

Развертывание и настройка сервисной сетки на базе istio

Для выполнения практической работы вам понадобится:

- *k3d cluster*
- *helm*

--

Часть 1. Создание кластера k3d и установка инструмента mobsf

Для начала работ создадим k3d cluster:

```
$ 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 172.19.0.1 address
INFO[0001] Starting cluster 'mycluster'
INFO[0001] Starting servers...
INFO[0001] Starting Node 'k3d-mycluster-server-0'
INFO[0006] All agents already running.
INFO[0006] Starting helpers...
INFO[0006] Starting Node 'k3d-mycluster-serverlb'
INFO[0012] Injecting '172.19.0.1 host.k3d.internal' into /etc/hosts of
INFO[0012] Injecting records for host.k3d.internal and for 2 network m
INFO[0013] Cluster 'mycluster' created successfully!
INFO[0013] You can now use it like this:
kubectl cluster-info
```

Создадим неймспейс для инструмента mobsf:

```
$ kubectl create ns mobsf
```

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

Создадим деплоймент mobsf с указанием версии инструмента v3.1.1 в созданном ранее неймспейсе:

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

```
filipp@filipp-notebook:~$ kubectl create deployment mobsf --image=
opensecurity/mobile-security-framework-mobsf:v3.1.1 -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
```

Выполним команду для просмотра всех сервисных сущностей в неймспейсе mobsf:

```
$ kubectl get all -n mobsf
```

```
filipp@filipp-notebook:~$ kubectl get all -n mobsf
NAME                                READY   STATUS    RESTARTS   AGE
pod/mobsf-5db69649fc-vt99b         1/1     Running   0           4m12s

NAME                                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/mobsf                       NodePort      10.43.233.95 <none>        8000:31454/TCP   3m5s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mobsf               1/1     1            1           4m12s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/mobsf-5db69649fc    1         1         1       4m12s
```

pod, service, deployment и replicaset инструмента mobsf успешно установлены и запущены. Запомним nodePort службы mobsf, он нам пригодится далее.

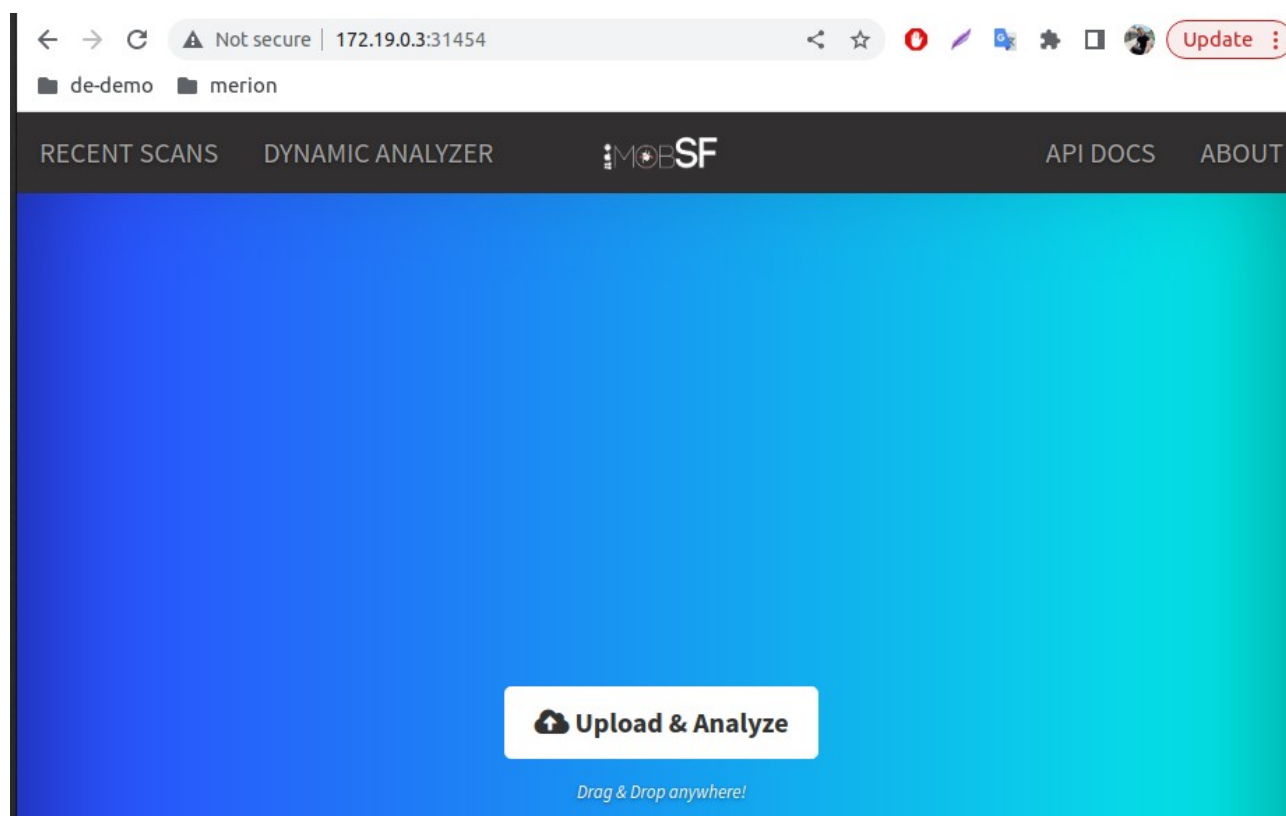
Выполним запрос к системному неймспейсу kubernetes, чтобы узнать external-ip у службы traefik (выполняющую роль обработчика входящих соединений в нашем кластере):

```
$ 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 10.43.78.93   172.19.0.3    80:30092/TCP,443:31235/TCP 7m42s
```

