Practice Session-09: Monitoring with Prometheus

In DevOps culture, developers require seamless integration of application and business metrics into their infrastructure, as they play a more active role in the CI/CD pipeline and have increased autonomy in operations and debugging. In this practice session, you're going to work with monitoring tools for both infrastructure and application.

Make sure that you have already gone through Lab-08.

Prerequisite

- Basic knowledge on k8s
- Basic knowledge on YAML
- Familiar with GitLab webIDE or Git commands

1. Environment Setup

Step 1.1. Create a Gitlab Project

1. In Gitlab, create a blank project with name `prac09-monitoring-prometheus` under your group `Devops2024fall/students/devops2024Fall-<lastname>-<studyCode>`
2. Inside your *controller* VM, create a directory for this practice session and change the directory:
   ```
   mkdir ~/prac09 && cd ~/prac09
   ```
3. Store your ID of the project `prac09-monitoring-prometheus` in a file using this command: `echo "<your_project_ID>" > ~/prac09/project_id.txt`
4. In the following exercises and steps, you may use Gitlab `WebIDE` to add, remove, and edit the files and folders in your Gitlab project.

Step 1.2. Basic requirements

1. We will monitor three VMs and services in this session.
   - *Controller* VM
   - *K3snode1* VM
   - *K3snode2* VM

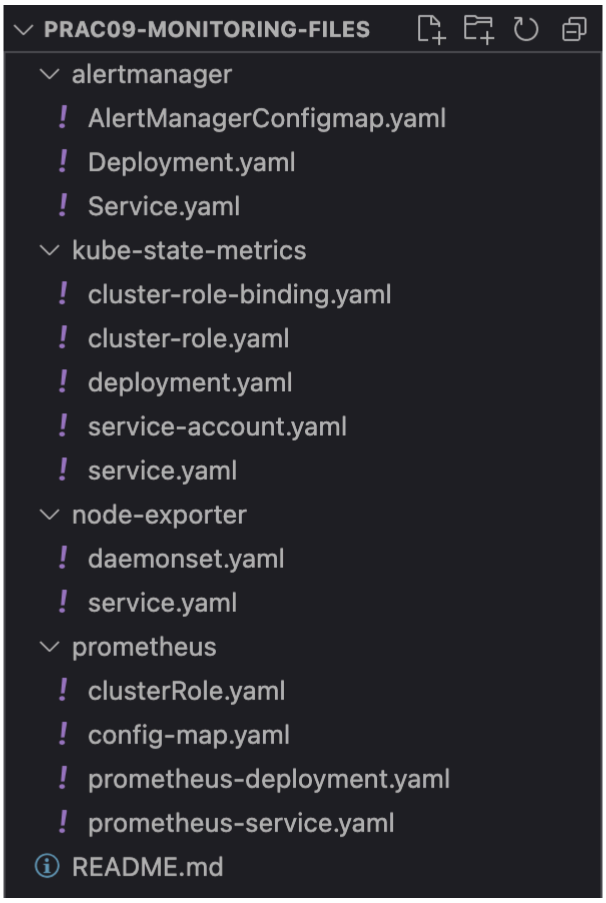
2. We have setup files in this Gitlab Repo, which are required to set up four services: Prometheus, Alertmanager, node exporter, and kube-state-metrics.

1. **Prometheus** is a high-scalable open-source monitoring framework. It provides out-of-the-box monitoring capabilities for the Kubernetes container orchestration platform. Also, In the observability space, it is gaining huge popularity as it helps with metrics and alerts. The prac09-monitoring-prometheus/prometheus has four YAML files.
    - clusterRole.yaml: Prometheus uses Kubernetes APIs to read all the available metrics from Nodes, Pods, Deployments, etc. For this reason, we need to create an RBAC policy with read access to required API groups and bind the policy to the monitoring namespace.
    - config-map.yaml: All configurations for Prometheus are part of the *prometheus.yaml* file and all the alert rules for Alertmanager are configured in *prometheus.rules*.
        - prometheus.yaml: This is the main Prometheus configuration which holds all the scrape configs, service discovery details, storage locations, data retention configs, etc
        - prometheus.rules: This file contains all the Prometheus alerting rules
    - prometheus-deployment.yaml: This is used to create prometheus deployment along with volumes, configs, etc.
    - prometheus-service.yaml: This is used to create a service for the prometheus dashboard to access for the user. We fix the nodeport address 30000 to access the Prometheus.
2. **Node exporter** is an official Prometheus exporter for capturing all the Linux system-related metrics. It collects all the hardware and OS level metrics that are exposed by the kernel. Here we are using it to collect all three VMs metrics. The prac09-monitoring-prometheus/node-exporter has two yaml files.
   daemonset.yaml: This is used to create node exporter service as a daemon set which will scrape the vm metrics from three vms.
   service.yaml: This is used to create a service for node exporters.

3. **AlertManager** is an open-source alerting system that works with the Prometheus Monitoring system. Here we send the alerts to Zulip (https://zulip.cs.ut.ee).
   AlertManagerConfigmap.yaml: This is used to create a configmap with a set of alert rules and Zulip integrations.
   Deployment.yaml: This is used to create an alertmanager deployment.

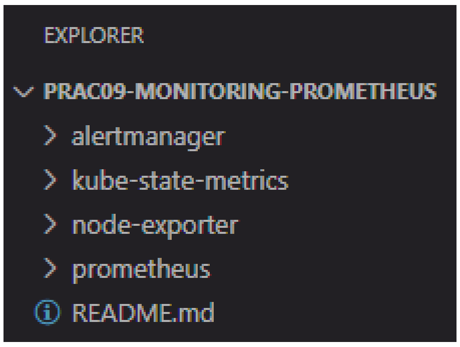
4. **Kube State metrics** is a service that talks to the Kubernetes API server to get all the details about all the API objects like deployments, pods, daemonsets, Statefulsets, etc.

We have all the files required to set up four services. Please have a look at each of the files and understand the structure.



```
PRAC09-MONITORING-FILES
  alertmanager
    AlertManagerConfigmap.yaml
    Deployment.yaml
    Service.yaml
  kube-state-metrics
    cluster-role-binding.yaml
    cluster-role.yaml
    deployment.yaml
    service-account.yaml
    service.yaml
  node-exporter
    daemonset.yaml
    service.yaml
  prometheus
    clusterRole.yaml
    config-map.yaml
    prometheus-deployment.yaml
    prometheus-service.yaml
  README.md
```

3. Download all the folders (i.e. prometheus, alertmanager, node-exporter, and kube-state-metrics) from our shared Gitlab project https://gitlab.cs.ut.ee/devops2024-fall/all-practice-sessions-pub/prac09-monitoring-files.git and upload to your Gitlab Project and Commit with a message *“Required project files added”*. Your Gitlab project should have the following structure:



```
EXPLORER
PRAC09-MONITORING-PROMETHEUS
  alertmanager
  kube-state-metrics
  node-exporter
  prometheus
  README.md
```

4. We need a shell runner for this project, there are two options to achieve this. You can use any one option, but recommended is to create a brand new runner(option 1) because some of you created a runner with sudo in the previous lab, which may create problems while running the helm commands.
   Option 1: Create a new shell runner

- You can create the runner for `prac09-monitoring-prometheus` with *tag: monitoring* under Settings→CI/CD→Runners, and execute the command in the *controller* VM to create the runner.
    1. Enter the GitLab instance URL: `https://gitlab.cs.ut.ee/`
    2. Enter a name for the runner: `monitoring`
    3. Enter an executor: `shell`

Option 2: Use existing shell runner
- Enable the shell runner for this project. You can do this at *Settings→Runners→Other available runners*. Here, look for your shell runner and click on *Enable Runner for this project.*
- Update the tags: add the tag *monitoring*

5. In further steps, if you get the pipeline jobs in the *Pending* state, then try to run the runner in the VM using the command
```
gitlab-runner run shell
```

Step 1.3. Notes

Below some of the instructions are descriptive. That means you may get errors while executing the given minimal version of the commands. You need to investigate and fix those errors. For this, you may need to google and debug the error by yourself.

