K8s monitoring

PROMETHEUS - GRAFANA

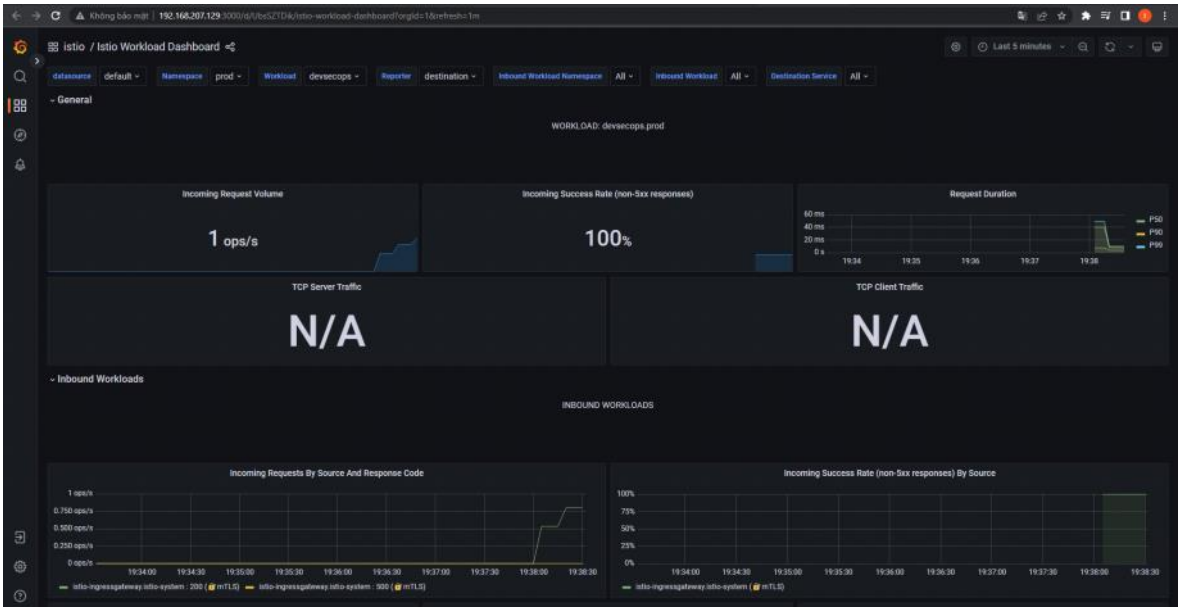
istio_requests_total is a COUNTER that aggregates request totals between Kubernetes workloads, and groups them by response codes, response flags and security policy. It is used to record the total number of requests handled by an Istio proxy. This metric is available only for HTTP, HTTP/2, and GRPC traffic. For TCP traffic there is no requests_total metric because it would be hard to say what to define as a request[3].

Reference:
https://istio.io/latest/zh/docs/reference/config/metrics/.
https://istio.io/latest/docs/tasks/observability/metrics/querying-metrics/

In grafana, Go to istio - Istio Workload Dashboard.
Run bash script below to test connection

```
while true; do curl -s localhost:30886/increment/22;  sleep 1; done
```

And you can see metric display in dashboard like below.



FALCO

Falco is the open source standard for runtime security for hosts, containers, Kubernetes and the cloud. Get real-time visibility into unexpected behaviors, config changes, intrusions, and data theft.

Install falco:
https://falco.org/docs/getting-started/installation/

Run falco:

```
falco
```

Now we create nginx and execute bash in nginx to test falco:

```
sudo kubectl run nginx --image nginx
pod/nginx created

sudo kubectl get pod nginx
NAME   READY  STATUS   RESTARTS  AGE
nginx  1/1    Running  0         13s

sudo kubectl exec -it nginx -- bash
```

```
root@dai:/home/dai$ falco
Thu Apr 13 16:25:03 2023: Falco version: 0.34.1 (x86_64)
Thu Apr 13 16:25:03 2023: Falco initialized with configuration file: /etc/falco/falco.yaml
Thu Apr 13 16:25:03 2023: Loading rules from file /etc/falco/falco_rules.yaml
Thu Apr 13 16:25:03 2023: Loading rules from file /etc/falco/falco_rules.local.yaml
Thu Apr 13 16:25:03 2023: The chosen syscall buffer dimension is: 8388608 bytes (8 MBs)
Thu Apr 13 16:25:03 2023: Starting health webserver with threadiness 8, listening on port 8765
Thu Apr 13 16:25:03 2023: Enabled event sources: syscall
Thu Apr 13 16:25:03 2023: Opening capture with Kernel module



16:35:10.331044274: Error File below a monitored directory opened for writing (user=root user_loginuid=1000 command=nano /
usr/local/bin/startup_script.sh pid=206288 file=/usr/local/bin/.startup_script.sh.swp parent=sudo pcmdline=sudo nano /usr/
local/bin/startup_script.sh gparent=sudo container_id=host image=<NA>)
16:42:15.946568973: Notice A shell was spawned in a container with an attached terminal (user=root user_loginuid=-1 k8s_ng
inx_nginx_default_e5660bc5-377e-435b-8e91-623859ddeaf0_0 (id=aaeab9af4dbd) shell=bash parent=runc cmdline=bash pid=210334
terminal=34816 container_id=aaeab9af4dbd image=nginx)
```

Comeback falco and we see alert a shell was spawned.