Перейдем в браузер и откроем страницу с веб-интерфейсом развернутого нами инструмента mobsf, чтобы убедиться в его работоспособности:

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



Инструмент mobsf успешно развернут и работает в штатном режиме

--

Часть 2. Установка istio service mesh

Выполним создание неймспейса `istio-system` для инструмента istio service mesh:

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

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

С помощью шаблонизатора `helm` выполним установку компонентов инструмента `istio service mesh` (в стандартной конфигурации по умолчанию):

```
$ 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: Tue Jan 24 16:53:05 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
```

```
$ 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: Tue Jan 24 16:53:37 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/99uiMML96AmsXY5d6
```

Далее произведем установку двух addons для `istio`:

- `kiali` (dashboard для визуализации межсервисного взаимодействия)
- `prometheus` (инструмент для сбора метрик)

Установка kiali dashboard (остерегайтесь установки через применение удаленного манифеста, перепроверяйте его содержимое - зайдите по самой ссылке в браузере и убедитесь что устанавливаете именно то, что требуется):

```
$ 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/release-1.13/samples/addons/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
```

Поупражняемся в изменении конфигурации службы (изменим тип службы kiali с ClusterIP на NodePort):

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

```
port: 9090
protocol: TCP
targetPort: 9090
selector:
  app.kubernetes.io/instance: kiali
  app.kubernetes.io/name: kiali
sessionAffinity: None
type: NodePort
status:
  loadBalancer: {}
-- INSERT --
```

Служба успешно изменена:

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

Произведем также установку инструмента 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/release-1.13/samples/addons/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
```

Убедимся, что все необходимые компоненты и addons istio service mesh установлены:

```
$ kubectl get all -n istio-system
```

```
filipp@filipp-notebook:~$ kubectl get all -n istio-system
```

NAME	READY	STATUS	RESTARTS	AGE
pod/istiod-5d5b45c577-rd2qq	1/1	Running	0	7m58s
pod/kiali-699f98c497-f45dn	1/1	Running	0	4m15s
pod/prometheus-699b7cc575-rwst4	2/2	Running	0	60s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/istiod	ClusterIP	10.43.40.156	<none>	15010/TCP,15012/TCP,443/TCP,15014/TCP	7m58s
service/kiali	NodePort	10.43.137.195	<none>	20001:32270/TCP,9090:30607/TCP	4m15s
service/prometheus	ClusterIP	10.43.193.9	<none>	9090/TCP	60s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/istiod	1/1	1	1	7m58s
deployment.apps/kiali	1/1	1	1	4m15s
deployment.apps/prometheus	1/1	1	1	60s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/istiod-5d5b45c577	1	1	1	7m58s
replicaset.apps/kiali-699f98c497	1	1	1	4m15s
replicaset.apps/prometheus-699b7cc575	1	1	1	60s

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
horizontalpodautoscaler.autoscaling/istiod	Deployment/istiod	0%/80%	1	5	1	7m58s

Все сущности установлены и запущены успешно

--

Часть 3. Добавление сервиса в service mesh

Для начала выполним команду получения pods в инструменте mobsf:

