

Введение в мониторинг. Системы мониторинга.

План

- Prometheus: запуск, конфигурация, знакомство с Web UI
- Мониторинг состояния микросервисов
- Сбор метрик хоста с использованием экспортера

Развитие проекта microservices

В прошлых ДЗ вы создали репозиторий microservices на GitHub. Убедитесь что данный проект находится у вас на локальной машине.

Если у вас нет репозитория microservices на GitHub, выполните сначала предыдущие ДЗ.



Проверка ДЗ

Создайте новую ветку в вашем локальном репозитории для выполнения данного ДЗ. Т.к. это первое задание по мониторингу, то ветку можно назвать **monitoring-1**.

Проверка данного ДЗ, как и многих последующих, будет производиться через Pull Request ветки с ДЗ к ветке мастер и добавлению в Reviewers пользователей **Artemmkin** и **chromko**.

После того, как **один** из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить.



Подготовка окружения

Создадим правило фаервола для Прометея и Puma:

```
$ gcloud compute firewall-rules create prometheus-default --allow tcp:9090
$ gcloud compute firewall-rules create puma-default --allow tcp:9292
```

Создадим Docker хост в GCE и настроим локальное окружение на работу с ним (ссылка на [gist](#)):

```
$ docker-machine create --driver google \
  --google-project infra-179031 \
  --google-machine-image https://www.googleapis.com/compute/v1/projects/
ubuntu-os-cloud/global/images/family/ubuntu-1604-lts \
  --google-machine-type n1-standard-1 \
  --google-zone europe-west1-b \
  vm1
```

```
...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this
virtual machine, run: docker-machine env vm1
```

```
$ eval $(docker-machine env vm1)
```

Запуск Прометея

Систему мониторинга Прометей будем запускать внутри Docker контейнера. Для начального знакомства воспользуемся готовым образом с DockerHub.

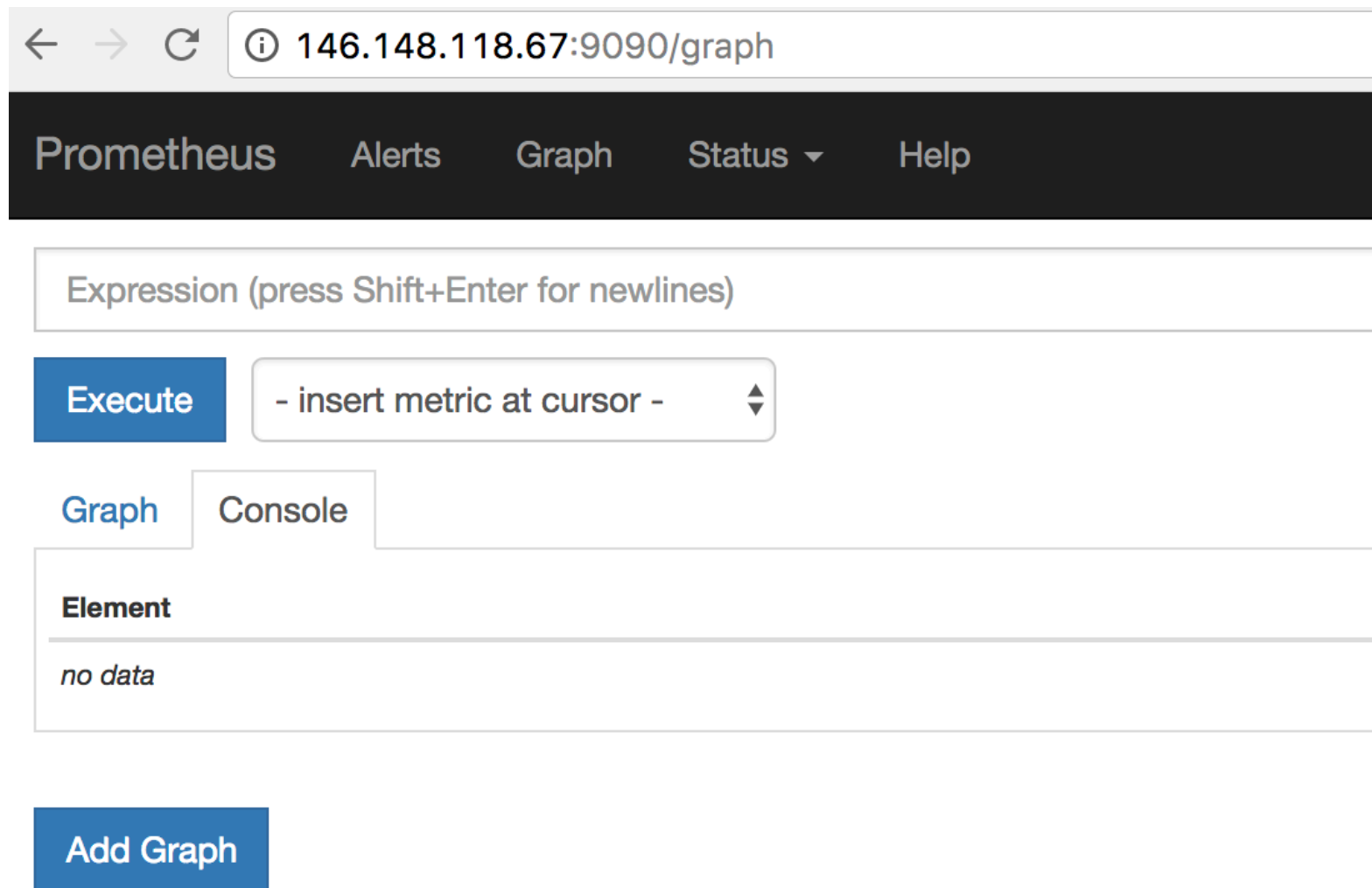
```
$ docker run --rm -p 9090:9090 -d --name prometheus prom/prometheus
```

```
e6c684c9b9c9ae1071fafc4adfa75d2c96f683f2489f05a4d8662c531054ab45
```

```
$ docker ps
```

CONTAINER ID	IMAGE	STATUS	PORTS	NAMES
e6c684c9b9c9	prom/prometheus	Up 4 seconds	0.0.0.0:9090->9090/tcp	prometheus

