Развертывание стека приложений в kubernetes для закрепления изученных практик и инструментов

--

Для начала работ вам понадобится:

- *docker*
- *k3d*
- *kubectl*
- *helm*

Подробная установка всех необходимых компонентов рассмотрена в предыдущих уроках

--

Часть 1. Рзавертывание кластера k3d и инсталяция istio service mesh с помощью helm

Создадим kubernetes-cluster k3d:

```
$ k3d cluster create mycluster
```

```
filipp@filipp-notebook:~$ k3d cluster create mycluster
INFO[0000] Prep: Network
INFO[0000] Created network 'k3d-mycluster'
INFO[0000] Created volume 'k3d-mycluster-images'
INFO[0000] Starting new tools node...
INFO[0000] Starting Node 'k3d-mycluster-tools'
INFO[0001] Creating node 'k3d-mycluster-server-0'
INFO[0001] Creating LoadBalancer 'k3d-mycluster-serverlb'
INFO[0001] Using the k3d-tools node to gather environment information
INFO[0001] HostIP: using network gateway 192.168.160.1 address
INFO[0001] Starting cluster 'mycluster'
INFO[0001] Starting servers...
INFO[0001] Starting Node 'k3d-mycluster-server-0'
INFO[0005] All agents already running.
INFO[0005] Starting helpers...
INFO[0005] Starting Node 'k3d-mycluster-serverlb'
INFO[0012] Injecting '192.168.160.1 host.k3d.internal' into /etc/hosts of all nodes...
INFO[0012] Injecting records for host.k3d.internal and for 2 network members into Core
INFO[0013] Cluster 'mycluster' created successfully!
INFO[0013] You can now use it like this:
kubectl cluster-info
```

Создадим неймспейс для istio service mesh:

```
$ kubectl create ns istio-system
```

```
filipp@filipp-notebook:~$ kubectl create ns istio-system
namespace/istio-system created
```

С помощью шаблонизатора helm проинсталируем chart istio base:

```
$ helm install istio-base istio/base -n istio-system
```

```
filipp@filipp-notebook:~$ helm install istio-base istio/base -n istio-system
NAME: istio-base
LAST DEPLOYED: Thu Mar 23 17:24:23 2023
NAMESPACE: istio-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Istio base successfully installed!

To learn more about the release, try:
  $ helm status istio-base
  $ helm get all istio-base
```

Также проинсталируем chart istio istiod:

```
$ helm install istiod istio/istiod -n istio-system --wait
```

```
filipp@filipp-notebook:~$ helm install istiod istio/istiod -n istio-system --wait
NAME: istiod
LAST DEPLOYED: Thu Mar 23 17:25:06 2023
NAMESPACE: istio-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
"istiod" successfully installed!

To learn more about the release, try:
  $ helm status istiod
  $ helm get all istiod

Next steps:
  * Deploy a Gateway: https://istio.io/latest/docs/setup/additional-setup/gateway/
  * Try out our tasks to get started on common configurations:
    * https://istio.io/latest/docs/tasks/traffic-management
    * https://istio.io/latest/docs/tasks/security/
    * https://istio.io/latest/docs/tasks/policy-enforcement/
    * https://istio.io/latest/docs/tasks/policy-enforcement/
  * Review the list of actively supported releases, CVE publications and our hardening guide
    * https://istio.io/latest/docs/releases/supported-releases/
    * https://istio.io/latest/news/security/
    * https://istio.io/latest/docs/ops/best-practices/security/

For further documentation see https://istio.io website

Tell us how your install/upgrade experience went at https://forms.gle/hMHGiwZHPU7UQRWe9
```

Проинсталируем необходимые addons для istio:

- kiali:

```
$ kubectl apply -f
https://raw.githubusercontent.com/istio/istio/release-
1.13/samples/addons/kiali.yaml
```

```
filipp@filipp-notebook:~$ kubectl apply -f https://raw.githubusercontent.com/istio/istio/relea
ns/kiali.yaml
serviceaccount/kiali created
configmap/kiali created
clusterrole.rbac.authorization.k8s.io/kiali-viewer created
clusterrole.rbac.authorization.k8s.io/kiali created
clusterrolebinding.rbac.authorization.k8s.io/kiali created
role.rbac.authorization.k8s.io/kiali-controlplane created
rolebinding.rbac.authorization.k8s.io/kiali-controlplane created
service/kiali created
deployment.apps/kiali created
```

- prometheus:

```
$ kubectl apply -f
https://raw.githubusercontent.com/istio/istio/release-
1.13/samples/addons/prometheus.yaml
```

```
filipp@filipp-notebook:~$ kubectl apply -f https://raw.githubusercontent.com/istio/istio/releas
ns/prometheus.yaml
serviceaccount/prometheus created
configmap/prometheus created
clusterrole.rbac.authorization.k8s.io/prometheus created
clusterrolebinding.rbac.authorization.k8s.io/prometheus created
service/prometheus created
deployment.apps/prometheus created
```

- grafana:

```
$ kubectl apply -f
https://raw.githubusercontent.com/istio/istio/release-
1.17/samples/addons/grafana.yaml
```

```
filipp@filipp-notebook:~$ kubectl apply -f https://raw.githubusercontent.com/istio/istio/releas
ns/grafana.yaml
serviceaccount/grafana created
configmap/grafana created
service/grafana created
deployment.apps/grafana created
configmap/istio-grafana-dashboards created
configmap/istio-services-grafana-dashboards created
```

Изменим тип службы для `kiali` с ClusterIP на `NodePort`:

```
$ kubectl edit svc kiali -n istio-system
```

```
spec:
  clusterIP: 10.43.228.50
  clusterIPs:
  - 10.43.228.50
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    port: 20001
    protocol: TCP
    targetPort: 20001
  - name: http-metrics
    port: 9090
    protocol: TCP
    targetPort: 9090
  selector:
    app.kubernetes.io/instance: kiali
    app.kubernetes.io/name: kiali
  sessionAffinity: None
  type: NodePort
```

Подобным образом поступим и со служюой `grafana`:

```
$ kubectl edit svc grafana -n istio-system
```

```
  name: grafana
  namespace: istio-system
  resourceVersion: "1256"
  uid: 4c3a8068-fb65-46a4-a35e-26c5742e5143
spec:
  clusterIP: 10.43.6.121
  clusterIPs:
  - 10.43.6.121
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: service
    port: 3000
    protocol: TCP
    targetPort: 3000
  selector:
    app.kubernetes.io/instance: grafana
    app.kubernetes.io/name: grafana
  sessionAffinity: None
  type: NodePort
```