```
$ kubectl get pods -n mobsf
```



```
filipp@filipp-notebook:~$ kubectl get pods -n mobsf
NAME                                READY   STATUS    RESTARTS   AGE
mobsf-5db69649fc-vt99b             1/1     Running   0           23m
```

Обратите внимание, что в выводе команды отображается значение 1/1 в столбце "READY" - это означает, что в pod на данный момент запущен 1 контейнер (с самим инструментом mobsf)

Далее, выполним добавление специальной метки "istio-injection=enabled" к неймпейсу mobsf для добавления инструмента в сервисную сетку (1 из 2 возможных способов, второй - использование kube-inject в istioctl. Однако при обоих способах будет задействована работа модифицирующего вебхука в kubernetes):

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

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

Убедимся, что метка добавилась к неймспейсу mobsf, выполнив команду:

```
$ kubectl get ns mobsf -o yaml
```

```
filipp@filipp-notebook:~$ kubectl get ns mobsf -o yaml
apiVersion: v1
kind: Namespace
metadata:
  creationTimestamp: "2023-01-24T13:39:57Z"
  labels:
    istio-injection: enabled
    kubernetes.io/metadata.name: mobsf
  name: mobsf
  resourceVersion: "2532"
  uid: b5e2a5d4-1d67-49a6-a748-b6a993886378
spec:
  finalizers:
    - kubernetes
status:
  phase: Active
```

В разделе metadata - labels метка отображается, следовательно она успешно добавилась к неймспейсу

Для вступления в силу модифицирующего вебхука, добавляющего сервис в service mesh, необходимо инициировать перезапуск pod. Сделаем это через дескалирование и скалирование репликаций сервиса mobsf:

```
$ kubectl scale deployment mobsf --replicas=0 -n mobsf
$ kubectl scale deployment mobsf --replicas=1 -n mobsf
```

```
filipp@filipp-notebook:~$ kubectl scale deployment mobsf --replicas=0 -n mobsf
deployment.apps/mobsf scaled
filipp@filipp-notebook:~$ kubectl scale deployment mobsf --replicas=1 -n mobsf
deployment.apps/mobsf scaled
```

Повторим команду вывода pods в неймпейсе mobsf:

```
$ kubectl get pods -n mobsf
```

```
filipp@filipp-notebook:~$ kubectl get pods -n mobsf
NAME                                READY   STATUS    RESTARTS   AGE
mobsf-5db69649fc-bfl22             2/2     Running   0           62s
```

Обратите внимание, что теперь в pod содержится 2 контейнера:

- контейнер с самим инструментом mobsf
- контейнер с sidecar проху (envoy)

Теперь сервис mobsf успешно добавлен в сервисную сетку

--

Часть 4. Визуализация сервисного взаимодействия в service mesh

external-ip службы trefik нам уже известен, узнаем nodePort службы kiali:

```
$ 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    10.43.137.195 <none>        20001:32270/TCP,9090:30607/TCP        19m
```

Нам нужен nodePort, сообщенный с портом контейнера 20001 (nodePort, сообщенный с портом контейнера 9090 используется для сбора метрик)

Перейдем в браузер и откроем страницу с веб-интерфейсом kiali:

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

The screenshot shows the Kiali console overview page. The browser address bar displays `172.19.0.3:32270/kiali/console/overview?duratio...`. The Kiali header includes the logo, a status indicator, a notification bell with '1', and the user 'anonymous'. Below the header, there are filters for 'Namespace' and 'Name', a 'Last 1m' time range, and a refresh button. The main content area shows three namespace cards: 'default', 'istio-system', and 'mobsf'. Each card displays '1 Label', 'Istio Config' status, and 'Applications' count. The 'istio-system' and 'mobsf' namespaces show a green checkmark and a count of 3 and 1 respectively. All namespaces indicate 'No inbound traffic'.

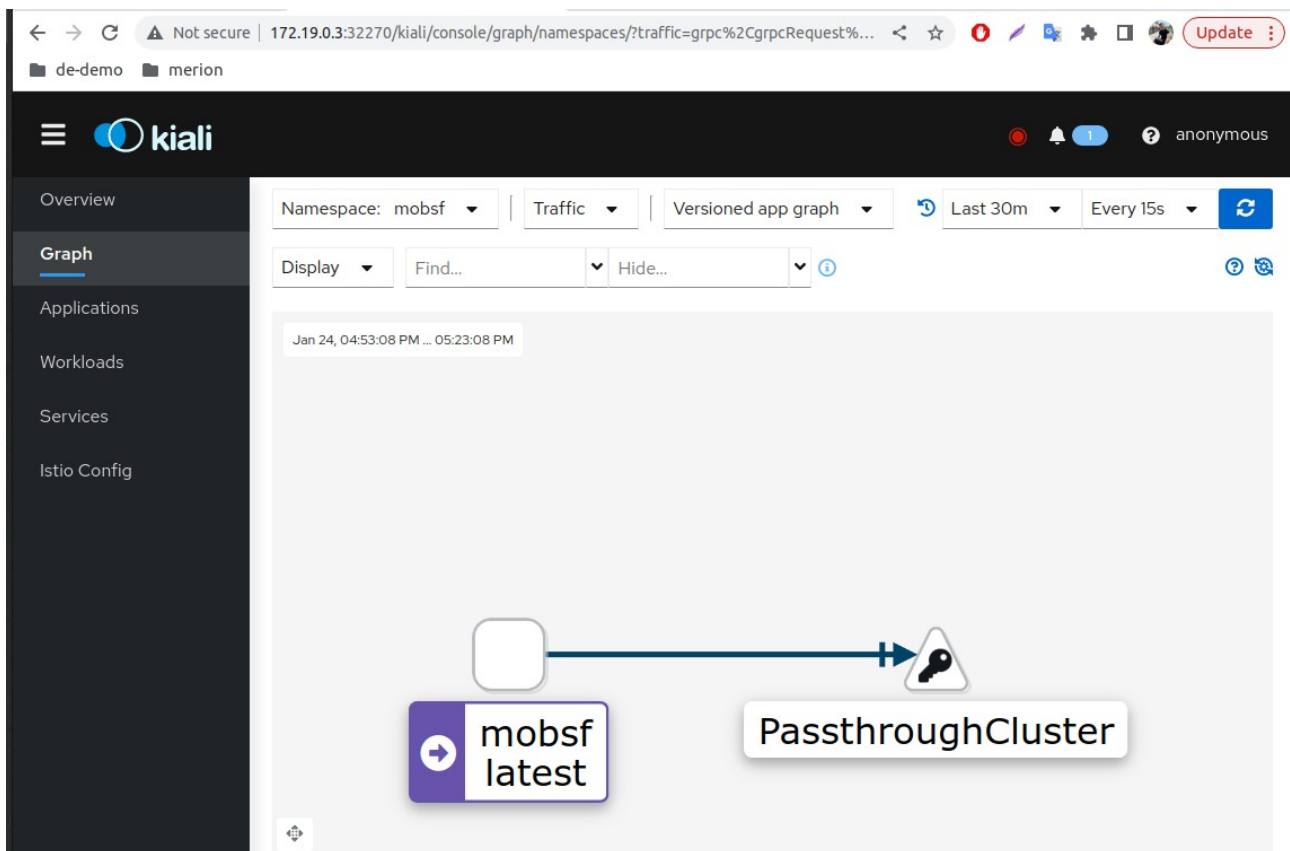
Namespace	Labels	Istio Config	Applications	Inbound Traffic
default	1 Label	N/A	0	No inbound traffic
istio-system	1 Label	N/A	3	No inbound traffic
mobsf	2 Labels	N/A	1	No inbound traffic

Продолжим работу в kiali и перейдем в раздел "Graph":

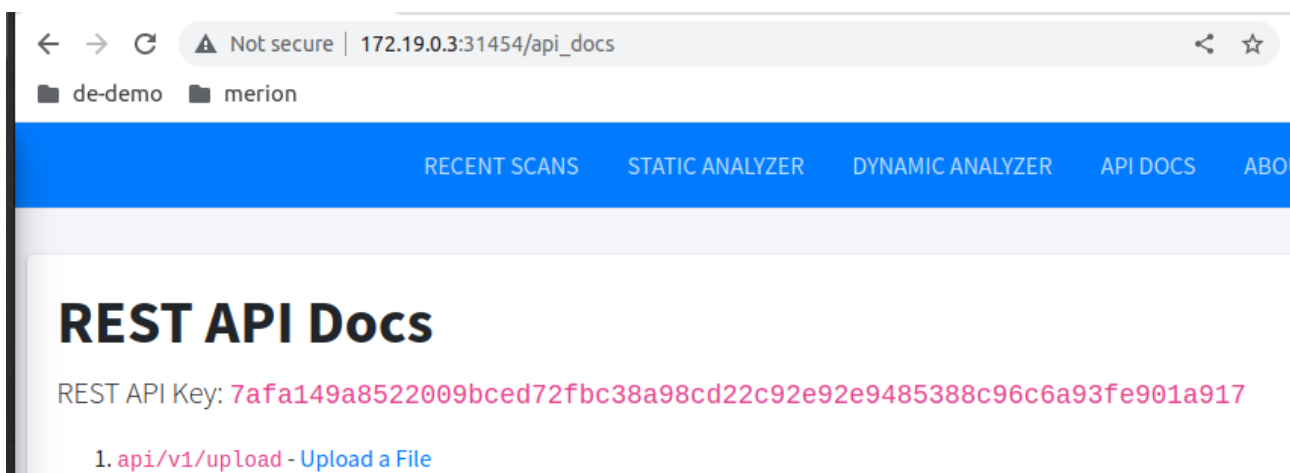