Откроем веб интерфейс

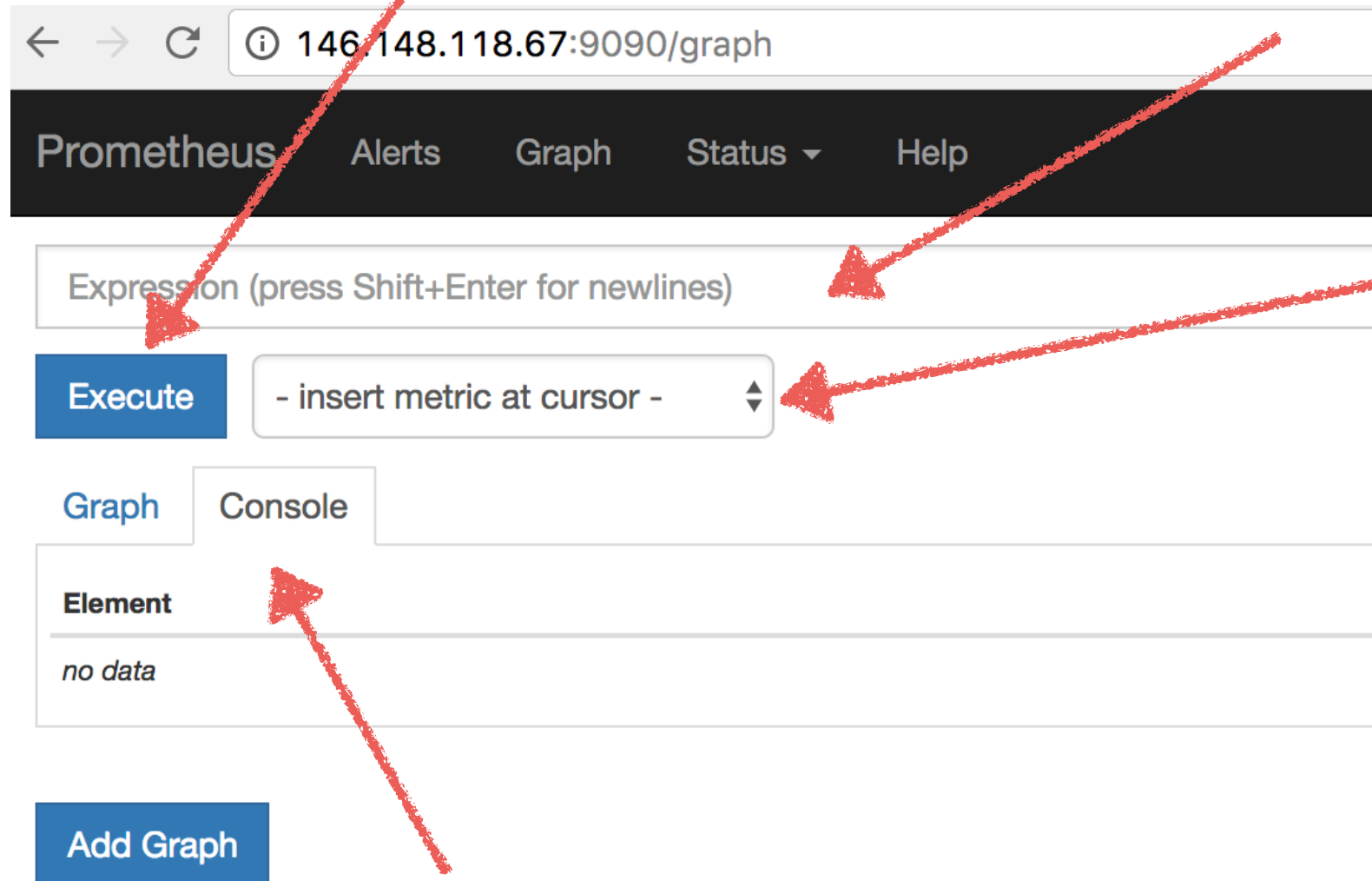


P.S. по умолчанию сервер слушает порт 9090, IP адрес созданной VM можно узнать, используя команду:

```
$ docker-machine ip vm1
```

Выполнение запроса

Строка ввода выражений для получения и анализа информации мониторинга (метрик) из хранилища



Выбор имеющихся метрик

Вкладка Console, которая сейчас активирована, выводит численное значение выражений. Вкладка Graph, левее от нее, строит график изменений значений метрик со временем

Если нажмем на "insert metric at cursor", то увидим, что Прометей уже собирает какие-то метрики. По умолчанию он собирает статистику о своей работе. Выберем, например, метрику prometheus_build_info и нажмем Execute, чтобы посмотреть информацию о версии.

[Prometheus](#) [Alerts](#) [Graph](#) [Status ▾](#) [Help](#)

Execute

Graph

Console

Element

```
prometheus_build_info{branch="HEAD",goversion="go1.9.1",instance="localhost:9090",job="prometheus",revision="3a7c51ab70fc761"}
```

Add Graph

Поясним результат вывода

```
prometheus_build_info{branch="HEAD",goversion="go1.9.1",instance="localhost:9090",job="prometheus",revision="3a7c51ab70fc7615cd318204d3aa7c078b7c5b20",version="1.8.1"} 1
```

название метрики - идентификатор собранной информации.

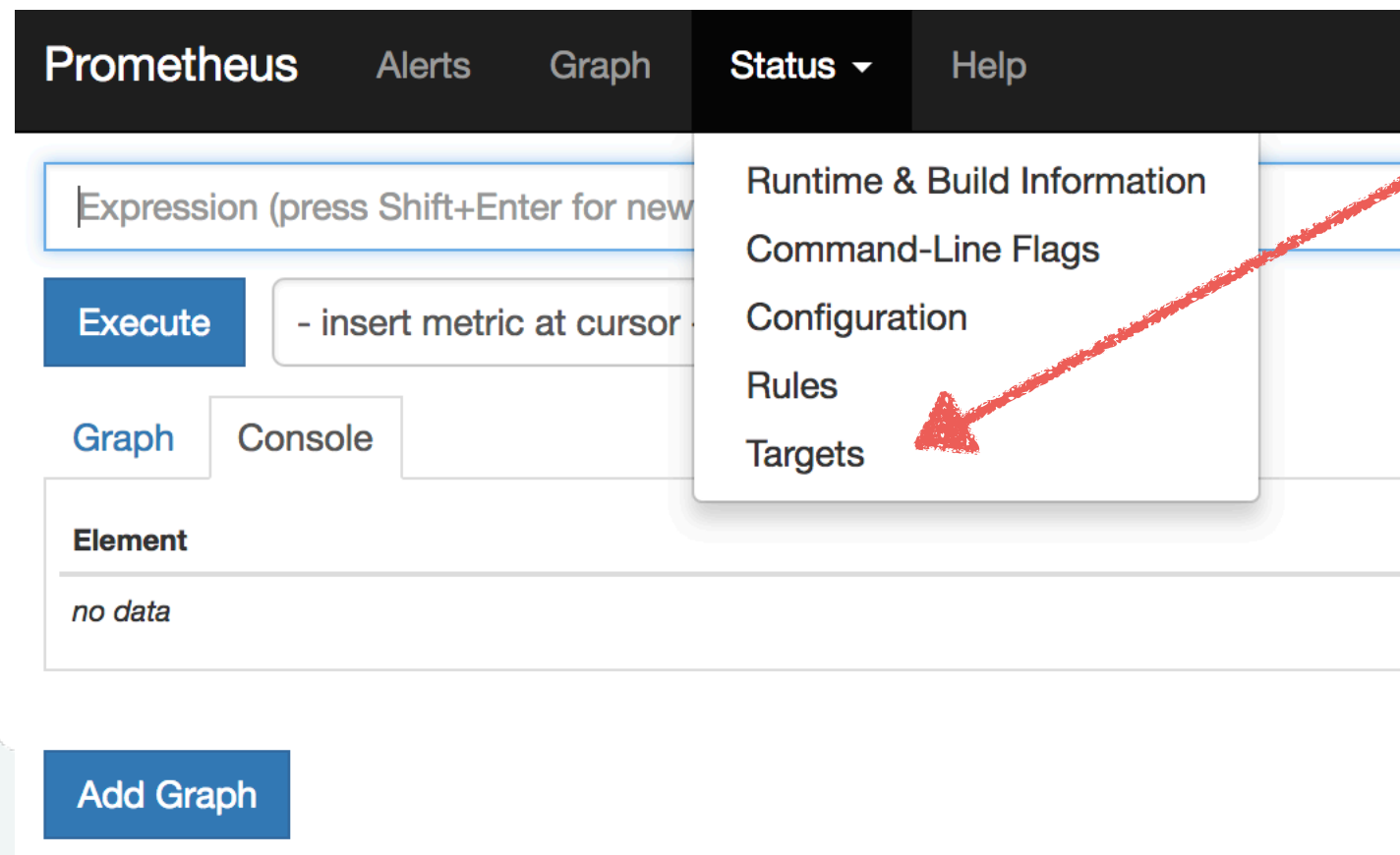
лейбл - добавляет метаданных метрике, уточняет ее.

Использование лейблов дает нам возможность не ограничиваться лишь одним названием метрик для идентификации получаемой информации. Лейблы содержатся в {} скобках и представлены наборами "ключ=значение".