2. Setting up Prometheus and associated services on *controller* VM

In this exercise, we are going to set up the following services in the ***controller*** VM. We will install it using the helm. The helm chart is from the Prometheus community project.
We will install the following services for us:

Service Name	Purpose	Port numbers	For more information
Prometheus-server	It is an open-source monitoring solution that pulls the performance metrics from the remote VMs and pods based on specific time intervals. Further, trigger the alerts based on PromQL queries	30000	https://prometheus.io/
Alertmanager	Send notification to webhook endpoints based on trigger from Prometheus events	31000	https://prometheus.io/docs/alerting/latest/alertmanager/
Kube-state-metrics	Kube-state-metrics for orchestration and cluster level metrics: deployments, pod metrics, resource reservation, etc.	8081	https://github.com/google/cadvisor
node-exporter	Pull the VM metrics such as CPU, Memory, Disk, and Network utilization metrics.	9100	https://prometheus.io/docs/guides/node-exporter/

In the following steps, we assume that you are using `WebIDE` to edit your GitLab project.

Step 2.1. Setting up of Prometheus and associated services

1. You can use Web IDE
2. Use hosts.yaml from the previous practice session
3. Create Ansible script to install prometheus and its services on k8s cluster

Filename: `prometheus-ansible.yaml`

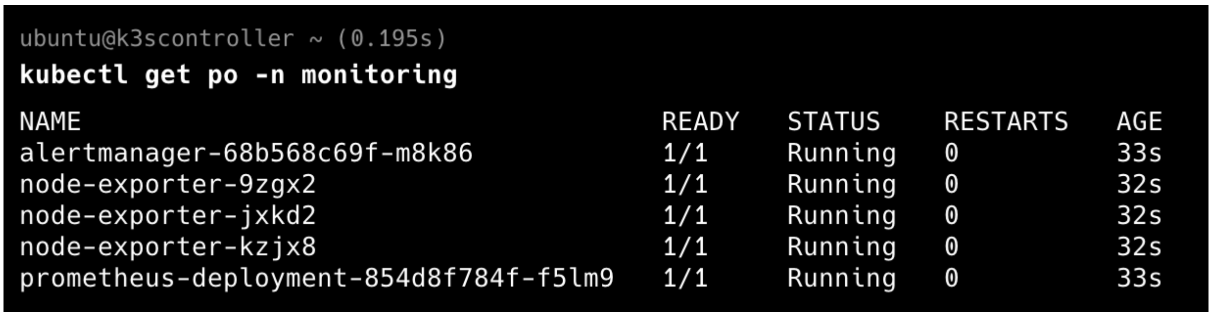
```
---
- name: create
  hosts: k3s_controller
  gather_facts: true
  # become: true
  become_user: ubuntu

  tasks:

    - name: create a namespace monitoring
      command: "k3s kubectl create namespace monitoring"


    - name: Install prometheus
      command: "k3s kubectl apply  -f prometheus/"

    - name: Install alertmanager
      command: "k3s kubectl apply  -f alertmanager/"

    - name: Install kube-state-metrics
      command: "k3s kubectl apply  -f kube-state-metrics/"

    - name: Install node-exporter
      command: "k3s kubectl apply  -f node-exporter/"
```

4. Create a `.gitlab-ci.yml` file inside the root directory of your project with a job to create monitoring services in the *controller* **VM.** Here, we will define the commands to install the helm chart.

Filename: `.gitlab-ci.yml`

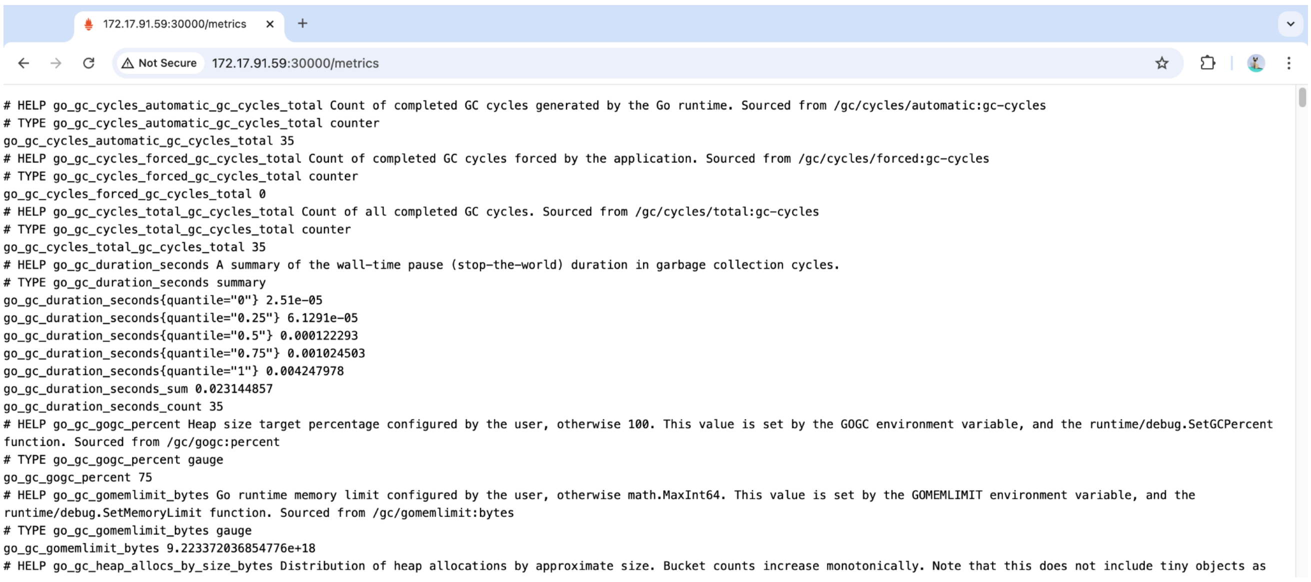
```
stages:
  - monitor-stage
monitor-job:
  tags:
      - monitoring
  stage: monitor-stage
  script:
      - ansible-playbook prometheus-ansible.yaml -i hosts.yaml
  after_script:
    - kubectl get po -n monitoring
```

5. Commit and push the changes with message *`Step 2.1 added ansible script to install prometheus services`*. Further, watch the jobs running in **CI-CD-->Pipelines.** After success, check for all services running in the *controller* VM using logs of the pipeline (You have mentioned *after_script* to display the pods,svc). If your script is installed then you should see the output as shown below



```
ubuntu@k3scontroller ~ (0.195s)
kubectl get po -n monitoring
NAME                                  READY   STATUS    RESTARTS   AGE
alertmanager-68b568c69f-m8k86         1/1     Running   0          33s
node-exporter-9zgx2                   1/1     Running   0          32s
node-exporter-jxkd2                   1/1     Running   0          32s
node-exporter-kzjx8                   1/1     Running   0          32s
prometheus-deployment-854d8f784f-f5lm9   1/1     Running   0          33s
```

6. You may also use any web browser and visit the `/metrics` endpoints of prometheus `http://<controller_VM_EXT_IP>:30000/metrics`