The screenshot shows the Kiali console with the 'Graph' section selected in the sidebar. The sidebar menu includes 'Overview', 'Graph', 'Applications', 'Workloads', 'Services', and 'Istio Config'. The main content area shows the 'default' namespace card, which displays '1 Label', 'Istio Config' status, and 'Applications' count. The 'Istio Config' status is 'N/A' and the 'Applications' count is '0'.

Namespace	Labels	Istio Config	Applications
default	1 Label	N/A	0

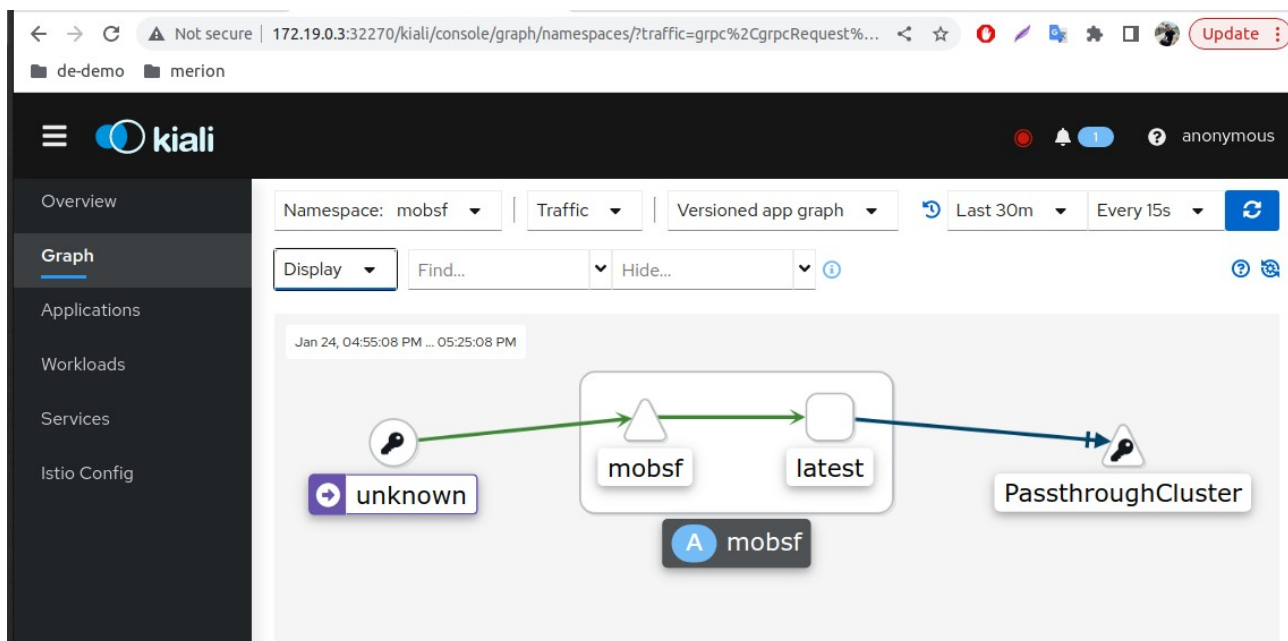
Далее выберем в "Namespace" значение "mobsf", в периоде (справа сверху) выберем "last 30m":



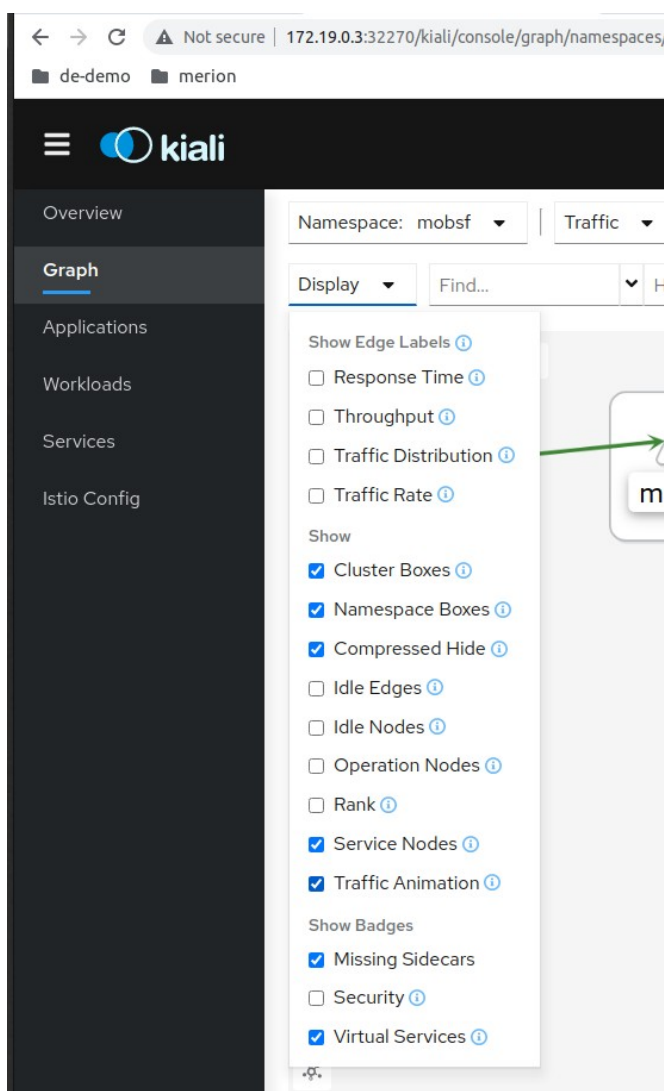
Для полноты визуализации перейдем в веб-интерфейс инструмента mobsf и симулируем полезную нагрузку (перехода по разным вкладкам будет достаточно):



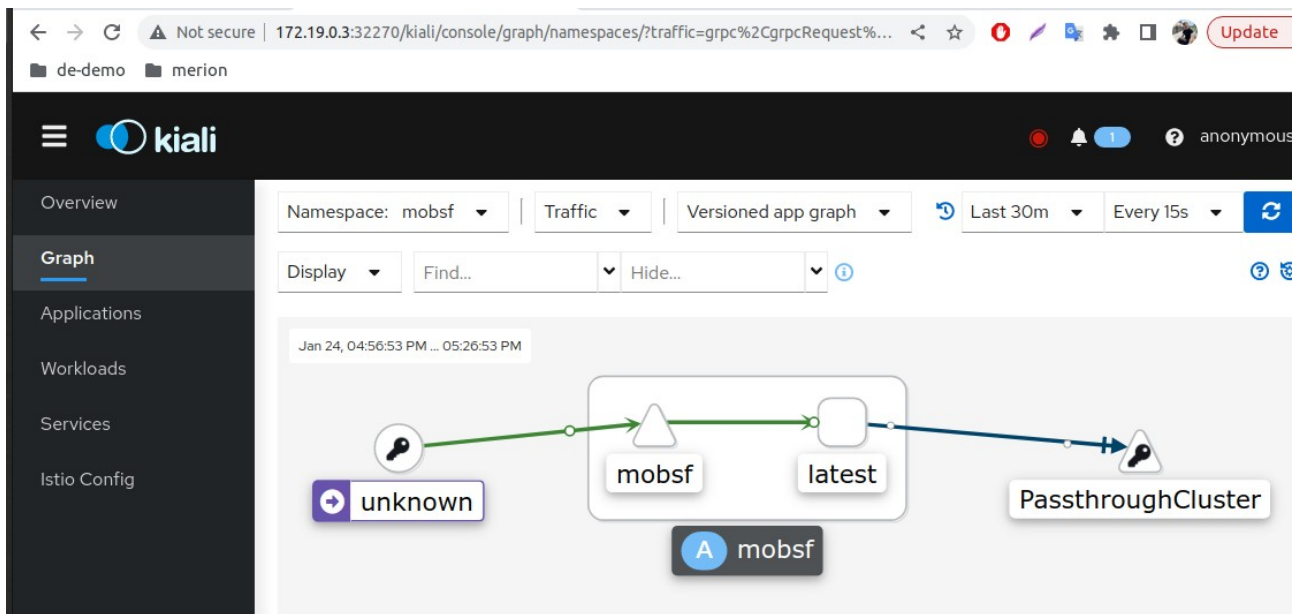