Службы успешно изменены:

```
filipp@filipp-notebook:~$ kubectl edit svc kiali -n istio-system
service/kiali edited
filipp@filipp-notebook:~$ kubectl edit svc grafana -n istio-system
service/grafana edited
```

Кластр запущен, istio service mesh с аддонами kiali, prometheus и grafana успешно развернуты

--

Часть 2. Установка инструмента SAST/DAST анализа mobsf

Создадим неймспейс для инструмента mobsf:

```
$ kubectl create ns mobsf
```

```
filipp@filipp-notebook:~$ kubectl create ns mobsf
namespace/mobsf created
```

Добавим сервисы, развертываемые в неймспейсе `mobsf` в сервисную сетку, посредством вызова модифицирующего вебхука через добавление специальной метки к неймспейсу:

```
$ kubectl label namespace mobsf istio-injection=enabled
```

```
filipp@filipp-notebook:~$ kubectl label namespace mobsf istio-injection=enabled
namespace/mobsf labeled
```

Создадим деплоймент инструмента mobsf на основе образа docker версии `v3.1.1`:

```
$ kubectl create deployment mobsf
--image=opensecurity/mobile-security-framework-
mobsf:v3.1.1 -n mobsf
```

```
filipp@filipp-notebook:~$ kubectl create deployment mobsf --image=opens
1.1 -n mobsf
deployment.apps/mobsf created
```

Создадим службу для деплоймента `mobsf`:

```
$ kubectl expose deployment mobsf --port=8000 --
type=NodePort -n mobsf
```

```
filipp@filipp-notebook:~$ kubectl expose deployment mobsf --port=8000 --type=NodePort -n mobsf
service/mobsf exposed
```

Инструмент mobsf успешно развернут в кластере и добавлен в сервисную сетку

--

Часть 3. Установка mariadb и adminer в кластере kubernetes

Создадим неймспейс для базы данных mariadb и средства управления базами данных adminer:

```
$ kubectl create ns db
```

```
filipp@filipp-notebook:~$ kubectl create ns db
namespace/db created
```

Добавим сервисы, развертываемые в неймспейсе `db` в сервисную сетку, посредством вызова модифицирующего вебхука через добавление специальной метки к неймспейсу:

```
$ kubectl label namespace db istio-injection=enabled
```

```
filipp@filipp-notebook:~$ kubectl label namespace db istio-injection=enabled
namespace/db labeled
```

Создадим pod с базой данных mariadb в неймспейсе `db`:

```
$ kubectl run mariadb --image=mariadb --port 3306 --env
MARIADB_ROOT_PASSWORD=superpass -n db
```

```
filipp@filipp-notebook:~$  kubectl run mariadb --image=mariadb --port 3306 --env MARIAD
n db
pod/mariadb created
```

Создадим pod со средством управления базами данных adminer в неймспейсе `db`:

```
kubectl run adminer --image=adminer -n db
```

```
filipp@filipp-notebook:~$ kubectl run adminer --image=adminer -n db
pod/adminer created
```

Создадим службы для pods `mariadb` и `adminer`:

```
$ kubectl expose pod mariadb -n db
$ kubectl expose pod adminer --port 8080 --type=NodePort
-n db
```

```
filipp@filipp-notebook:~$ kubectl expose pod mariadb -n db
service/mariadb exposed
filipp@filipp-notebook:~$ kubectl expose pod adminer --port 8080 --type=NodePort -n db
service/adminer exposed
```

pods `adminer` и `mariadb` успешно развернуты в кластере и добавлены в сервисную сетку

--

Часть 4. Визуализация сбора метрик и работы системы мониторинга приложений

Сэмулируем рабочую нагрузку на развернутые в кластере инструменты. Сперва получим `external-ip` обработчика входящих соединений в нашем кластере - службы `traefik`:

```
$ kubectl get svc traefik -n kube-system
```

```
filipp@filipp-notebook:~$ kubectl get svc traefik -n kube-system
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
traefik   LoadBalancer   10.43.247.75   192.168.160.3   80:30861/TCP,443:31093/TCP   28m
```

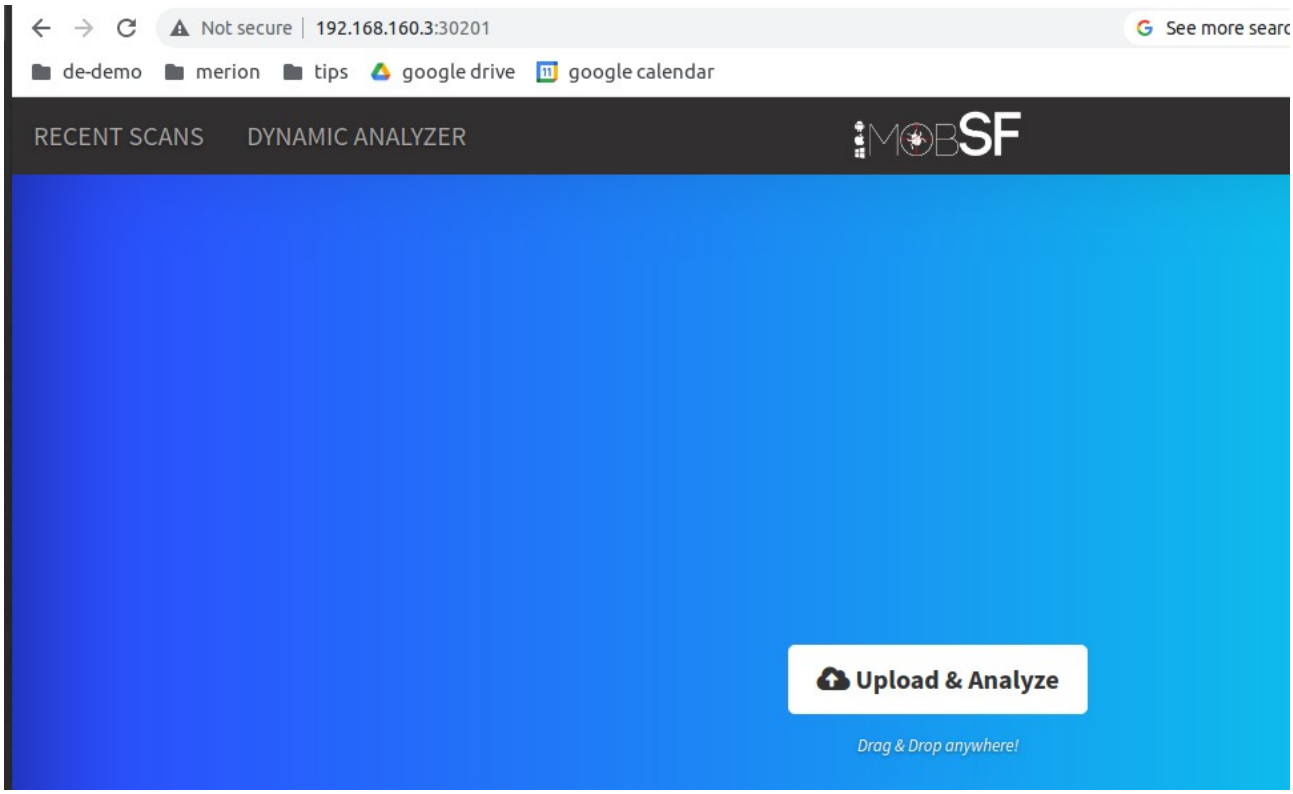
Далее, получим node port службы `mobsf`:

```
$ kubectl get svc mobsf -n mobsf
```

```
filipp@filipp-notebook:~$ kubectl get svc mobsf -n mobsf
NAME    TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
mobsf   NodePort   10.43.151.11   <none>        8000:30201/TCP   10m
```