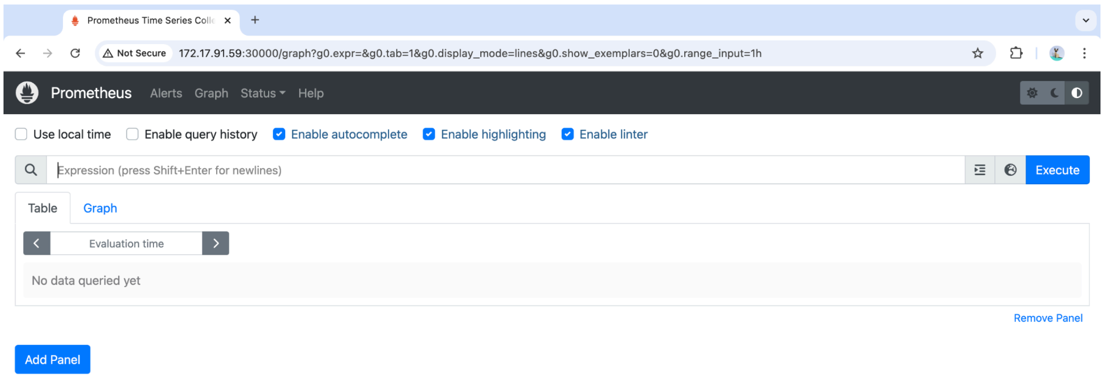
```
# HELP go_gc_cycles_automatic_gc_cycles_total Count of completed GC cycles generated by the Go runtime. Sourced from /gc/cycles/automatic:gc-cycles
# TYPE go_gc_cycles_automatic_gc_cycles_total counter
go_gc_cycles_automatic_gc_cycles_total 35
# HELP go_gc_cycles_forced_gc_cycles_total Count of completed GC cycles forced by the application. Sourced from /gc/cycles/forced:gc-cycles
# TYPE go_gc_cycles_forced_gc_cycles_total counter
go_gc_cycles_forced_gc_cycles_total 0
# HELP go_gc_cycles_total_gc_cycles_total Count of all completed GC cycles. Sourced from /gc/cycles/total:gc-cycles
# TYPE go_gc_cycles_total_gc_cycles_total counter
go_gc_cycles_total_gc_cycles_total 35
# HELP go_gc_duration_seconds A summary of the wall-time pause (stop-the-world) duration in garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.51e-05
go_gc_duration_seconds{quantile="0.25"} 6.1291e-05
go_gc_duration_seconds{quantile="0.5"} 0.000122293
go_gc_duration_seconds{quantile="0.75"} 0.001024503
go_gc_duration_seconds{quantile="1"} 0.004247978
go_gc_duration_seconds_sum 0.023144857
go_gc_duration_seconds_count 35
# HELP go_gc_gogc_percent Heap size target percentage configured by the user, otherwise 100. This value is set by the GOGC environment variable, and the runtime/debug.SetGCPercent
function. Sourced from /gc/gogc:percent
# TYPE go_gc_gogc_percent gauge
go_gc_gogc_percent 75
# HELP go_gc_gomemlimit_bytes Go runtime memory limit configured by the user, otherwise math.MaxInt64. This value is set by the GOMEMLIMIT environment variable, and the
runtime/debug.SetMemoryLimit function. Sourced from /gc/gomemlimit:bytes
# TYPE go_gc_gomemlimit_bytes gauge
go_gc_gomemlimit_bytes 9.223372036854776e+18
# HELP go_gc_heap_allocs_by_size_bytes Distribution of heap allocations by approximate size. Bucket counts increase monotonically. Note that this does not include tiny objects as
```

AGS-1: At this point, Nagios will check all the services: Prometheus, alertmanager, kube-state-metrics, and node-exporter.

Step 2.2. Working with Prometheus dashboard

In this task, you're going to explore Prometheus' service for monitoring the VMs and container metrics pulled from node-exporter and cadvisor. Prometheus provides a functional query language called **PromQL** (Prometheus Query Language) that lets the user select and aggregate time series data in real-time (Ref: https://prometheus.io/docs/prometheus/latest/querying/basics/)

- Now, access the Prometheus service in the browser at `http://controller_VM_EXTERNAL_IP:30000`. A sample output is shown below.


Prometheus Time Series Colle

Not Secure 172.17.91.59:30000/graph?g0.expr=&g0.tab=1&g0.display_mode=lines&g0.show_exemplars=0&g0.range_input=1h

Prometheus Alerts Graph Status Help

☐ Use local time ☐ Enable query history ☑ Enable autocomplete ☑ Enable highlighting ☑ Enable linter

Expression (press Shift+Enter for newlines) Execute

Table Graph

Evaluation time

No data queried yet

Remove Panel

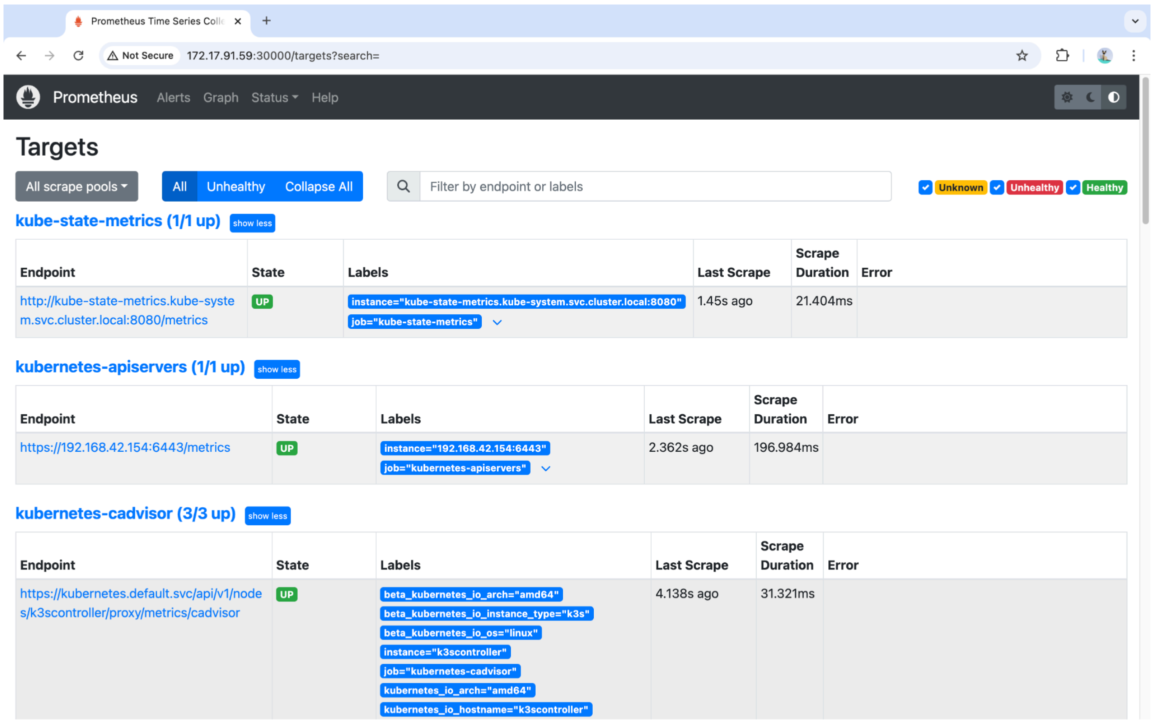
Add Panel

You can have a look at the Prometheus dashboard to understand the different tabs.

Alerts: List of alerts defined to trigger notifications, when defined events occur such as *application container attends a downtime* (This will be precisely explored further in the below Exercises)

Graph: It has a Query browser to write the PromQL queries to get the metrics scraped from the *cadvisor* and *node exporter*.