Теперь вернемся в веб-интерфейс дашборда kiali и немного подождем:



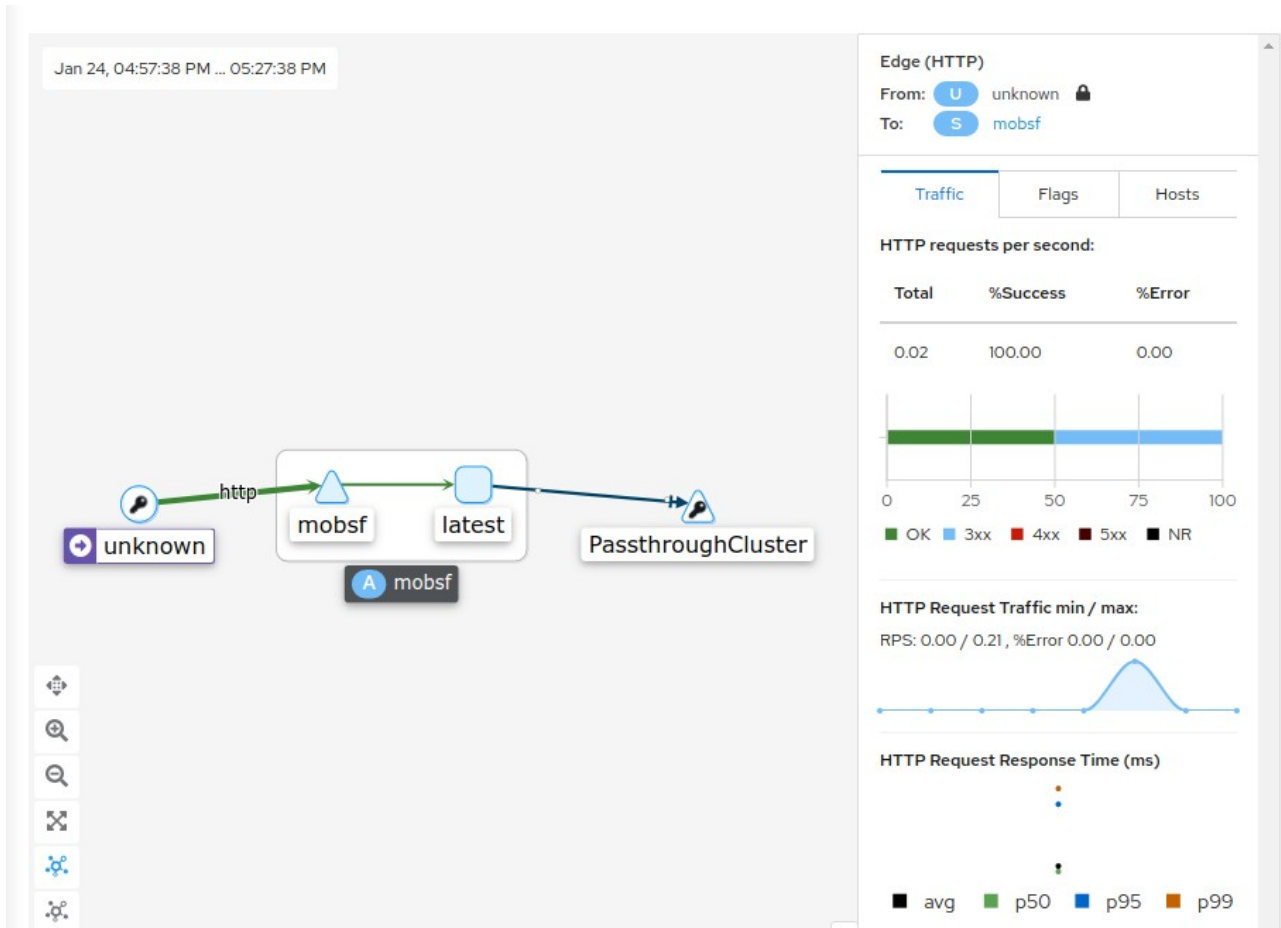
Для красоты и наглядности можем указать в выпадающем списке "Display" анимацию трафика:



Теперь в нашем графе визуализируется взаимодействие с сервисом внутри service mesh и анимируется прохождение трафика:

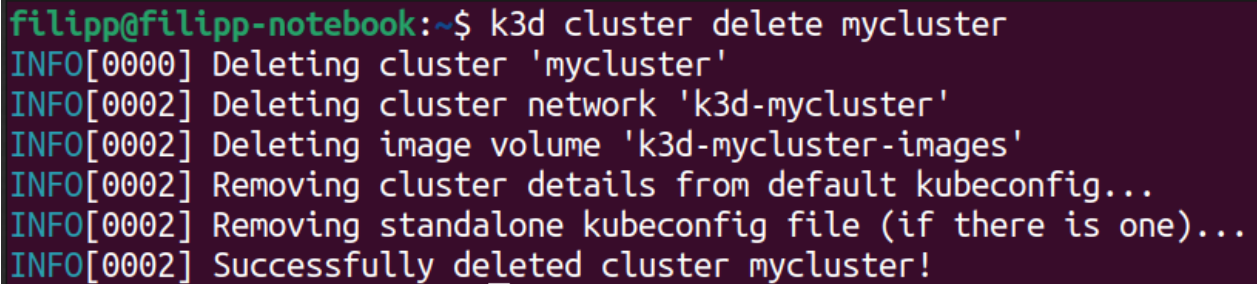


Для получения подробной информации об http трафике нажмите на ребро (edge) или вершину (vertex) графа:



Практическая работа по развертыванию istio service mesh и добавлению в нее сервисов закончена. Для завершения работы можно вернуться в терминал и выполнить команду по уничтожению кластера k3d:

```
$ k3d cluster delete mycluster
```

A terminal window with a dark purple background. The prompt is 'filipp@filipp-notebook:~\$'. The command 'k3d cluster delete mycluster' has been executed. The output consists of six lines of log messages, each starting with 'INFO[0002]'. The messages indicate the deletion of the cluster, its network, image volume, and details from kubeconfig, followed by a confirmation of successful deletion.

```
filipp@filipp-notebook:~$ k3d cluster delete mycluster
INFO[0000] Deleting cluster 'mycluster'
INFO[0002] Deleting cluster network 'k3d-mycluster'
INFO[0002] Deleting image volume 'k3d-mycluster-images'
INFO[0002] Removing cluster details from default kubeconfig...
INFO[0002] Removing standalone kubeconfig file (if there is one)...
INFO[0002] Successfully deleted cluster mycluster!
```

--