Откроем веб-интерфейс инструмента mobsf и произведем имитацию полезной нагрузки:

```
browser: http://<traefik-svc-external-ip>:<mobsf-svc-nodeport>
```



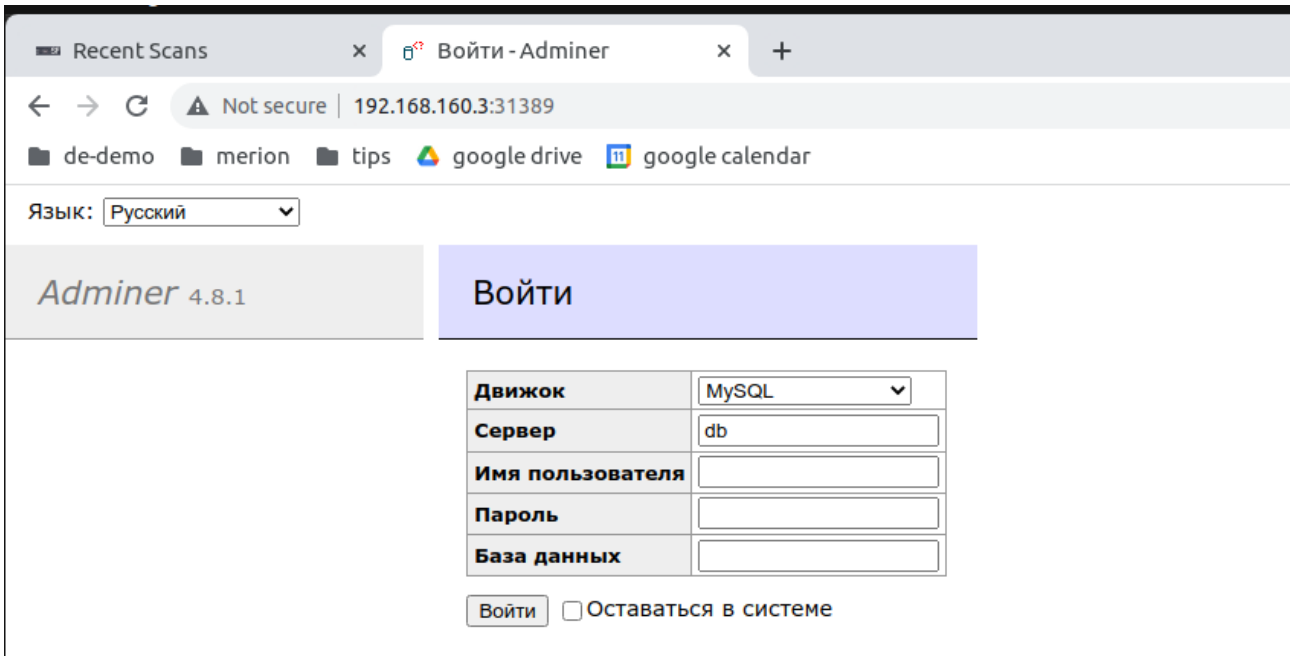
Вернемся в терминал и выполним команду просмотра node port службы adminer в неймспейсе `db`:

```
$ kubectl get svc adminer -n db
```

```
filipp@filipp-notebook:~$ kubectl get svc adminer -n db
NAME      TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
adminer   NodePort   10.43.183.236   <none>        8080:31389/TCP   64s
```

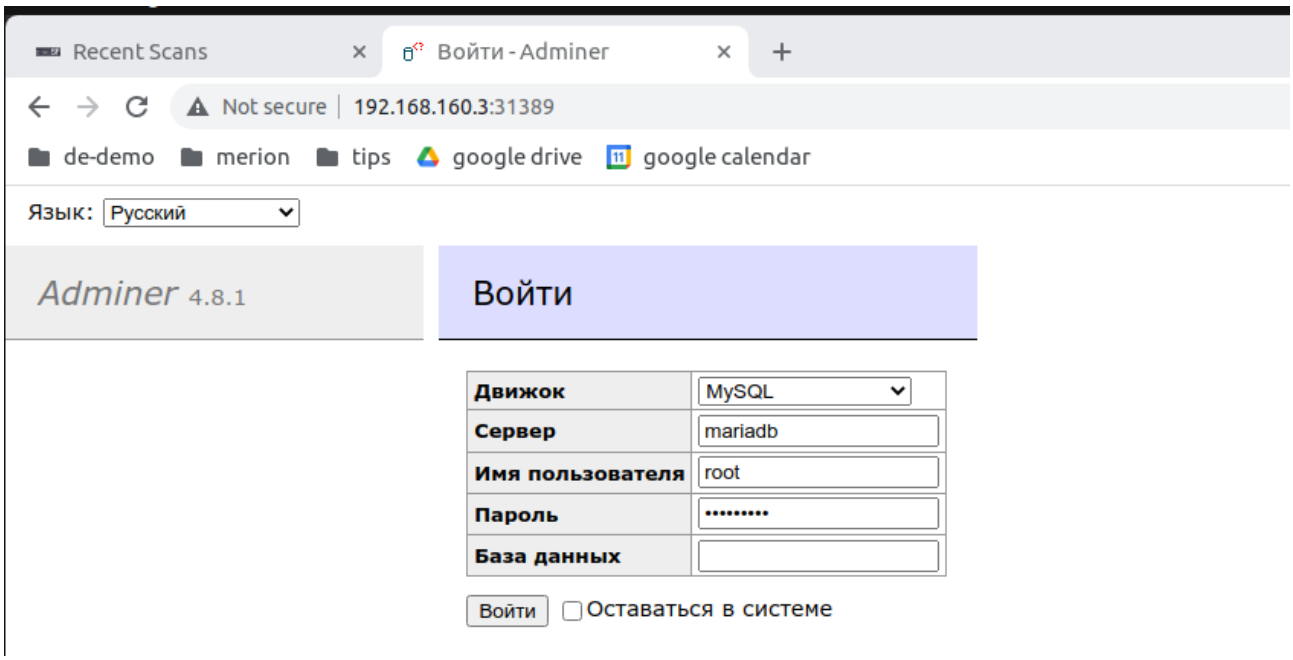
Откроем веб-интерфейс инструмента adminer:

`browser: http://<traefik-svc-external-ip>:<adminer-svc-nodeport>`

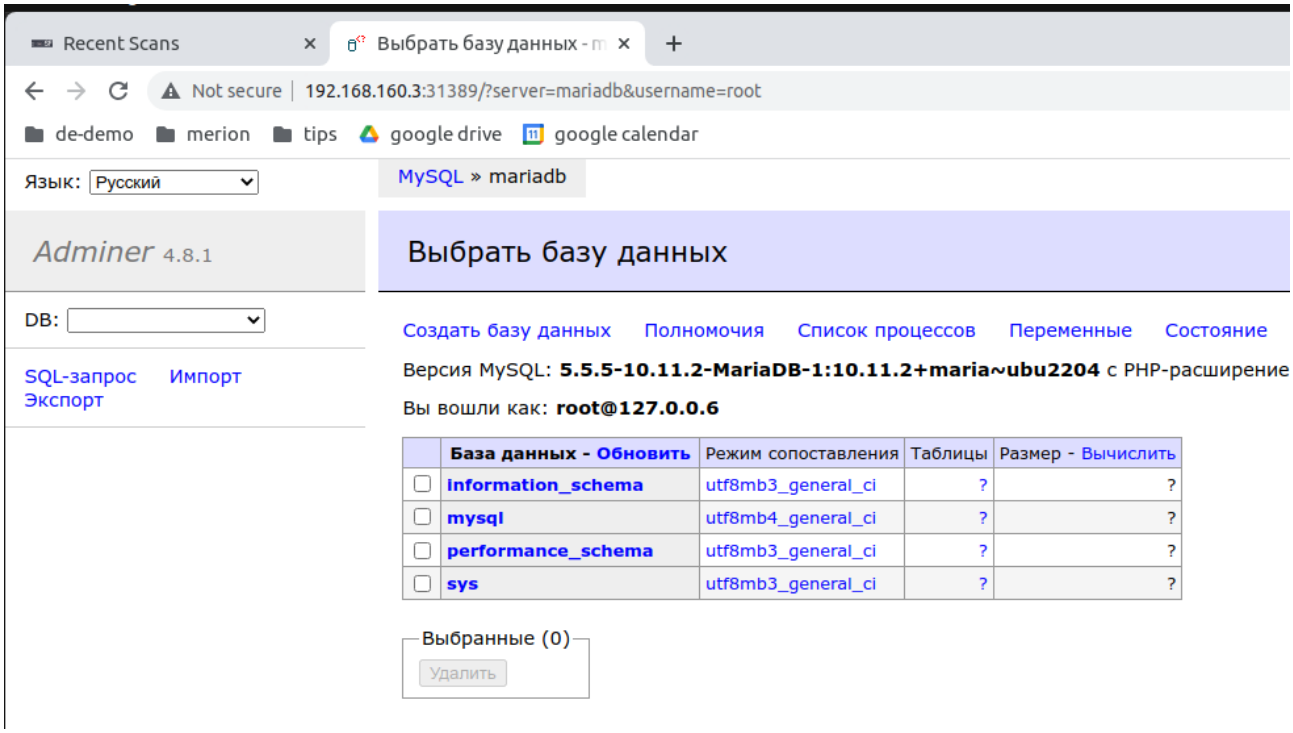


Выполним соединение с базой данных mariadb

Сервер: `mariadb`
Имя пользователя: `root`
Пароль: `superpass`
(поле "База данных" оставить пустым)



соединение успешно выполнено:



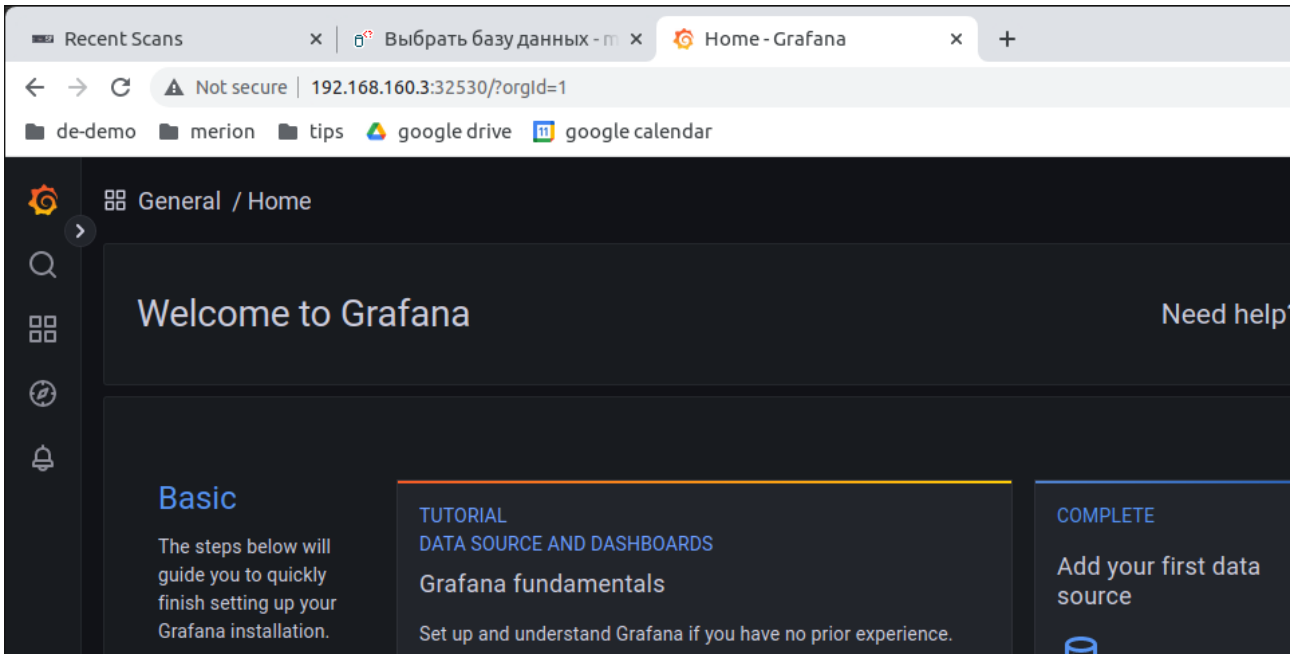
Возвращаемся в терминал и узнаем node port службы `grafana` в неймспейсе `istio-system`:

```
$ kubectl get svc grafana -n istio-system
```