Status: This has a set of sub-tabs. For example, **Configuration** describes the configuration file `prometheus.yml` with targets. Now go to **Status-->Targets**, which will display a set of targets (endpoints) from where it pulls the resource utilization metrics. (If you see the node-exporter and cAdvisor on other VMs are down with red color then those nodes or services are not working or are down)


Prometheus Time Series Colle

Not Secure 172.17.91.59:30000/targets?search=

Prometheus Alerts Graph Status Help

Targets

All scrape pools All Unhealthy Collapse All Filter by endpoint or labels ☑ Unknown ☑ Unhealthy ☑ Healthy

kube-state-metrics (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://kube-state-metrics.kube-system.svc.cluster.local:8080/metrics	UP	instance="kube-state-metrics.kube-system.svc.cluster.local:8080" job="kube-state-metrics"	1.45s ago	21.404ms	

kubernetes-apiservers (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://192.168.42.154:6443/metrics	UP	instance="192.168.42.154:6443" job="kubernetes-apiservers"	2.362s ago	196.984ms	

kubernetes-cadvisor (3/3 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://kubernetes.default.svc/api/v1/nodes/k3scontroller/proxy/metrics/cadvisor	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_instance_type="k3s" beta_kubernetes_io_os="linux" instance="k3scontroller" job="kubernetes-cadvisor" kubernetes_io_arch="amd64" kubernetes_io_hostname="k3scontroller"	4.138s ago	31.321ms	

3. Working with PromQL for k8s node and pod metrics

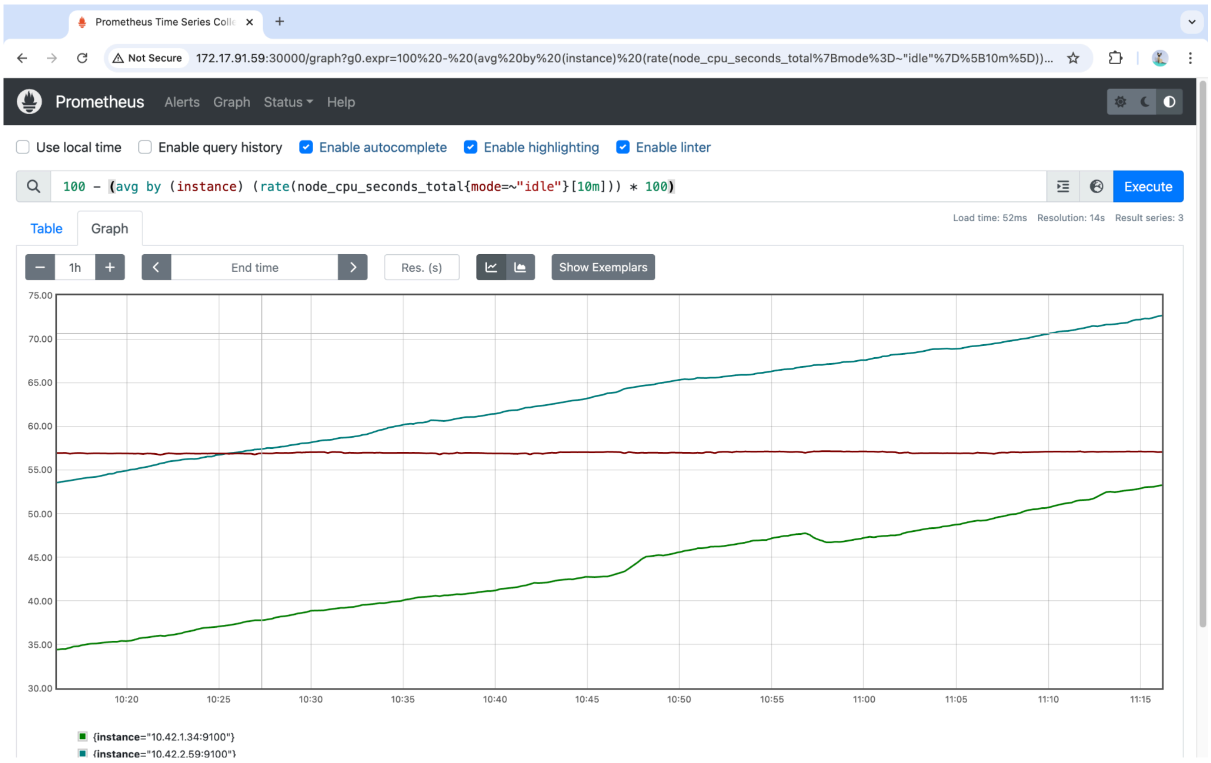
Step 3.1. Working with K8s node metrics.

Now, let us write PromQL queries to see the current resource utilization of the **controller** VM.

1. Queries are written and modified under **Graph-->Expression** and executed by clicking on **Execute**. There are two types of outputs(Table and Graph) and see the graph windows for output.
2. Now let us calculate the metrics for CPU utilization of ***all nodes.***

Metric Name	PromQL query
Average CPU utilization	100 - (avg by (instance) (rate(node_cpu_seconds_total{mode=~"idle"}[10m])) * 100)

   - Here `rate(... [10m])` calculates the per-second average rate of increase of the time series in the range vector.
   - `node_cpu_seconds_total` metric comes from `/proc/stat` of the nodes and tells us how many seconds each CPU spent doing each type of work:
     - user: The time spent in userland
     - system: The time spent in the kernel
     - iowait: Time spent waiting for I/O
     - idle: Time the CPU had nothing to do
     - irq&softirq: Time servicing interrupts
     - guest: If you are running VMs, the CPU they use
     - steal: If you are a VM, time other VMs "stole" from your CPUs
   - Overall, we calculate the idle time of all the CPUs in %, and subtracting by 100 gives us the busy time serving all the activities(iowait,user,...) on intervals of every 10 minutes.
3. Now run the query in **Graph-->Expression** and sample output of first query "Average CPU utilization" is shown below:(Your graph may be different, it depends on the CPU Utilisation)



172.17.91.59:30000/graph?g0.expr=100%20-%20(avg%20by%20(instance)%20(rate(node_cpu_seconds_total%7Bmode%3D~"idle"%7D%5B10m%5D))...

Use local time [ ] · Enable query history [ ] · Enable autocomplete [✓] · Enable highlighting [✓] · Enable linter [✓]

```
100 - (avg by (instance) (rate(node_cpu_seconds_total{mode=~"idle"}[10m])) * 100)
```

Load time: 52ms Resolution: 14s Result series: 3

🟩 {instance="10.42.1.34:9100"}
🟩 {instance="10.42.2.59:9100"}

4. Similarly, calculate the memory, disk, network receive, and transmit

Metric Name	**PromQL query**
Average Memory utilization	100 * (1 - ((avg_over_time(node_memory_MemFree_bytes{job="node-exporter"}[5m]) + avg_over_time(node_memory_Cached_bytes{job="node-exporter"}[5m]) + avg_over_time(node_memory_Buffers_bytes{job="node-exporter"}[5m])) / avg_over_time(node_memory_MemTotal_bytes{job="node-exporter"}[5m])))
Disk utilization	(avg(node_filesystem_size_bytes{device!="rootfs"}) by (instance) - avg(node_filesystem_free_bytes{device!="rootfs"}) by (instance)) / avg(node_filesystem_size_bytes{device!="rootfs"}) by (instance)
Network Receive packets per second	rate(node_network_receive_packets_total{job="node-exporter"}[10m])
Network Transmit packets per second	rate(node_network_transmit_packets_total{job="node-exporter"}[10m])

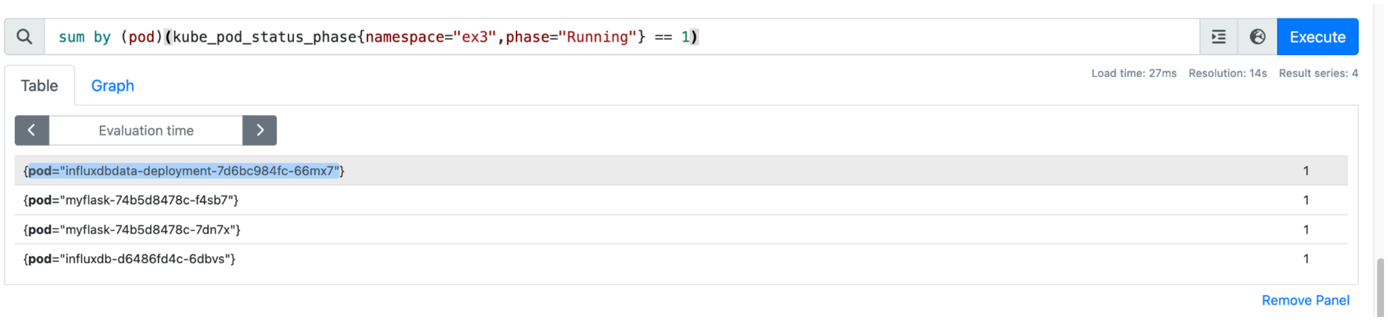
Step 3.2. Working with Pod metrics.