значение метрики

Targets

Targets (цели) - представляют собой системы или процессы, за которыми следит Прометей. Помним, что Прометей является pull системой, поэтому он постоянно делает HTTP запросы на имеющиеся у него адреса (endpoints). Посмотрим текущий список целей



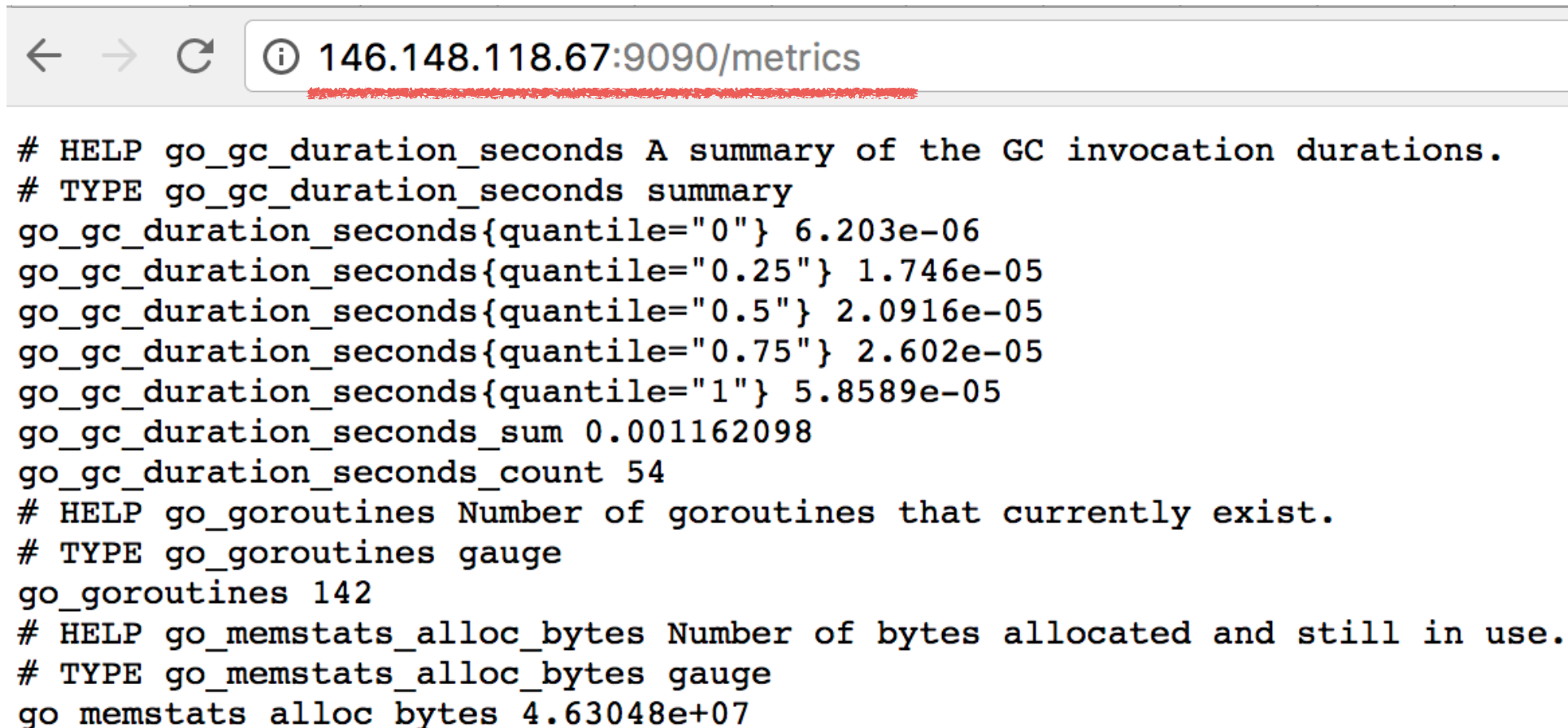
В Targets мы можем сейчас увидеть только сам Прометей. У каждой цели есть свой список адресов (endpoints), по которым следует обращаться для получения информации. В веб интерфейсе мы можем видеть состояние каждого endpoint-а (up); лейбл (instance="someURL"), который прометей автоматически добавляет к каждой метрике, получаемой с данного endpoint-а; а так же время, прошедшее с момента последней операции сбора информации с endpoint-а.

Targets

prometheus (1/1 up)			
Endpoint	State	Labels	Last Scrape
http://localhost:9090/metrics	UP	instance="localhost:9090"	3.967s ago

Обратите внимание на endpoint, который мы с вами видели на предыдущем слайде.

Мы можем открыть страницу в веб браузере по данному HTTP пути (host:port/metrics), чтобы посмотреть, как выглядит та информация, которую собирает Прометей.



```
← → ↻ ⓘ 146.148.118.67:9090/metrics

# HELP go_gc_duration_seconds A summary of the GC invocation durations.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 6.203e-06
go_gc_duration_seconds{quantile="0.25"} 1.746e-05
go_gc_duration_seconds{quantile="0.5"} 2.0916e-05
go_gc_duration_seconds{quantile="0.75"} 2.602e-05
go_gc_duration_seconds{quantile="1"} 5.8589e-05
go_gc_duration_seconds_sum 0.001162098
go_gc_duration_seconds_count 54
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 142
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 4.63048e+07
```

Остановим контейнер:

```
$ docker stop prometheus
```

Создание Docker образа

Познакомившись с веб интерфейсом Прометея и его стандартной конфигурацией, соберем на основе готового образа с DockerHub свой Docker образ с конфигурацией для мониторинга наших микросервисов.

Создайте в корне вашего проекта директорию prometheus. Затем в этой директории создайте простой Dockerfile, который будет копировать файл конфигурации с нашей машины внутрь контейнера:

prometheus/Dockerfile

```
FROM prom/prometheus
```

```
ADD prometheus.yml /etc/prometheus/
```

Конфигурация

Вся конфигурация Прометея, в отличие от многих других систем мониторинга, происходит через файл конфигурации и опции командной строки.

Мы определим простой конфигурационный файл для сбора метрик с наших микросервисов. В директории `prometheus` создайте файл `prometheus.yml` со следующим содержимым (см. след. слайд)

prometheus.yml

(ссылка на [gist](#))

```
global:  
  scrape_interval: '5s'
```

```
scrape_configs:  
  - job_name: 'prometheus'  
    static_configs:  
      - targets:  
        - 'localhost:9090'
```

С какой частотой
собирать метрики

```
  - job_name: 'ui'  
    static_configs:  
      - targets:  
        - 'ui:9292'
```

Адреса для сбора
метрик (endpoints)

```
  - job_name: 'comment'  
    static_configs:  
      - targets:  
        - 'comment:9292'
```

В джобы объединяют в
группы endpoint-ы,
выполняющие
одинаковую функцию

Создаем образ

В директории prometheus собираем Docker образ:

```
$ export USER_NAME=artemmkina  
$ docker build -t $USER_NAME/prometheus .
```

Где USER_NAME - ваш логин от DockerHub.

В конце занятия можно будет запустить на DockerHub собранные вами на этом занятии образы.

Образы микросервисов

Код микросервисов обновился, мы добавили туда healthcheck-и для проверки работоспособности нашего приложения. Вам потребуется скопировать содержимое директорий микросервисов (ui, post-ru и comment) в ваш репозиторий microservices. При этом Dockerfile-ы для сборки образов сервисов лучше оставить свои.

Сборку образов теперь необходимо производить при помощи скриптов docker_build.sh, которые появились в каждой директории сервиса. С его помощью мы будем использовать информацию Git репозитория в нашем healthcheck-е.

Скопируйте файлы директорий микросервисов из ветки `microservices` репозитория reddit в ваш репозиторий на гитхабе. Выполните сборку образов при помощи скриптов `docker_build.sh`. Обратите внимание, на установку пакетов для post сервиса при сборке образа и поправьте ваш Dockerfile.

```
/ui (microservices ✓) $ bash docker_build.sh
```

```
/post-py (microservices ✓) $ bash docker_build.sh
```

```
/comment (microservices ✓) $ bash docker_build.sh
```

docker-compose.yml

Будем поднимать наш Прометей совместно с микросервисами.
Определите в вашем docker-compose.yml файле, сервис Прометея.

```
services:  
  ...  
  
  prometheus:  
    image: ${USER_NAME}/prometheus  
    ports:  
      - '9090:9090'
```

Отметим, что сборка Docker образов в данный момент производится нами вручную через скрипт docker_build.sh (мы предполагаем, что сборку у нас будет производить CI сервер). Поэтому удалите build директивы из docker-compose.yml и используйте директиву image.

docker-compose.yml

Мы будем использовать Прометей для мониторинга всех наших микросервисов, поэтому нам необходимо, чтобы контейнер с Прометеем мог общаться по сети со всеми другими сервисами, определенными в компоуз файле. Самостоятельно добавьте секцию `networks` в определение сервиса Прометея в `docker-compose.yml`.

