действительно ли они дошли до "правильной ошибки", говорящей о том, что дальнейшая проверка будет производиться вручную.