We will calculate the pod health and metrics using PromQL

1. Let us work with pod status, which is used to check the current status of the deployment or the pods.
   - Check the running pods under namespace ex3 using the query. Here kube_pod_status_phase will give you the status {running, failed, pending}

Metric Name	**PromQL query**

Running	sum by (pod)(kube_pod_status_phase{namespace="ex3",phase="Running"} == 1)



```
sum by (pod)(kube_pod_status_phase{namespace="ex3",phase="Running"} == 1)
```

Load time: 27ms Resolution: 14s Result series: 4

```
{pod="influxdbdata-deployment-7d6bc984fc-66mx7"}    1
{pod="myflask-74b5d8478c-f4sb7"}                    1
{pod="myflask-74b5d8478c-7dn7x"}                    1
{pod="influxdb-d6486fd4c-6dbvs"}                    1
```

- Check the running status of the influxdb pod, here you have to add `pod=~"influxdb-.*"` inside the query `{}` and remove the `sum by (pod)`

Metric Name	**PromQL query**
Running	(kube_pod_status_phase{namespace="ex3",pod=~"influxdb-.*",phase="Running"} == 1)

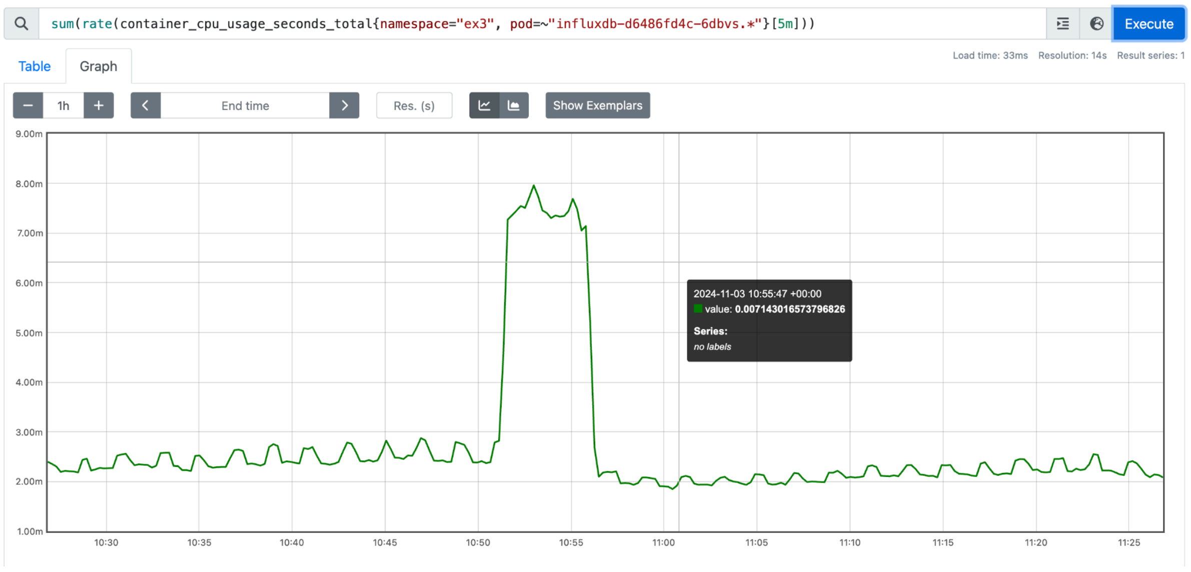
- Get the pods, that are restarting or in CrashLoopback error with the following query (Not to worry if you get 0 pods in this query result)

```
sum by
(deployment)(changes(kube_pod_status_ready{namespace="ex
3",condition="true"}[5m]))
```

2. Now let us calculate the metrics for CPU utilization of ***influxdb pod***

Metric Name	**PromQL query**
CPU utilization	sum(rate(container_cpu_usage_seconds_total{namespace="ex3", pod=~"influxdb-.*"}[5m]))

- This query will calculate the rate of CPU usage over the last 5 minutes for a specific pod in *ex3* namespace.
- The following screenshot shows the CPU utilization of the influxdb pod.
  - Here you to change the pod name to your influxdb pod name
  - We are not measuring the CPU usage in terms of % as measured in node metrics. The CPU utilization is measured in milli-units. The letter *m* indicates the "milli" or one-thousandth (1/1000). It is a unit of measurement used to represent values in milli-units or milliseconds. For example, "50mCpuUsage" on the Y-axis means 50 milliCPU seconds, which is equivalent to 0.05 CPU seconds.



```
sum(rate(container_cpu_usage_seconds_total{namespace="ex3", pod=~"influxdb-d6486fd4c-6dbvs.*"}[5m]))
```

Load time: 33ms Resolution: 14s Result series: 1

2024-11-03 10:55:47 +00:00
🟩 value: **0.007143016573796826**

Series:
no labels

3. Similarly, calculate the memory usage of the all the pods under namespace ex3

Metric Name	**PromQL query**
Memory utilization	sum(container_memory_working_set_bytes{namespace="ex3"}) by (pod)

No AGSL for the time being. Its more of a practice of PromQL

4. Working with Prometheus alerts

In this section, you will trigger the alerts to notify the users in a Zulip stream when certain events occur in the monitoring metrics, for example, CPU utilization of the container exceeds 90% then notify the user. In this experiment, you will use *alertmanager* service along with Prometheus.

As an example, you will monitor the downtime of the flask application created in Exercises 3 and send an alert notification to the Zulip topic "**Lab9-Alerts**" if the downtime occurs in the last 10 seconds.

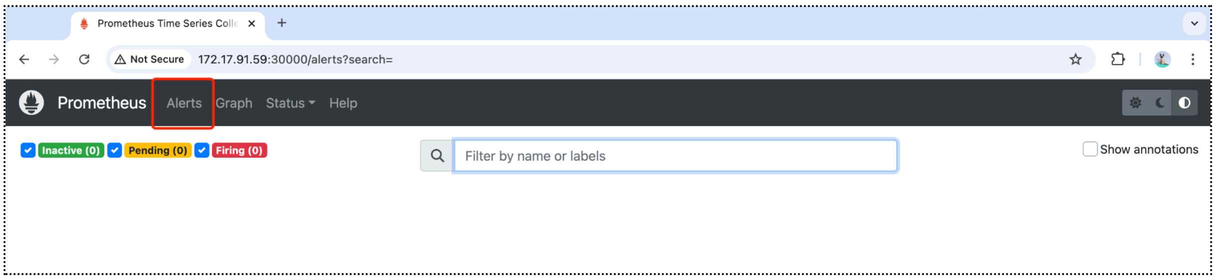
Learn more about Alertmanager: https://prometheus.io/docs/alerting/latest/alertmanager/

Below are some hints to perform this:

The below codes are just for reference. Copy the code at your own risk. We strongly recommend NOT to simply copy-paste the code. Make yourself comfortable with YAML.

Step 4.1 Alerts rules in Prometheus

The alert rules are defined in the `prometheus.yaml` which is defined in *prometheus/config-map.yaml* and the alert and configurations can be seen in the prometheus dashboard as shown below.
Now, you don't have any rules defined now.