Запуск микросервисов

Поднимем сервисы, определенные в docker-compose.yml

```
$ docker-compose up -d
```

Проверьте, что у вас работает приложение и запустился Прометей.

The image shows two browser windows. The left window, at address 146.148.118.67:9292, displays 'Microservices Reddit' with a green success message 'Post successfully published' and a status 'The Reddit App Is Working!' with a counter at 0. The right window, at address 146.148.118.67:9090/graph, shows the Prometheus web interface with the 'Graph' tab selected. The expression input field is empty, and the graph area shows 'no data'.

Мониторинг состояния микросервисов

Список endpoint-ов

Посмотрим список endpoint-ов, с которых собирает информацию Прометей. Помните, что помимо самого Прометея, мы определили в конфигурации мониторинг post и comment сервисов. Endpoint-ы должны быть в состоянии **UP**.

comment (1/1 up)		
Endpoint	State	Labels
http://comment:9292/metrics	UP	instance="comment:9292"
prometheus (1/1 up)		
Endpoint	State	Labels
http://localhost:9090/metrics	UP	instance="localhost:9090"
ui (1/1 up)		
Endpoint	State	Labels
http://ui:9292/metrics	UP	instance="ui:9292"

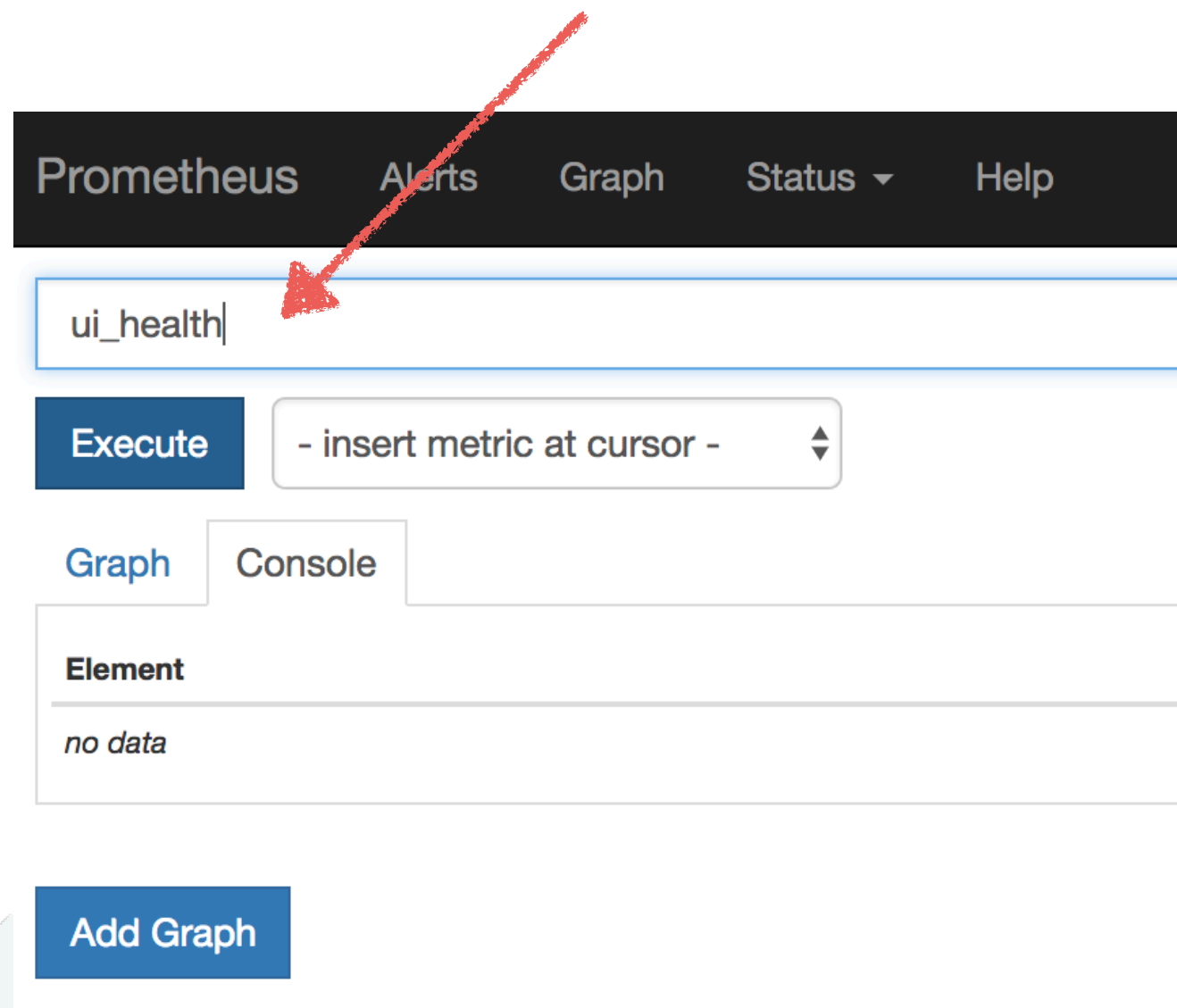
Healthchecks

Healthcheck-и представляют собой проверки того, что наш сервис здоров и работает в ожидаемом режиме.

В нашем случае healthcheck выполняется внутри кода микросервиса и выполняет проверку того, что все сервисы, от которых зависит его работа, ему доступны. Если требуемые для его работы сервисы здоровы, то healthcheck проверка возвращает `status = 1`, что соответствует тому, что сам сервис здоров. Если один из нужных ему сервисов нездоров или недоступен, то проверка вернет `status = 0`.