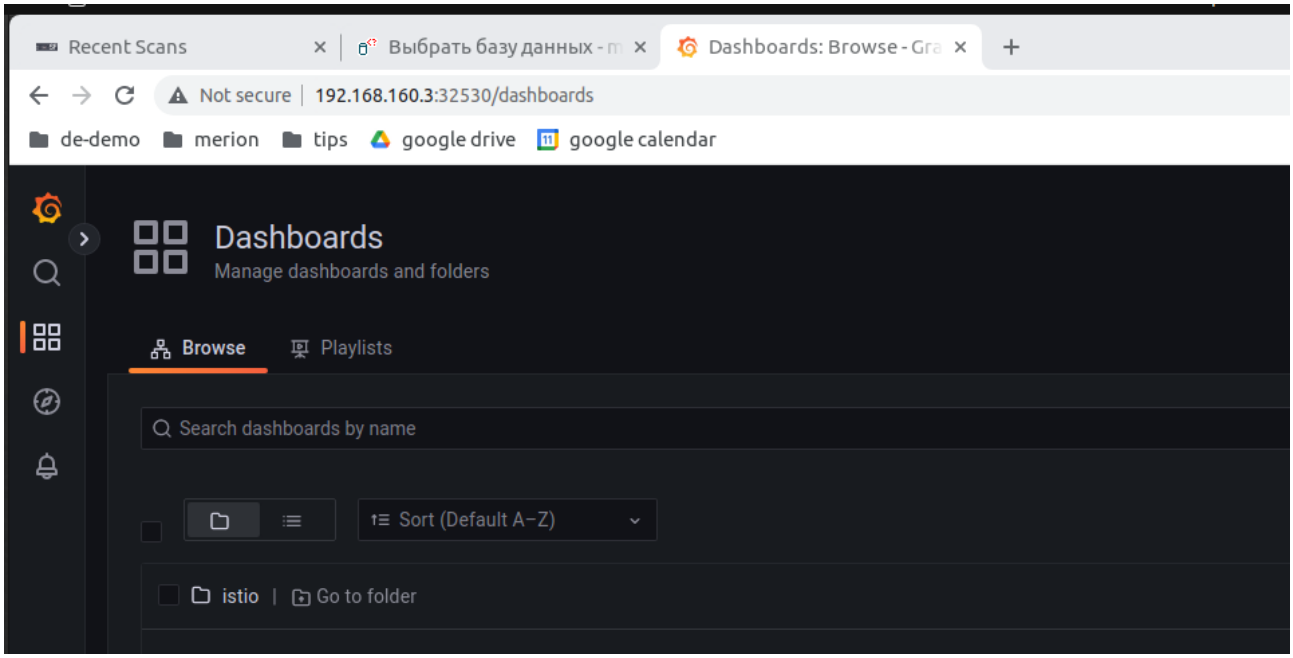
```
filipp@filipp-notebook:~$ kubectl get svc grafana -n istio-system
NAME      TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
grafana   NodePort   10.43.6.121   <none>        3000:32530/TCP   39m
```

Переходим в веб-интерфейс инструмента grafana в браузере:

`browser: http://<traefik-svc-external-ip>:<grafana-svc-nodeport>`

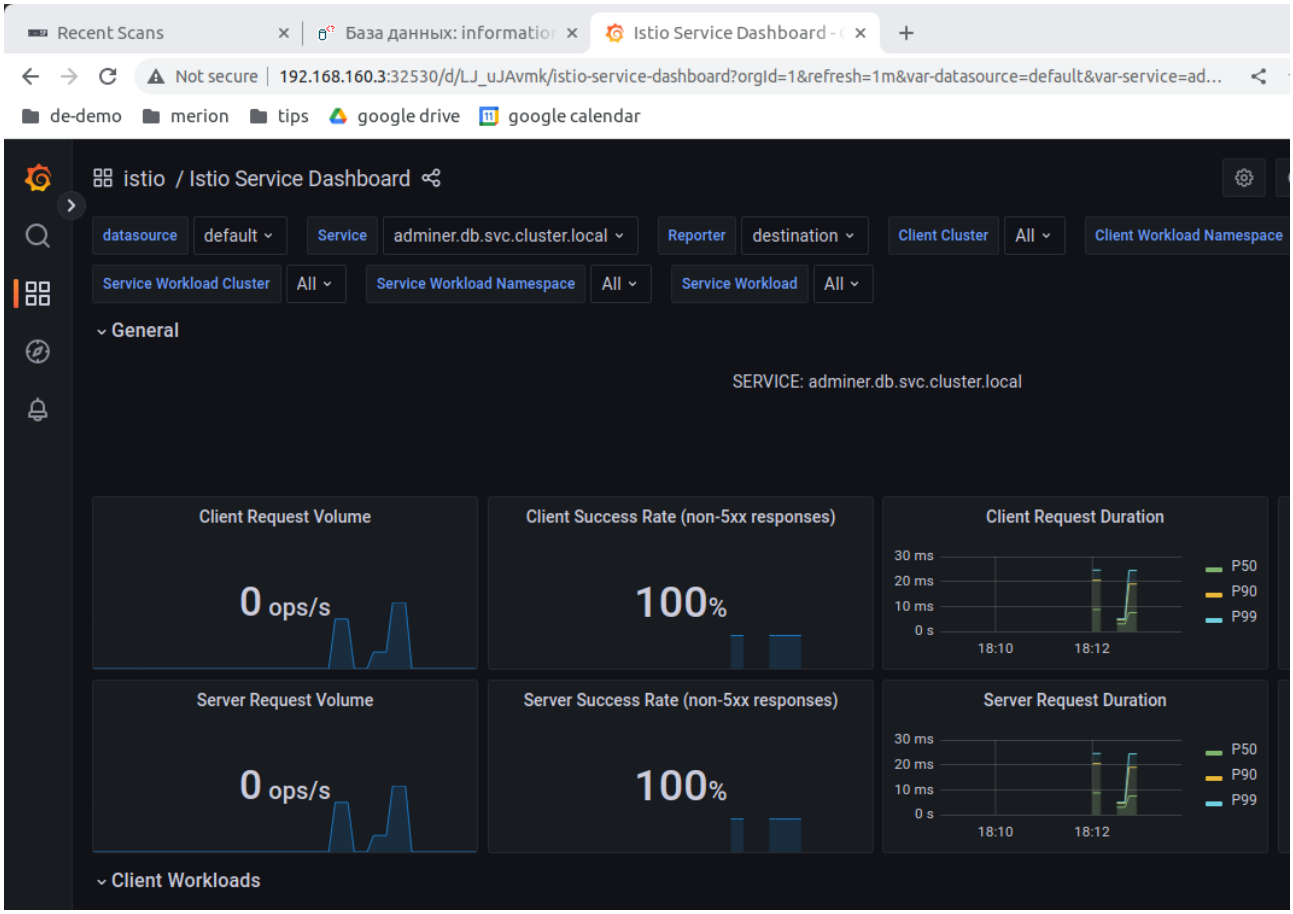


Переходим во вкладку "Dashboards -> Browse":

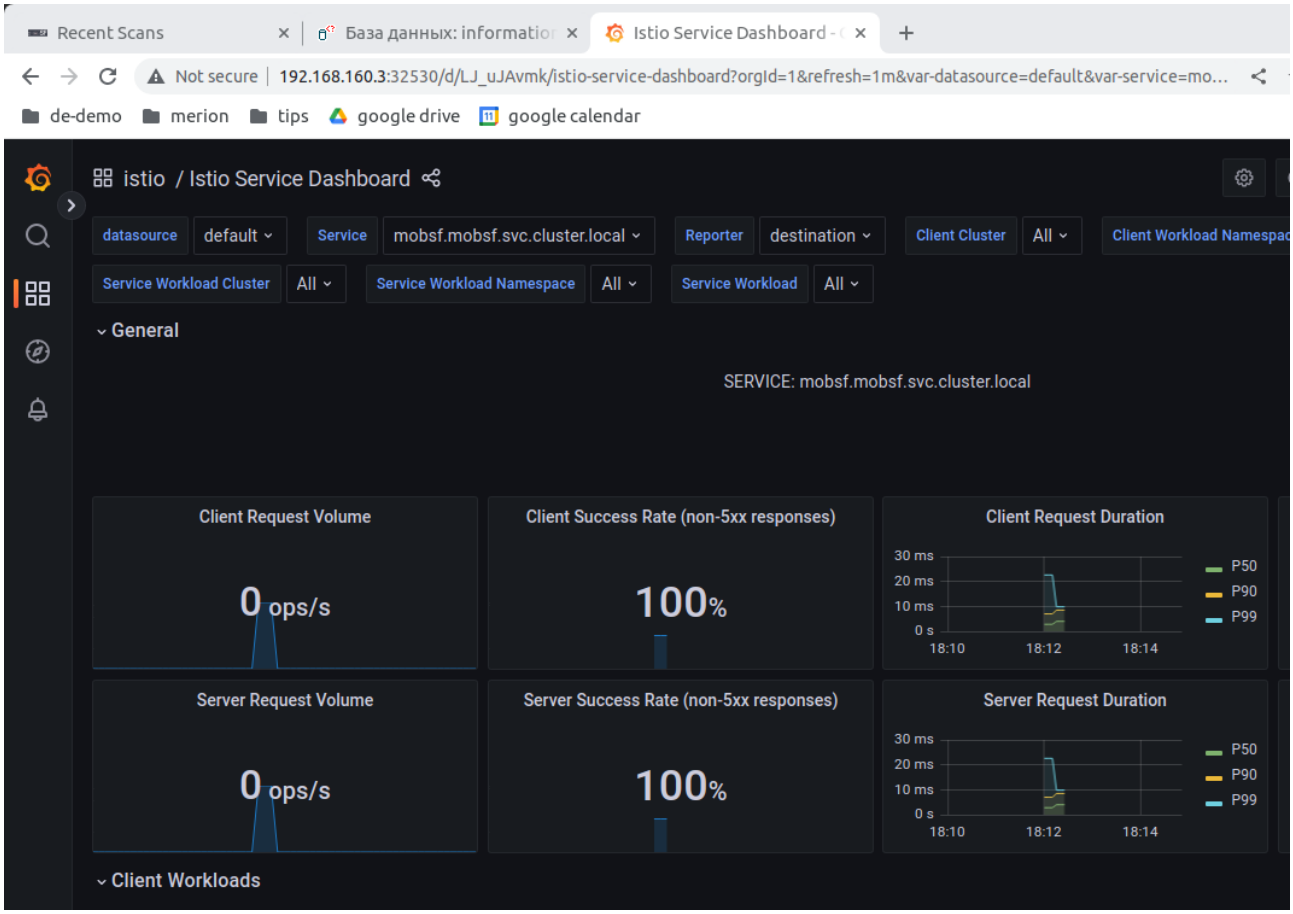


Переходим в уже имеющийся и преднастроенный дашборд под названием "Istio -> Istio service dashboard" и выбираем в опциях дашборда нужные службы

- `adminer`



- `mobsf:`



Система мониторинга настроена корректно, осуществляется сбор метрик и их визуализация