Step 4.2 Create alertmanager rules and associated configuration files

1. We will write an alert rule to notify if, influxDB pod is over using the cpu, i.e more than a certain limit.
2. Update a file `config-map.yaml` inside `prometheus` directory (sample code given below) with rules to monitor and trigger. In this file, you should write a PromQL query to monitor the event
   a. `expr: sum(rate(container_cpu_usage_seconds_total{namespace="ex3", pod=~"influxdb-.*"}[1m])) > 0.003584`
   b. Describe the interval to run the expression
   c. You can the annotations and descriptions
3. More about alerting rules: https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/#alerting-rules

Filename: `prometheus/config-map.yaml`

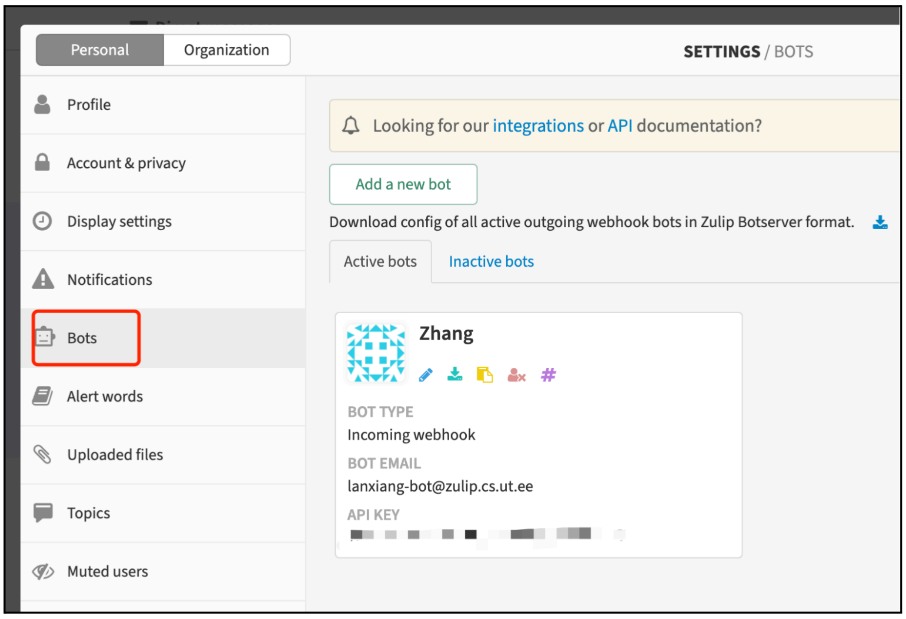
```
.. previous code as it should …
data:
  prometheus.rules: |-
    groups:
    - name: devops demo
      rules:
      - alert: High CPU Usage
        expr: sum(rate(container_cpu_usage_seconds_total{namespace="ex3",
pod=~"influxdb-.*"}[1m])) > 0.003584
        for: 1m
        labels:
          severity: Critical
        annotations:
          summary: High Memory Usage by {{ $labels.pod }}

.. keep the code as it below…
```

4. Update a file `AlertManagerConfigmap.yaml` inside `alertmanager` directory (sample code given below). Here we define the notification endpoint for an alertmanager to send notifications to the end user, for example in our case it is Zulip topic “**Lab9-Alerts**”. This is used by alertmanager service to invoke the notification endpoint when alerts are triggered by Prometheus service.

Filename: `/alertmanager/AlertManagerConfigmap.yaml`

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: alertmanager-config
  namespace: monitoring
data:
  config.yml: |-
    global:
      http_config:
        tls_config:
          insecure_skip_verify: true
    route:
      receiver: "Zulip-notifications"
      group_by: ['alertname', 'priority']
      group_wait: 10s
      repeat_interval: 30m
      routes:
        - receiver: "Zulip-notifications"
          match:
            severity: critical
          group_wait: 10s
          repeat_interval: 1m

    receivers:
    - name: "Zulip-notifications"
      webhook_configs:
      - url:
"https://zulip.cs.ut.ee/api/v1/external/alertmanager?api_key=vwxcdvdfsaE5VnUZejiQHvH7ZiXU
A6Y3&stream=DevOps2024Fall&topic=Lab9-Alerts"
```

5. You have to modify the yellow color highlighted text i.e api_key
   a. To get the api_key from Zulip. Follow the steps below:
      i. In your Zulip account, go to Settings--Personal Settings → Bots



Download config of all active outgoing webhook bots in Zulip Botserver format.

Zhang

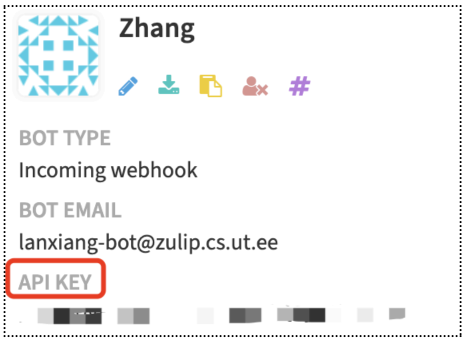
BOT TYPE
Incoming webhook

BOT EMAIL
lanxiang-bot@zulip.cs.ut.ee

API KEY
[redacted]

ii. Create a bot by clicking on **Add a new bot**
   1. Bot Type: **Incoming Webhook**
   2. Name: **Your last name**
   3. Bot email: **Your last name**
   4. Click Add

iii. Now bot is created. Copy the API KEY and update `/alertmanager/AlertManagerConfigmap.yaml`



Zhang

BOT TYPE
Incoming webhook

BOT EMAIL
lanxiang-bot@zulip.cs.ut.ee

API KEY
[redacted]

Inside your controller VM, store your Bot Email in a file using this command: `echo "<your_Bot_email_value>" > ~/prac09/bot_email_value.txt`

Just for your reference:
For more details to get the webhook URL follow the steps here:
https://zulip.com/integrations/doc/alertmanager

6. We have a `prometheus/config-map.yaml` file with the path to the URL endpoint of alertmanager service. (You need not to update anything here. This is just for reference)
   Refer to the following sample yaml code block `prometheus/config-map.yaml`.

Filename: `prometheus/config-map.yaml`

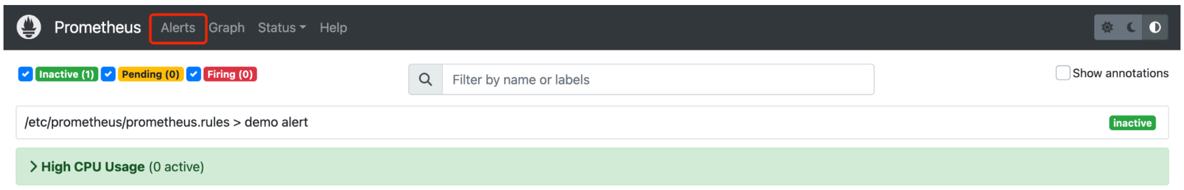
```
### Previous CONTENT

    alerting:
      alertmanagers:
      - scheme: http
        static_configs:
        - targets:
          - "alertmanager.monitoring.svc:9093"