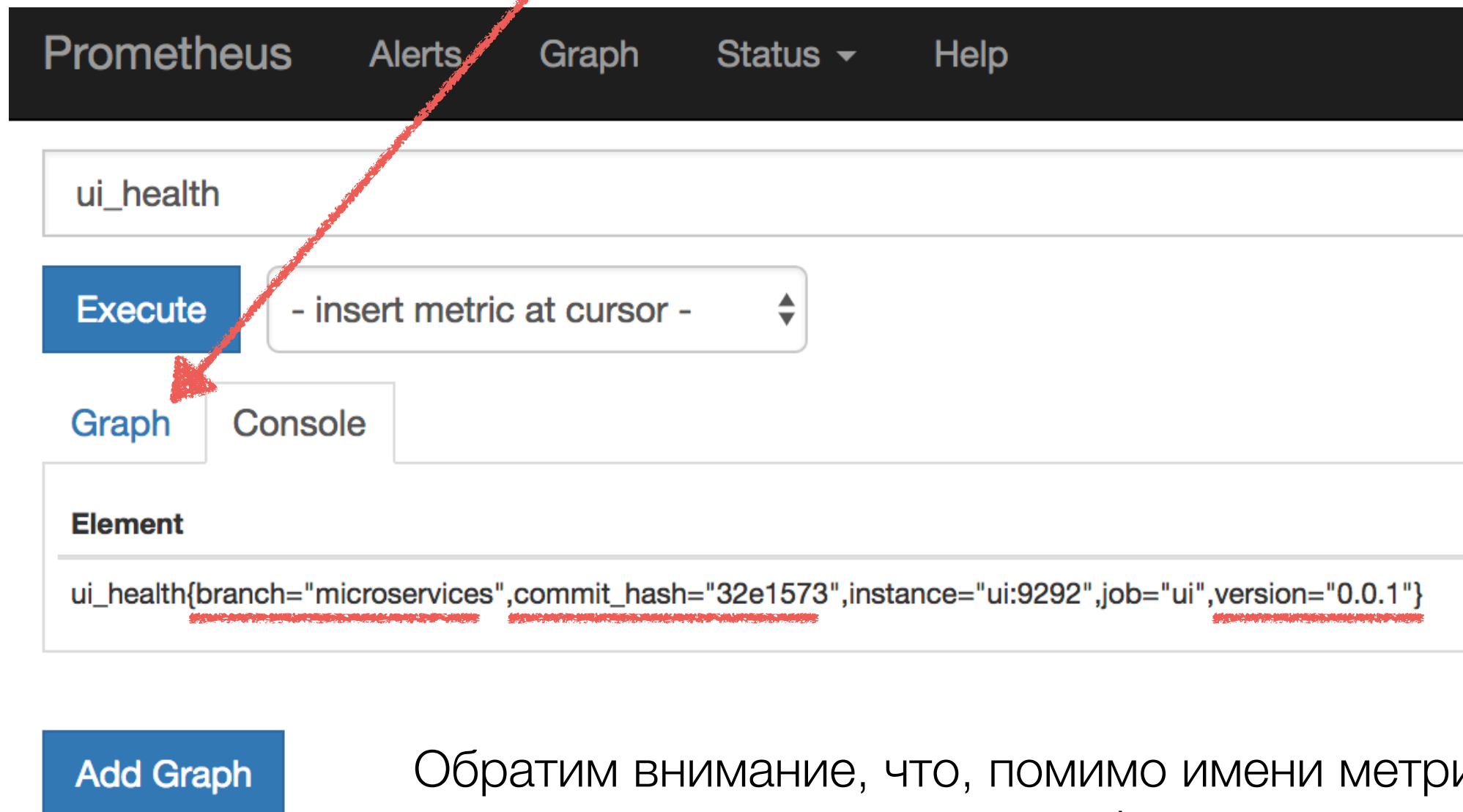
Состояние сервиса UI

В веб интерфейсе Прометея выполните поиск по названию метрики ui_health



The screenshot shows the Prometheus web interface. At the top is a navigation bar with links: Prometheus, Alerts, Graph, Status (with a dropdown arrow), and Help. Below this is a search bar containing the text 'ui_health'. A red arrow points from the text 'В веб интерфейсе Прометея выполните поиск по названию метрики ui_health' to the search bar. Below the search bar is a blue 'Execute' button and a dropdown menu showing '- insert metric at cursor -'. Underneath are two tabs: 'Graph' (selected) and 'Console'. The main content area under the 'Graph' tab shows 'Element' and 'no data'. At the bottom left is a blue 'Add Graph' button.

Построим график того, как менялось значение метрики `ui_health` со временем



Prometheus Alerts Graph Status ▾ Help

ui_health

Execute - insert metric at cursor -

Graph Console

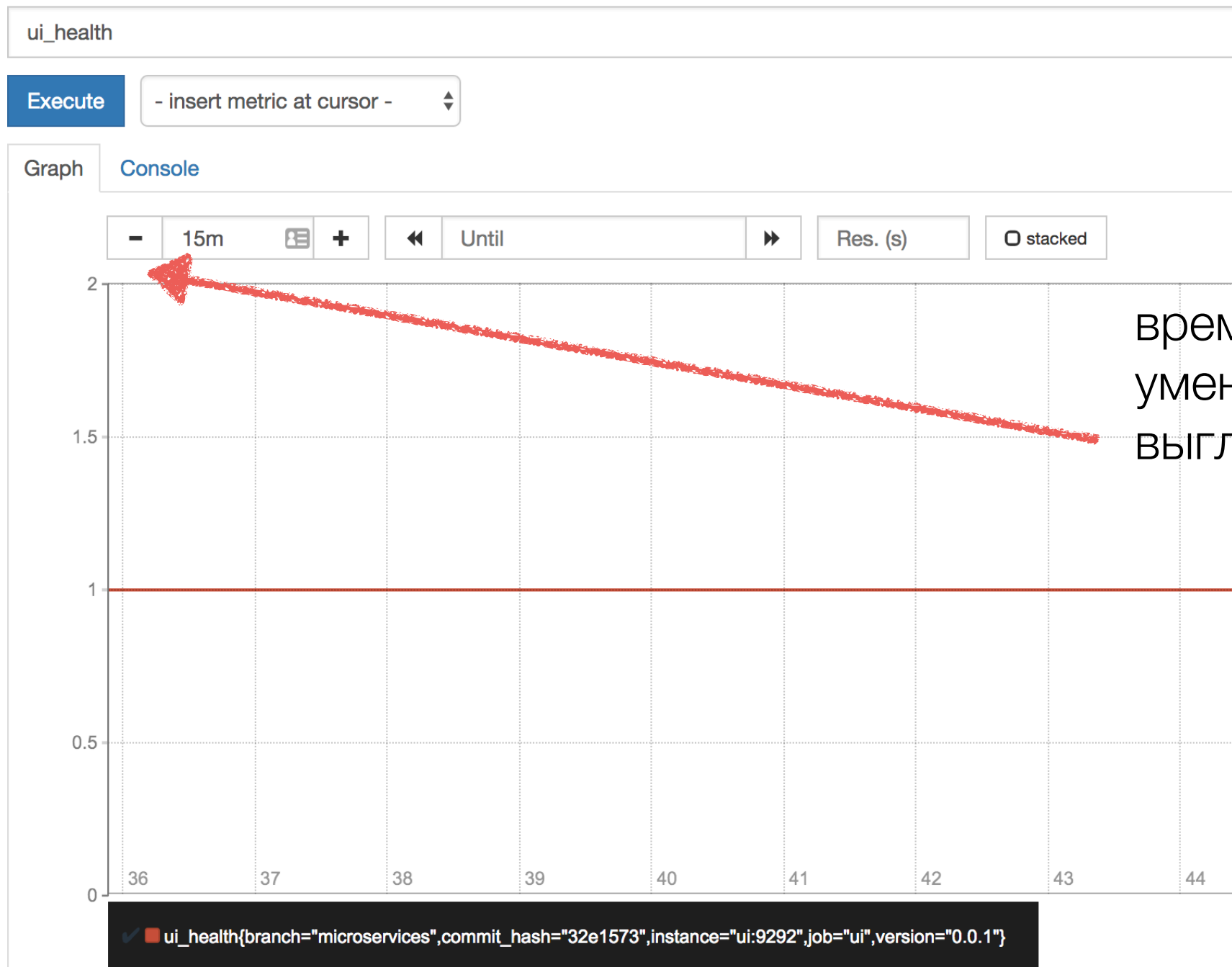
Element

`ui_health{branch="microservices",commit_hash="32e1573",instance="ui:9292",job="ui",version="0.0.1"}`

Add Graph

Обратим внимание, что, помимо имени метрики и ее значения, мы также видим информацию в лейблах о версии приложения, комите и ветке кода в Git-е.

Видим, что статус UI сервиса был стабильно 1, что означает, что сервис работал. Данный график оставьте открытым.