--

Часть 5. Визуализация сервисного взаимодействия в виде графа в дашборде kiali

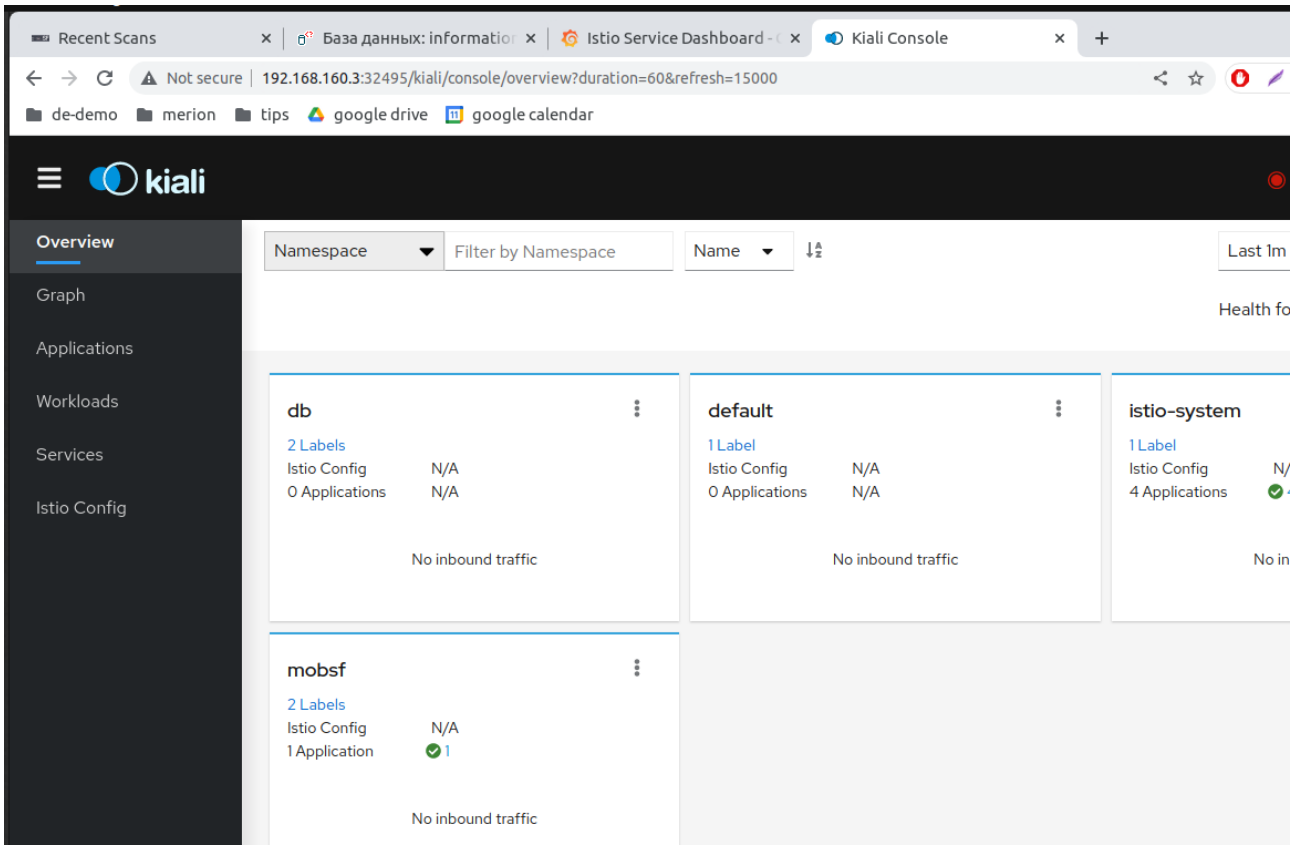
Возвращаемся в терминал и узнаем node port службы `kiali` в неймспейсе `istio-system`:

```
$ kubectl get svc kiali -n istio-system
```

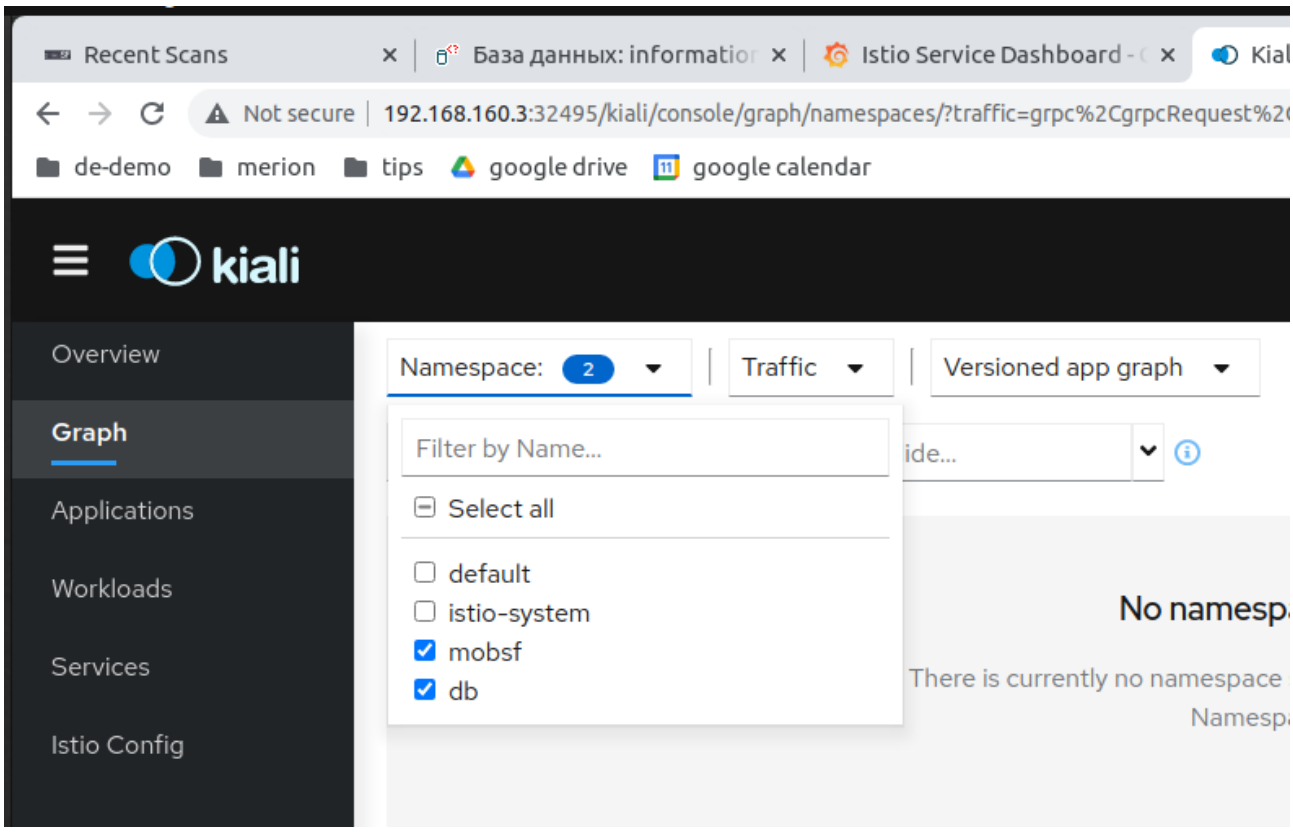
```
filipp@filipp-notebook:~$ kubectl get svc kiali -n istio-system
NAME    TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)                          AGE
kiali   NodePort   10.43.228.50   <none>        20001:32495/TCP,9090:32030/TCP   50m
```