### Previous content
```

7. Once you edit the configmap file, you need to delete and apply the prometheus and alertmanager services. Create an Ansible script **update-prometheus-ansible.yaml** to delete and create the following
   a. Delete prometheus/config-map.yaml (`k3s kubectl delete -f prometheus/config-map.yaml`)
   b. Apply the changes prometheus/config-map.yaml (`k3s kubectl apply -f prometheus/config-map.yaml`)

   c. Delete alertmanager/AlertManagerConfigmap.yaml (`k3s kubectl delete -f alertmanager/AlertManagerConfigmap.yaml`)
   d. Apply the changes alertmanager/AlertManagerConfigmap.yaml (`k3s kubectl apply -f alertmanager/AlertManagerConfigmap.yaml`)
   e. Delete the prometheus deployment (`k3s kubectl delete -f prometheus/prometheus-deployment.yaml`)
   f. Create the prometheus deployment (`k3s kubectl apply -f prometheus/prometheus-deployment.yaml`)
   g. Delete the alertmanager deployment (`k3s kubectl delete -f alertmanager/Deployment.yaml`)
   h. Create the alertmanager deployment (`k3s kubectl apply -f alertmanager/Deployment.yaml`)
8. Update the `.gitlab-ci.yaml` by adding an ansible command to run **update-prometheus-ansible.yaml** in `script`.
9. Commit and push the changes with the message `Step 4.2 added alertmanager services.`

Step 4.3 Test the alert notification

1. Once the above-mentioned job has succeeded, Goto Prometheus service http://controller_VM_EXT_IP:30000 and click on the tab **Alerts** to find the alerts in *Inactive*, *Pending* and *Firing* state.



/etc/prometheus/prometheus.rules > demo alert inactive

> High CPU Usage (0 active)

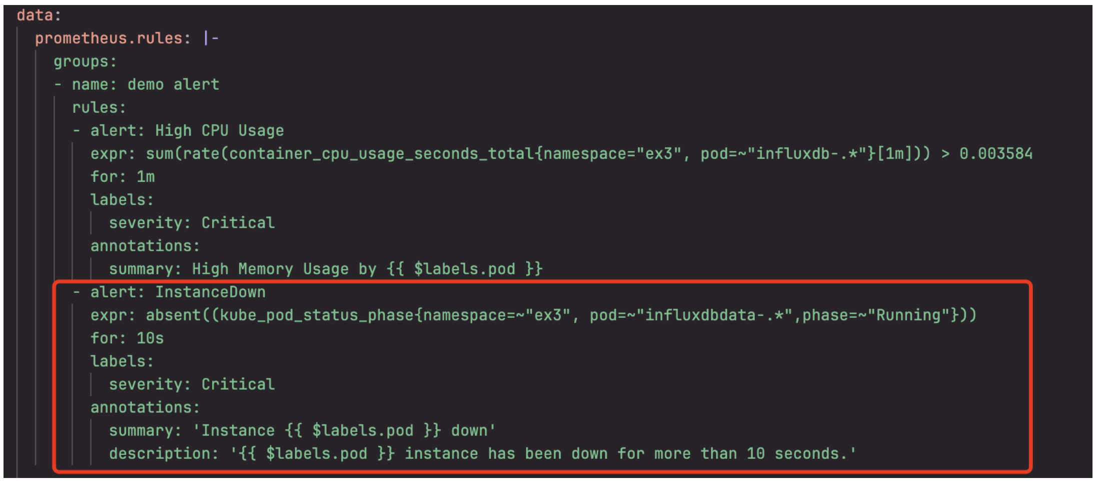
2. Don't worry if your alert is not in a firing state, because your expression does not hold true (as it depends on the threshold value you kept `0.003584`)
3. Finally, the alert signal should show in the Firing state. This means, Prometheus triggered alertmanager to push the notifications.
4. At the same time, you should be able to get the notification in the Zulip topic.

AGS-2: At this point, Nagios may check the Zulip bot messages related to CPU Usage. It's okay if there is no instance of high CPU usage for a continuous 1-minute period. You can move to the next step even without any alert.

5. Adding more alerts

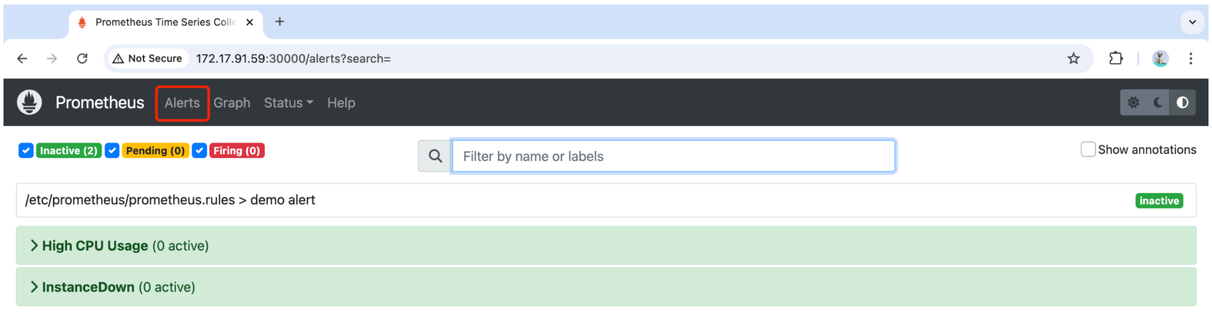
In this task you have to create one more alert to notify when the influxDBdata application is in Running Status.

1. Modify the `promethehus/config-map.yaml` under rules, and add alert `- alert: Instance Down`



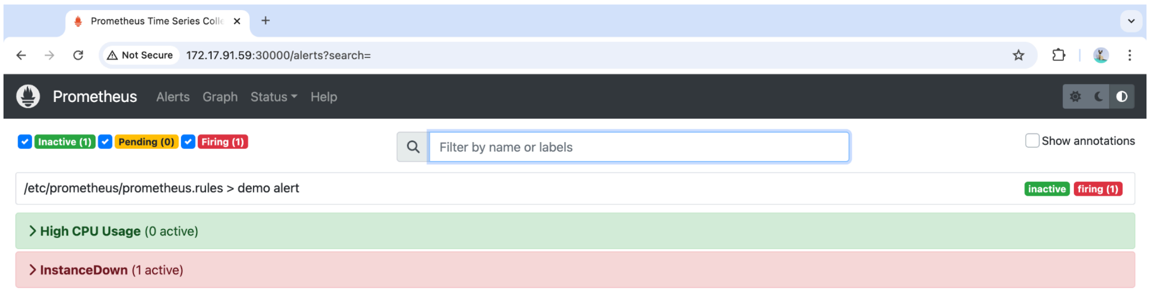
```
data:
  prometheus.rules: |-
    groups:
    - name: demo alert
      rules:
      - alert: High CPU Usage
        expr: sum(rate(container_cpu_usage_seconds_total{namespace="ex3", pod=~"influxdb-.*"}[1m])) > 0.003584
        for: 1m
        labels:
          severity: Critical
        annotations:
          summary: High Memory Usage by {{ $labels.pod }}
      - alert: InstanceDown
        expr: absent((kube_pod_status_phase{namespace=~"ex3", pod=~"influxdbdata-.*",phase=~"Running"}))
        for: 10s
        labels:
          severity: Critical
        annotations:
          summary: 'Instance {{ $labels.pod }} down'
          description: '{{ $labels.pod }} instance has been down for more than 10 seconds.'
```

2. Commit the code with the message “Added 5. Task with an alert configuration ”
3. Now look into the pipeline for updates in prometheus and alertmanager services.
4. Open the prometheus on the web browser and check for the alerts. You should see two alerts.



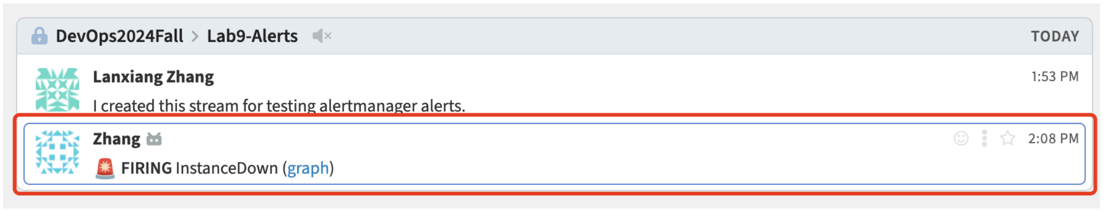
Prometheus Time Series Coll
Not Secure 172.17.91.59:30000/alerts?search=
Prometheus Alerts Graph Status Help
✓ Inactive (2) ✓ Pending (0) ✓ Firing (0) Filter by name or labels Show annotations
/etc/prometheus/prometheus.rules > demo alert inactive
> High CPU Usage (0 active)
> InstanceDown (0 active)

5. **Intentionally delete the deployment of the influxdbdata application deployed on the *controller* VM to check if the alert configuration working correctly.**
   a. k3s kubectl delete -f ~/prac06/prac06-ansible/influxdbdata.yaml (Check for the correct path to your influxdbdata.yaml file)
6. Now should see the alert firing event



Prometheus Time Series Coll
Not Secure 172.17.91.59:30000/alerts?search=
Prometheus Alerts Graph Status Help
✓ Inactive (1) ✓ Pending (0) ✓ Firing (1) Filter by name or labels Show annotations
/etc/prometheus/prometheus.rules > demo alert inactive firing (1)
> High CPU Usage (0 active)
> InstanceDown (1 active)

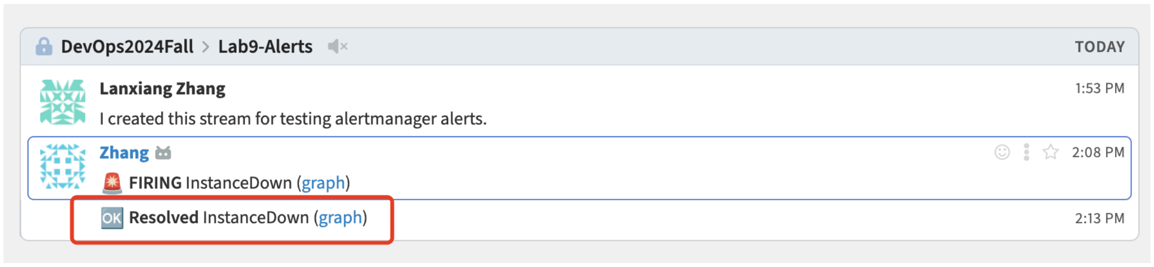
7. You also see the Zulip notification under the topic Lab9-Alerts



DevOps2024Fall > Lab9-Alerts TODAY
Lanxiang Zhang 1:53 PM
I created this stream for testing alertmanager alerts.
Zhang 2:08 PM
🚨 FIRING InstanceDown (graph)

8. Create the inflxudbdata deployment again using the command *k3s kubectl apply -f ~/prac06/Prac06-Ansible/influxdbdata.yaml*

9. You can see the resolved alert after creating the deployment in Zulip.



DevOps2024Fall > Lab9-Alerts TODAY

Lanxiang Zhang 1:53 PM
I created this stream for testing alertmanager alerts.

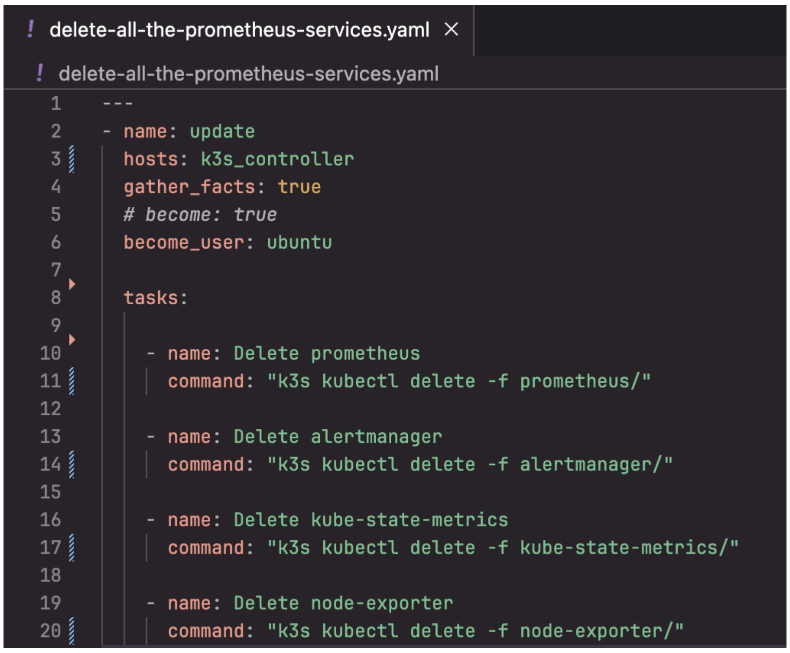
Zhang 2:08 PM
🚨 FIRING InstanceDown (graph)

🆗 Resolved InstanceDown (graph) 2:13 PM

AGS-3: At this point, Nagios will check the Zulip bot messages related to InstanceDown.

If you see AGS-1 and AGS-3 are okay in Nagios scoring, you may simply delete all the resources in the monitoring namespace.

Below is the code sample to delete all the Prometheus services. You can do it using Ansible and GitLab or else you can do it using *k3s kubeclt delete* commands. The ansible code sample is shown below.



delete-all-the-prometheus-services.yaml