временной промежуток можно уменьшить, чтобы график выглядел лучше

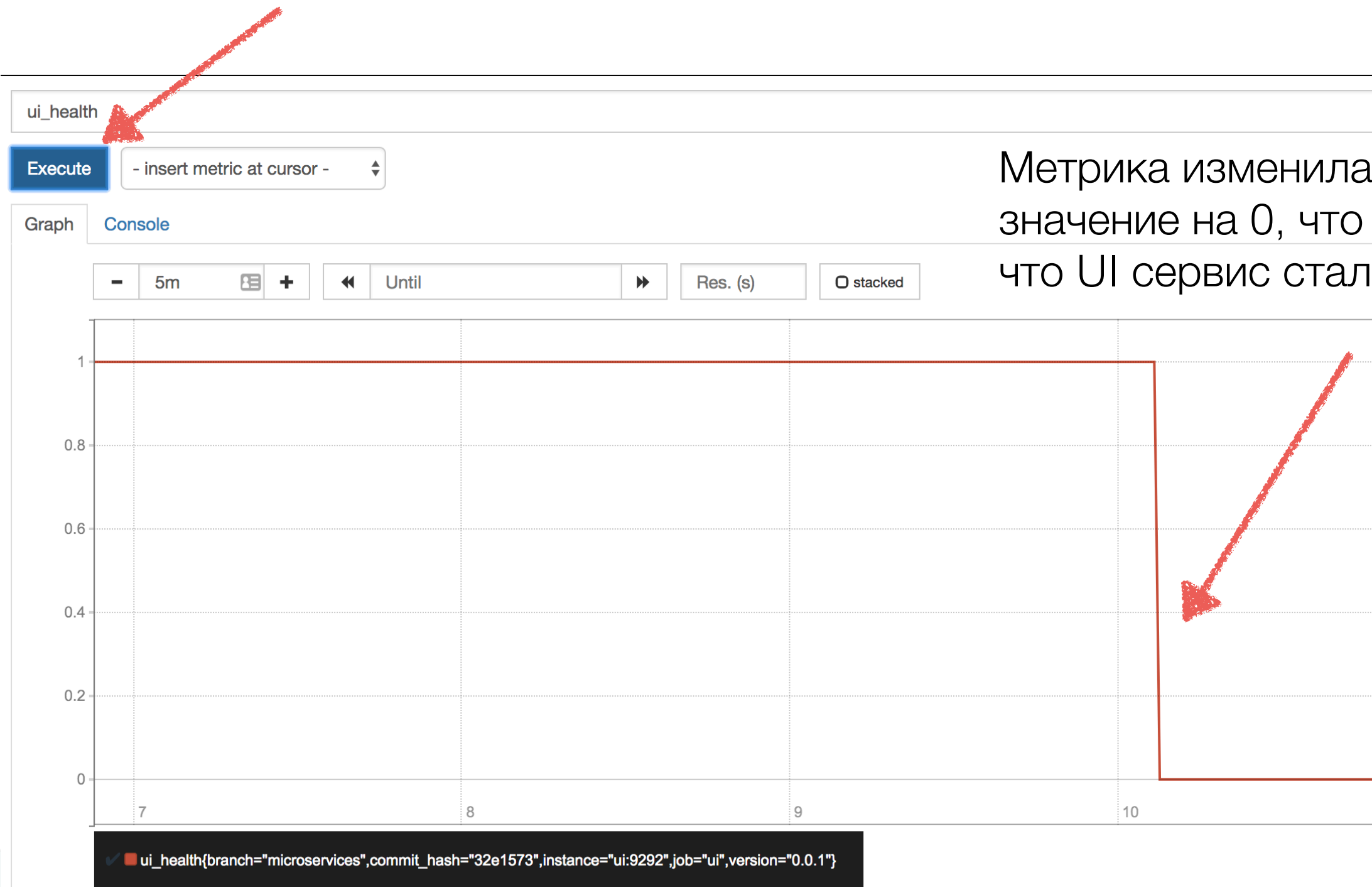
Остановим post сервис

Мы говорили, что условились считать сервис здоровым, если все сервисы, от которых он зависит также являются здоровыми. Попробуем остановить сервис post на некоторое время и пронаблюдаем, как изменится статус ui сервиса, который зависим от post.

```
$ docker-compose stop post
```

```
Stopping starthealthchecks_post_1 ... done
```

Обновим наш график



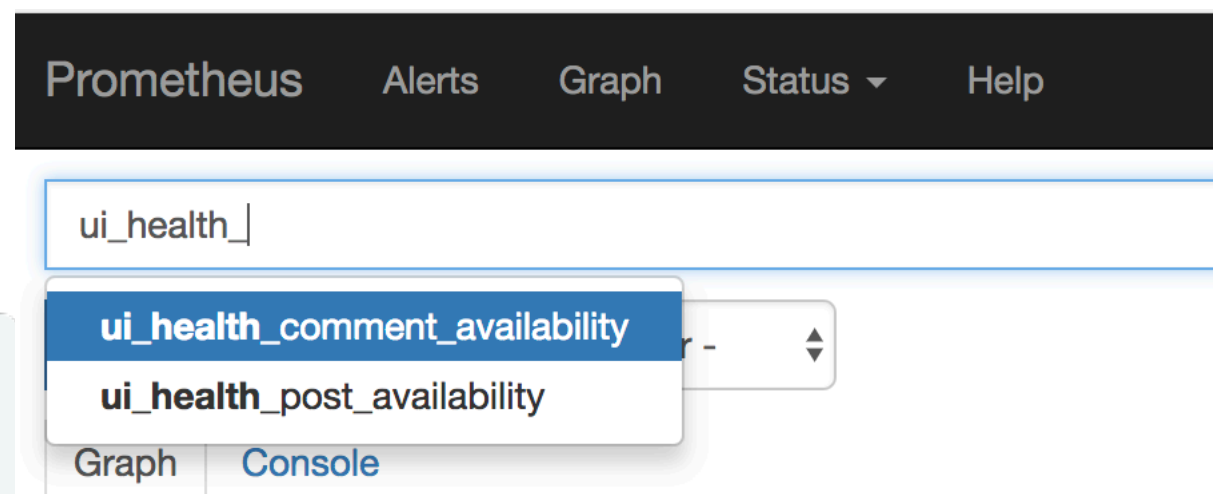
Метрика изменила свое значение на 0, что означает, что UI сервис стал нездоров

Поиск проблемы

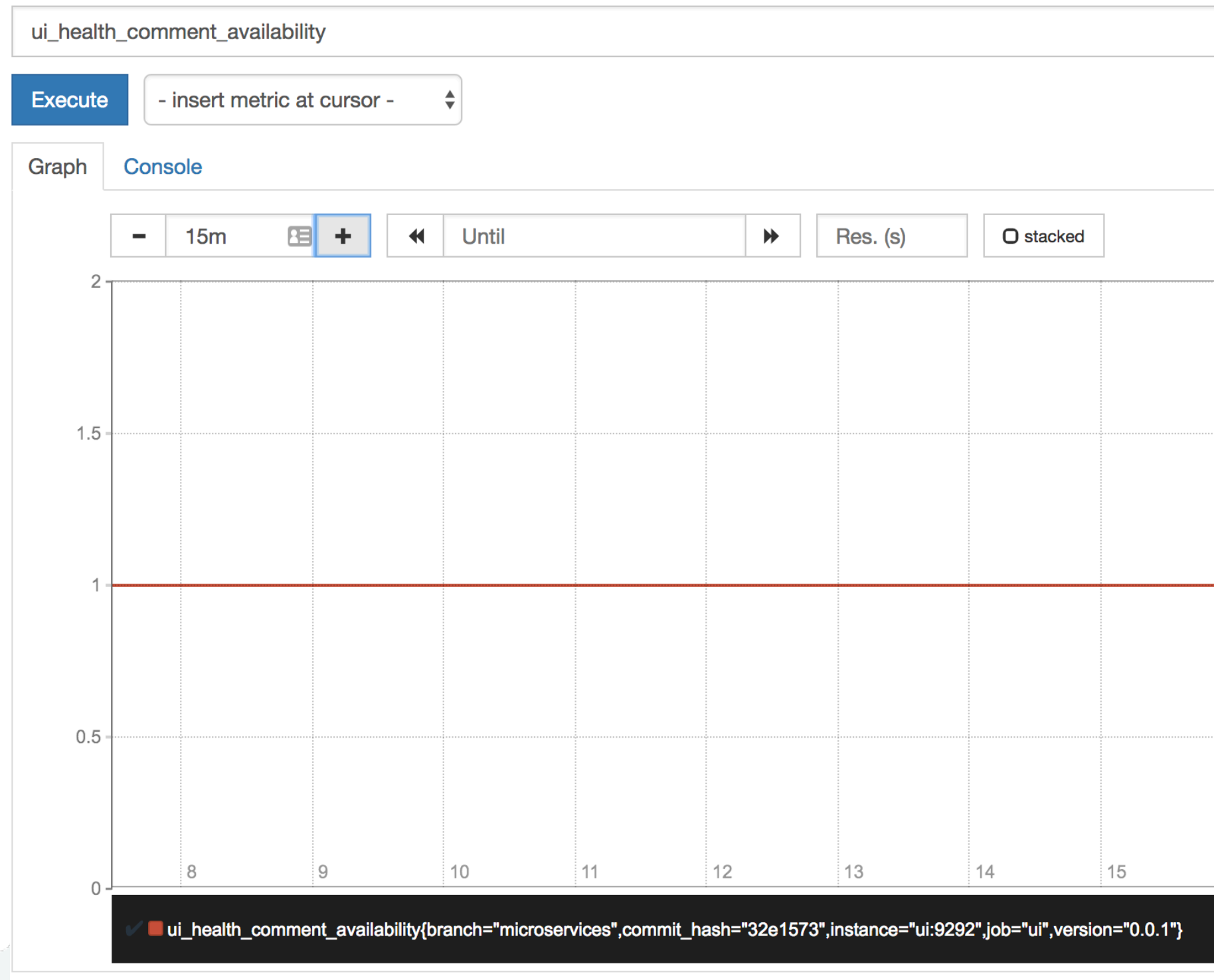
Помимо статуса сервиса, мы также собираем статусы сервисов, от которых он зависит. Названия метрик, значения которых соответствует данным статусам, имеет формат `ui_health_<service-name>`.