And we can see rule list in falco_rules.yaml

```
cat /etc/falco/falco_rules.yaml | grep -i "A shell was spawned in a container with an attached terminal"
    A shell was spawned in a container with an attached terminal (user=%user.name user_loginuid=%user.loginuid %
container.info



cat /etc/falco/falco_rules.yaml | grep -i "A shell was spawned in a container with an attached terminal" -A15 -B20
- rule: System user interactive
  desc: an attempt to run interactive commands by a system (i.e. non-login) user
  condition: spawned_process and system_users and interactive and not user_known_system_user_login
  output: "System user ran an interactive command (user=%user.name user_loginuid=%user.loginuid command=%proc.cmdline
pid=%proc.pid container_id=%container.id image=%container.image.repository)"
  priority: INFO
  tags: [host, container, users, mitre_execution, T1059]
# In some cases, a shell is expected to be run in a container. For example, configuration
# management software may do this, which is expected.
- macro: user_expected_terminal_shell_in_container_conditions
  condition: (never_true)
- rule: Terminal shell in container
  desc: A shell was used as the entrypoint/exec point into a container with an attached terminal.
  condition: >
    spawned_process and container
    and shell_procs and proc.tty != 0
    and container_entrypoint
    and not user_expected_terminal_shell_in_container_conditions
  output: >
    A shell was spawned in a container with an attached terminal (user=%user.name user_loginuid=%user.loginuid %
container.info
    shell=%proc.name parent=%proc.pname cmdline=%proc.cmdline pid=%proc.pid terminal=%proc.tty container_id=%
container.id image=%container.image.repository)
  priority: NOTICE
  tags: [container, shell, mitre_execution, T1059]
# For some container types (mesos), there isn't a container image to
# work with, and the container name is autogenerated, so there isn't
# any stable aspect of the software to work with. In this case, we
# fall back to allowing certain command lines.
- list: known_shell_spawn_cmdlines
  items: [
    '"sh -c uname -p 2> /dev/null"',
    '"sh -c uname -s 2>&1"',
    '"sh -c uname -r 2>&1"',
    '"sh -c uname -v 2>&1"',
```

This command displays the same lines as the first command but also includes the 15 lines after and 20 lines before the line containing the text. This is useful for understanding the context of the rule, as it also shows the related rules and conditions, such as the "user_expected_terminal_shell_in_container_conditions" macro, which defines the conditions when a shell in a container is expected and should not trigger the rule. The command also includes the list of known command lines that are allowed for some container types.

The falco rules listed are used to monitor and alert on certain events happening within a containerized environment.
The first rule, "System user interactive", is triggered when a non-login system user attempts to run interactive commands. The condition for this rule to trigger is when a process is spawned, the user is a system user, the command is interactive, and the user is not a known system user login. The output of this rule includes information such as the user name, login UID, command, process ID, container ID, and container image repository. The priority for this rule is set to INFO and the associated tags include host, container, users, mitre_execution, and T1059.
The second rule, "Terminal shell in container", is triggered when a shell is used as the entry point or execution point within a container that has an attached terminal. The conditions for this rule include the spawning of a process within a container, the process is a shell process, the process has a TTY attached, the container has an entry point, and the user did not expect a terminal shell in the container. The output of this rule includes information such as the user name, login UID, container information, shell process name, parent process name, command line, process ID, terminal, container ID, and container image repository. The priority for this rule is set to NOTICE and the associated tags include container, shell, mitre_execution, and T1059.
Lastly, the rule defines a list of known shell spawn command lines that are allowed for certain container types where there is no container image to work with or where the container name is autogenerated.

Reference:
https://falco.org/docs/rules/

HELM - FALCO

Install helm:

```
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null
sudo apt-get install apt-transport-https --yes
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg]
https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

Install falco-falcosidekick:

```
kubectl create namespace falco
helm repo add falcosecurity https://falcosecurity.github.io/charts
helm repo update

helm install falco falcosecurity/falco \
--set falcosidekick.enabled=true \
--set falcosidekick.webui.enabled=true \
-n falco

Or
helm install falco falcosecurity/falco --set ebpf.enabled=true  --set falcosidekick.enabled=true  --set
falcosidekick.webui.enabled=true
```

If you see falco UI error, you can create PV, PVC file.

```
PersistentVolume.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: falcosidekick-ui-redis-data-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
  hostPath:
    path: "/mnt/data"


PersistentVolumeClaim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: falco-falcosidekick-ui-redis-data-falco-falcosidekick-ui-redis-0
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: manual

kubectl apply -f persistentVolume.yaml -n falco
kubectl apply -f persistentVolumeClaim.yaml -n falco

or
kubectl create pv -- falco-falcosidekick-ui-redis-data --capacity 1Gi --storage-class standard
kubectl create pvc --name falco-falcosidekick-ui-redis-data-falco-falcosidekick-ui-redis-0 --claim-name falco-falcosidekick-
```

```
ui-redis-data-falco-falcosidekick-ui-redis-0 --storage-class standard
```

If pv and pvc not bound, update pvc spec

```
kubectl patch pvc falco-falcosidekick-ui-redis-data-falco-falcosidekick-ui-redis-0 -p '{"spec": {"volumeName": "falcosidekick-ui-redis-data-pv"}}' -n falco
```

Check bound status

```
kubectl -n falco describe pvc falco-falcosidekick-ui-redis-data-falco-falcosidekick-ui-redis-0
```

Port-forwarding:

```
kubectl -n falco port-forward deploy/falco-falcosidekick-ui 2802:2802 --address 192.168.207.129
```

Login falco-UI with user,password: admin