```yaml
---
- name: update
  hosts: k3s_controller
  gather_facts: true
  # become: true
  become_user: ubuntu

  tasks:

    - name: Delete prometheus
      command: "k3s kubectl delete -f prometheus/"

    - name: Delete alertmanager
      command: "k3s kubectl delete -f alertmanager/"

    - name: Delete kube-state-metrics
      command: "k3s kubectl delete -f kube-state-metrics/"

    - name: Delete node-exporter
      command: "k3s kubectl delete -f node-exporter/"
```

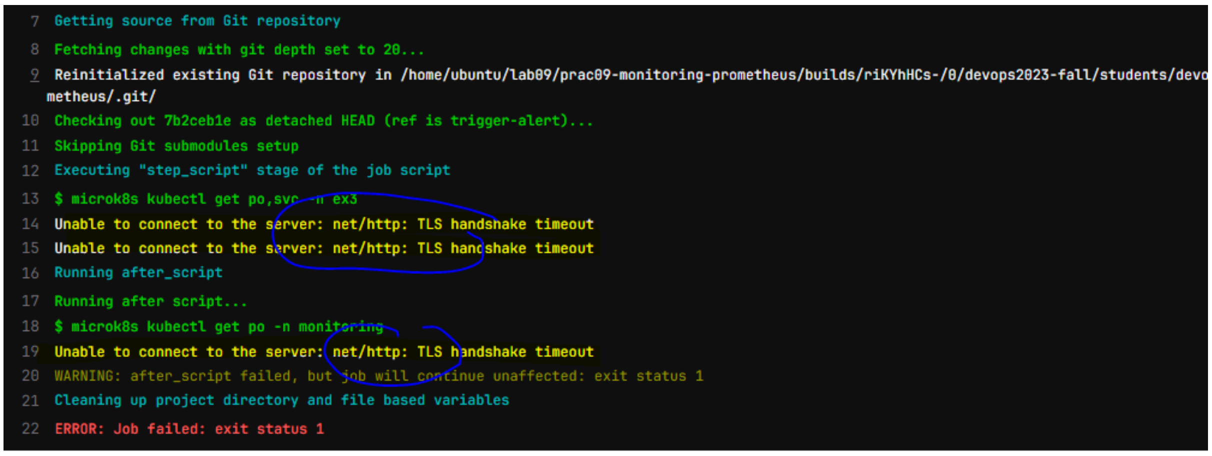
Deliverable

- Go to https://scoring.devops.cs.ut.ee/nagios/
- Find your host by your pseudonym
- Make sure that All the services are in OK state.
- UPDATE: Even if you see all OK (green color), still please keep your application running, if any. You may ignore the CRITICAL (red color) checks if the corresponding lab is graded for you.

Deadline : 13th Nov 2024, 2PM EET

Possible solutions to common problems:

1. If your receive error related to TLS Handshake or Unable to connect or k8s Timeout errors



```
Getting source from Git repository
Fetching changes with git depth set to 20...
Reinitialized existing Git repository in /home/ubuntu/lab09/prac09-monitoring-prometheus/builds/riKYhHCs-/0/devops2023-fall/students/devo
metheus/.git/
Checking out 7b2ceb1e as detached HEAD (ref is trigger-alert)...
Skipping Git submodules setup
Executing "step_script" stage of the job script
$ microk8s kubectl get po,svc -n ex3
Unable to connect to the server: net/http: TLS handshake timeout
Unable to connect to the server: net/http: TLS handshake timeout
Running after_script
Running after script...
$ microk8s kubectl get po -n monitoring
Unable to connect to the server: net/http: TLS handshake timeout
WARNING: after_script failed, but job will continue unaffected: exit status 1
Cleaning up project directory and file based variables
ERROR: Job failed: exit status 1
```

This error is due to memory shortage, so please install the following tool that enables the swap space when memory is exhausted.

```
sudo apt-get update

sudo apt-get install swapspace
```