Посмотрим, не случилось ли чего плохого с сервисами, от которых зависит UI сервис.

Наберем в строке выражений `ui_health_` и Прометей нам предложит дополнить названия метрик.



Проверим comment сервис. Видим, что сервис свой статус не менял в данный промежуток времени



А с post сервисом все плохо



ЧИНИМ

Проблему мы обнаружили и знаем, как ее поправить (ведь мы же ее и создали :)). Поднимем post сервис.

```
$ docker-compose start post
```

```
Starting post ... done
```

Post сервис поправился

ui_health_post_availability

Execute

- insert metric at cursor -

Graph

Console

-

1m

+

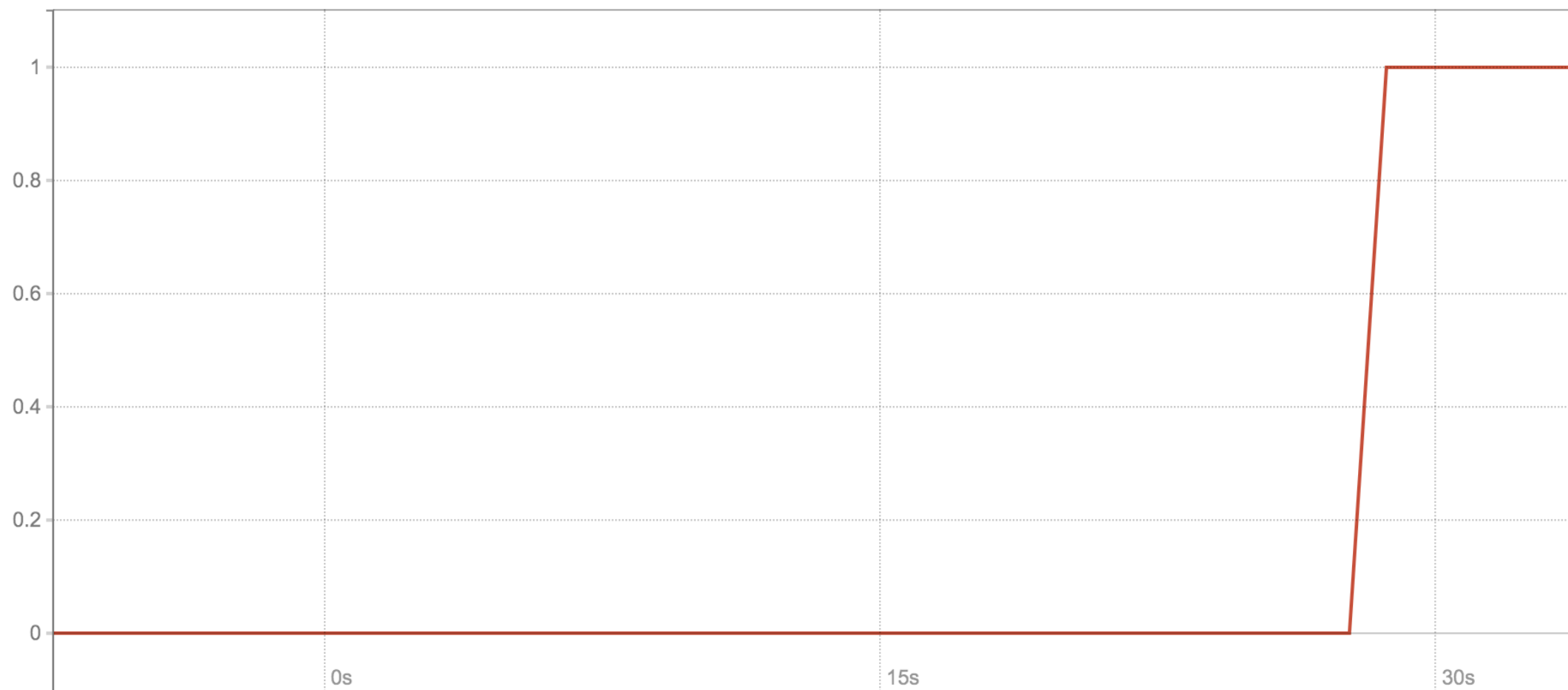
◀

Until

▶

Res. (s)

☐ stacked



✓ ui_health_post_availability{branch="microservices",commit_hash="32e1573",instance="ui:9292",job="ui",version="0.0.1"}

UI сервис тоже



При желании, можно попробовать уронить comment сервис или БД для какого-то из сервисов и провести аналогичные операции по мониторингу ситуации.

Сбор метрик хоста

Exporters

Экспортер похож на вспомогательного агента для сбора метрик. В ситуациях, когда мы не можем реализовать отдачу метрик Прометею в коде приложения, мы можем использовать экспортер, который будет транслировать метрики приложения или системы в формате доступном для чтения Прометеем.

Node exporter

Используем Node экспортер для сбора информации о работе Docker хоста (виртуалки, где у нас запущены контейнеры) и предоставлении этой информации Прометею.

docker-compose.yml

Node экспортер будем запускать также в контейнере.

Определим еще один сервис в docker-compose.yml файле.

Не забудьте также добавить определение сетей для сервиса node-exporter, чтобы обеспечить доступ Прометея к экспортеру.

(ссылка на [gist](#))

```
services:
```

```
...
```

```
node-exporter:
```

```
image: prom/node-exporter:v0.15.0
```

```
user: root
```

```
volumes:
```

- /proc:/host/proc:ro
- /sys:/host/sys:ro
- /:/rootfs:ro

```
command:
```

- '--path.procfs=/host/proc'
- '--path.sysfs=/host/sys'
- '--collector.filesystem.ignored-mount-points="^/(sys|proc|dev|host|etc)(\$\$|/)"'

prometheus.yml

Чтобы сказать Прометею следить за еще одним сервисом, нам нужно добавить информацию о нем в конфиг Прометея. Добавим еще один job:

```
scrape_configs:  
  ...  
  - job_name: 'node'  
    static_configs:  
      - targets:  
        - 'node-exporter:9100'
```