Переходим в веб-интерфейс инструмента kiali в браузере:

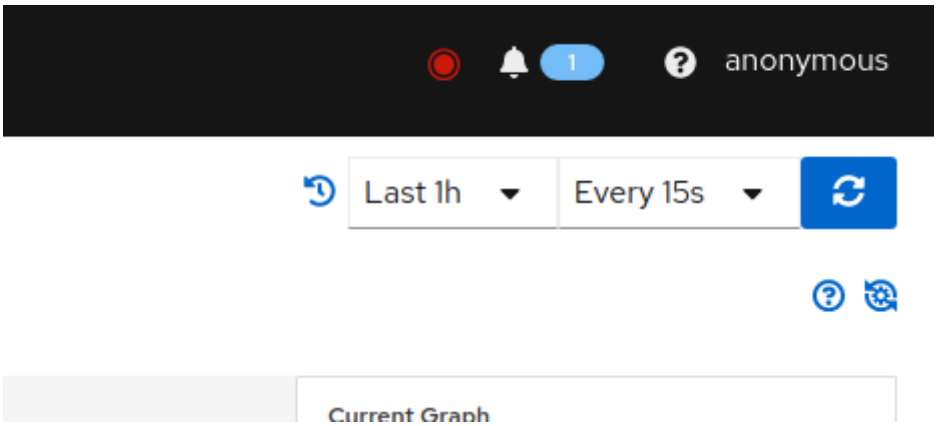
`browser: http://<traefik-svc-external-ip>:<kiali-svc-nodeport>`



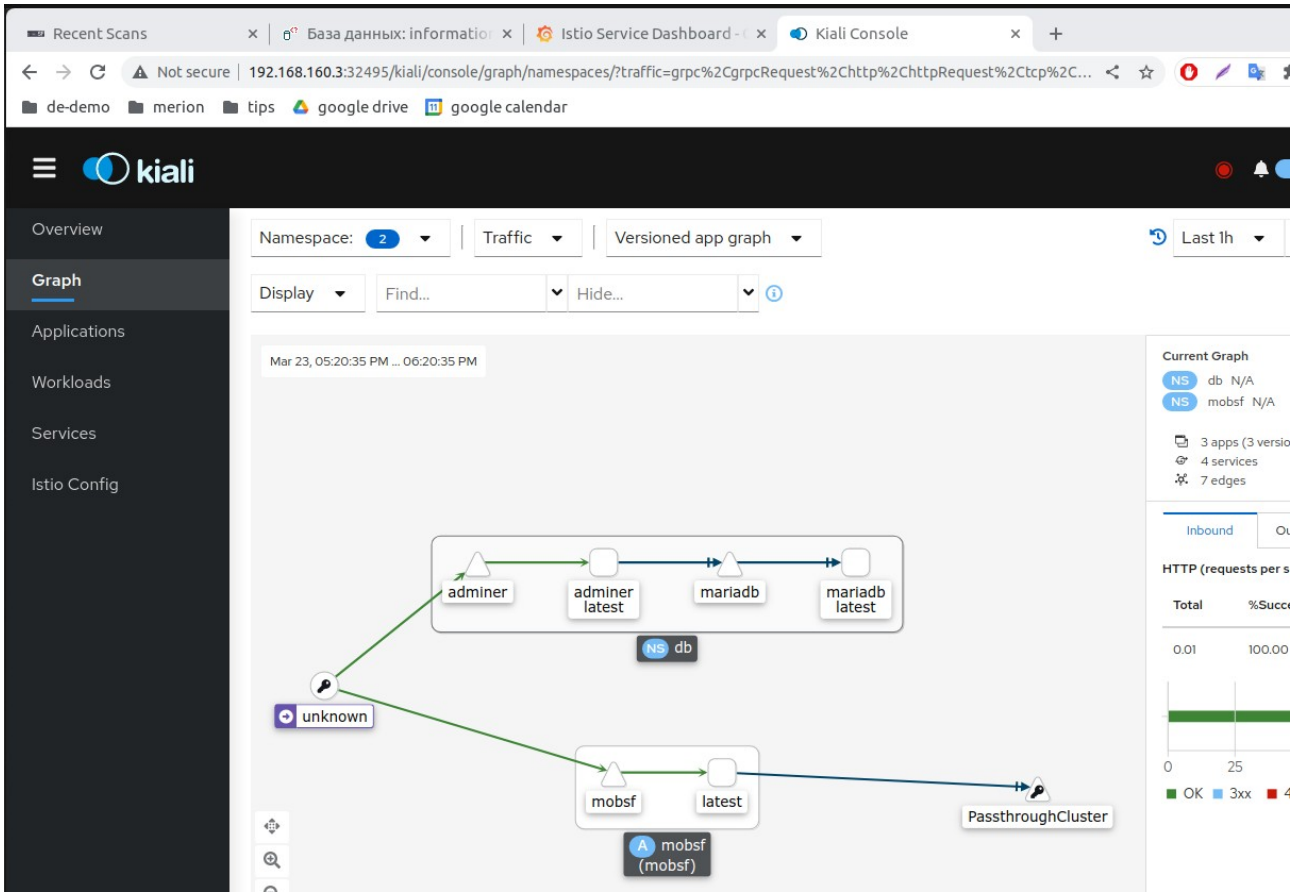
Переходим во вкладку "Graph", в фильтре указываем неймспейсы `mobsf` и `db`:



В выпадающем списке указываем интервал "Last 1h" и визуализируем граф межсервисного взаимодействия:



Построенный граф отображает межсервисное взаимодействие:



Межсервисное взаимодействие в сервисной сетке успешно визуализируется в виде графа

--

Для завершения работ выполните команду уничтожения кластера k3d:

```
$ k3d cluster delete mycluster
```

```
filipp@filipp-notebook:~$ k3d cluster delete mycluster
INFO[0000] Deleting cluster 'mycluster'
INFO[0003] Deleting cluster network 'k3d-mycluster'
INFO[0003] Deleting image volume 'k3d-mycluster-images'
INFO[0003] Removing cluster details from default kubeconfig...
INFO[0003] Removing standalone kubeconfig file (if there is one)...
INFO[0003] Successfully deleted cluster mycluster!
```

Кластер и все рабочие нагрузки успешно удалены

--