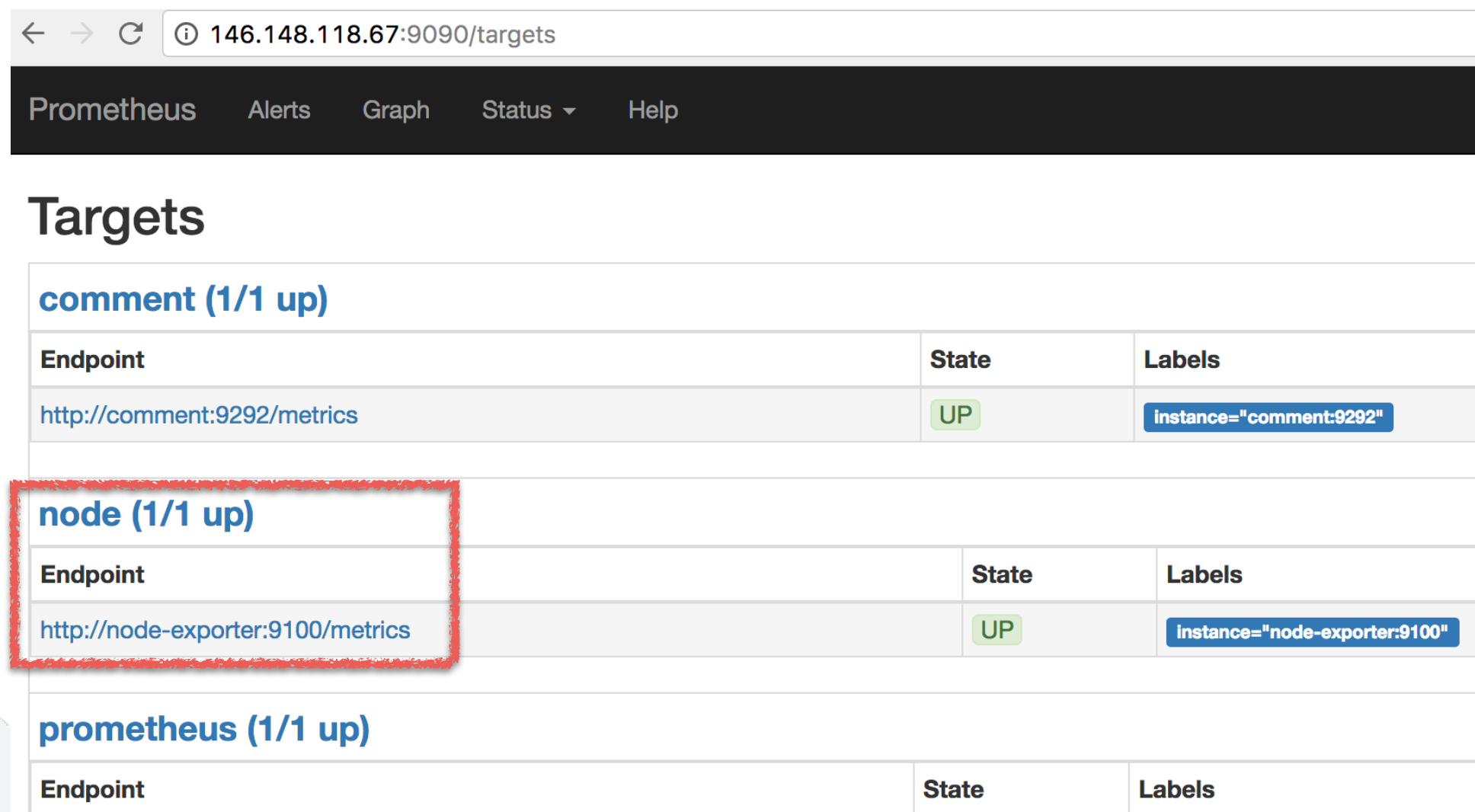
Не забудем собрать новый Docker образ для Прометея:

```
/prometheus (microservices ✓) $ docker build -t $USER_NAME/prometheus .
```

Пересоздадим наши сервисы

```
$ docker-compose down  
$ docker-compose up -d
```

посмотрим, список endpoint-ов Прометея - должен появиться еще один endpoint.



The screenshot shows the Prometheus web interface at the URL `146.148.118.67:9090/targets`. The navigation bar includes links for Prometheus, Alerts, Graph, Status, and Help. The main content area is titled "Targets" and displays three target groups, each with a table of endpoints.

comment (1/1 up)

Endpoint	State	Labels
http://comment:9292/metrics	UP	instance="comment:9292"

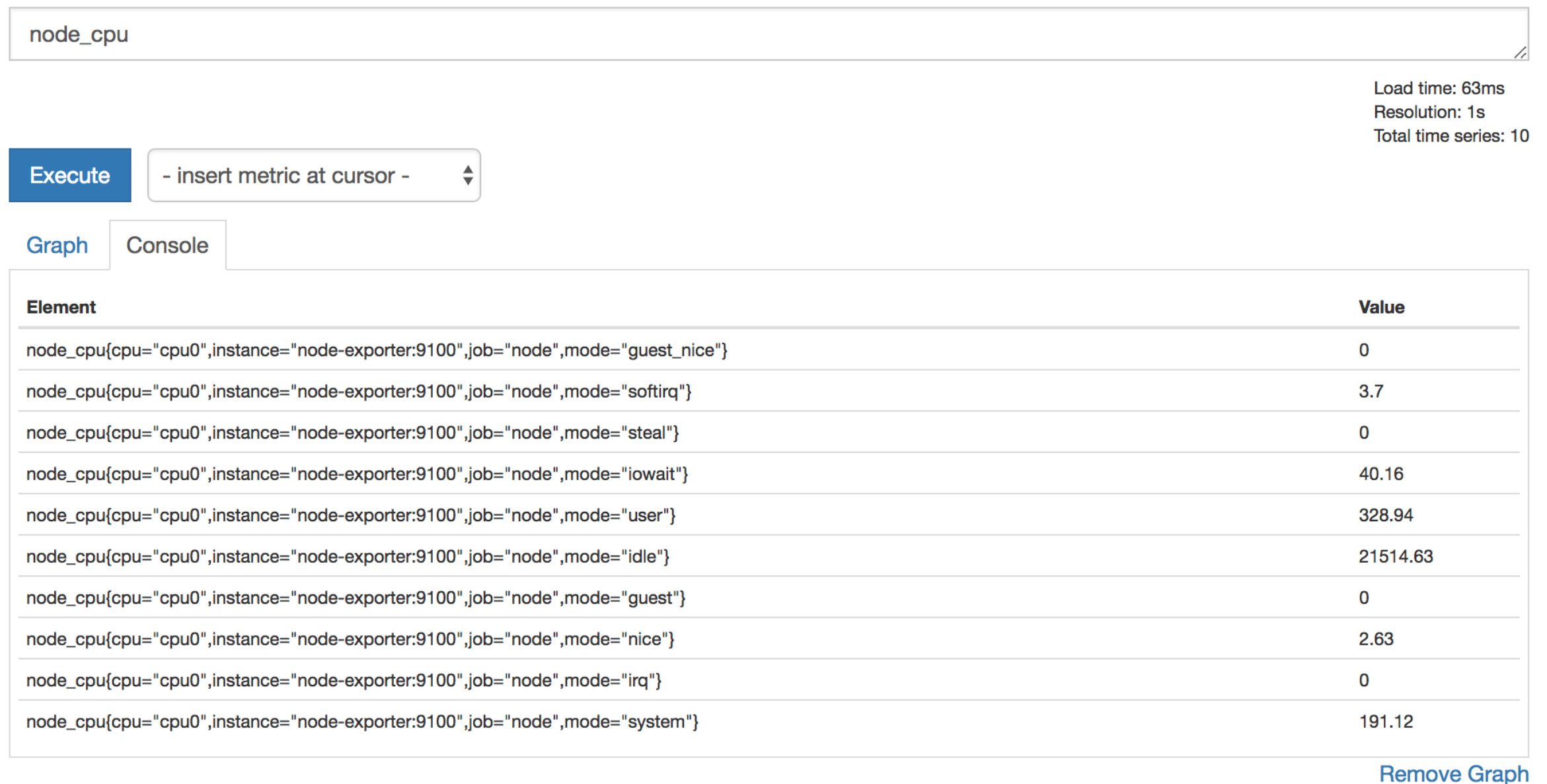
node (1/1 up)

Endpoint	State	Labels
http://node-exporter:9100/metrics	UP	instance="node-exporter:9100"

prometheus (1/1 up)

Endpoint	State	Labels
----------	-------	--------

Получим информацию об использовании CPU на хостовой машине



Завершение работы

Запустите собранные вами образы на DockerHub:

```
$ docker login
Login Succeeded
$ docker push $USER_NAME/ui
$ docker push $USER_NAME/comment
$ docker push $USER_NAME/post
$ docker push $USER_NAME/prometheus
```

Удалите виртуалку:

```
$ docker-machine rm vm1
```

Задания со звездочкой

1. Сделать мониторинг MongoDB с использованием экспортера.
2. Исследовать вопрос необходимости использования blackbox экспортера. Если вы решите добавлять данный экспортер, нужно дать пояснение в пул реквесте, почему вы видите необходимость его использования.