Attention! Due to Courses website experiencing technical difficulties, files and courses uploaded before the 2020 spring semester are not accessible. We are working on fixing the problem.

Cloud Computing 2022/23 spring

Practice 14 - Service Discovery, Monitoring, and Alerting in cloud infrastructure

In this lab, we will take a look at service discovery, monitoring, and alerting services used in the cloud infrastructure.

Service discovery help to discover, and track the nodes, services, and applications. Monitoring helps to keep track of important events and metrics so that nodes, services, and application downtime or other anomalies are noticed immediately. Further, such abnormal behaviors are notified by generating alerts and sending notifications using email or webhooks, or HTTP events. In this practice session, we will study configuring open-source tools for service discovery, monitoring, and alerting. The following Figure shows the overview of the services to be configured on three OpenStack VMs.

1. **Consul**: It is used for automated service discovery, i.e. track machines to be load balanced and scaled. Consul agents installed on servers that make to join the Consul group.
2. **Prometheus**: Is used for collecting metrics and configuring alerts.
    1. Prometheus **AlertManager** used for sending alerts. It sends alerts to external webhooks -arbitrary REST API endpoints.
    2. Prometheus **Node-exporter** installed on machines to expose the metrics. Prometheus is configured to periodically pull metrics from every server.
3. **Grafana**: Used for visualizing performance metrics.

References

1. Consul documentation: https://developer.hashicorp.com/consul/tutorials
2. Prometheus: https://prometheus.io/docs/introduction/overview/
3. Alert Manager: https://prometheus.io/docs/alerting/latest/alertmanager/#alertmanager
4. Grafana : https://grafana.com/oss/

Exercise 14.1 Setting up of Consul, Prometheus, and Grafana services in VM1

In this task, we are going to set up consul, Prometheus, and Grafana in the VM1 which are used for service discovery, metrics collection, and monitoring the nodes and services.

- Create a Virtual Machine with name `Lastname_VM1`
    - **Image**: `ubuntu22.04`
    - **Flavour**: should be `m3.tiny`
    - **Security Group**:should be `lab14`
- SSH into the VM1

- Installing **Consul as a Server**
    - Install the consul `sudo apt-get update && sudo apt-get install consul`

- Check for the Consul installation `consul -v`. We have installed Consul, and this can be configured to designate this node as a server or agent by configuring the settings. In the next step, we configure this machine as **server**.
- Create a key with Consul CLI `consul keygen > sudo /etc/consul.d/consul.keygen`
- Edit/Create the Consul's configuration file `sudo nano /etc/consul.d/consul.hcl` (NB! Change the bind_addr to your VM1_IP_ADDRESS)

```
datacenter = "lab14"
bootstrap_expect = 1
server = true
data_dir = "/opt/consul"
log_level = "INFO"
client_addr = "0.0.0.0"
bind_addr = "VM1_IP_ADRESS"
node_name = "consul"
leave_on_terminate = true
rejoin_after_leave = true
connect{
        enabled = true
}
ui = true
encrypt = "3EY9MGe2a1ElUNrsl5E6rHAtJEL7lqe+1CDZAyY8yTU="
```

  - We named our datacenter group as `datacenter = "lab14"` and `server=true`
- Start the Consul service `sudo service consul start`
- Access to consul's web UI http://VM1_IP_ADDRESS:8500 and see the up/down services, currently it's only the consul server itself

- Installing **Prometheus**
  - Download the Prometheus repo `wget https://github.com/prometheus/prometheus/releases/download/v2.37.0/prometheus-2.37.0.linux-amd64.tar.gz`
  - Untar `tar -xvzf prometheus-2.37.0.linux-amd64.tar.gz`
  - Copy to bin `sudo cp prometheus-2.37.0.linux-amd64/prometheus /usr/bin/prometheus`
  - Create user `sudo useradd --system prometheus`
  - Make directory `sudo mkdir /etc/prometheus /var/lib/prometheus`
  - Copy config file `sudo cp prometheus-2.37.0.linux-amd64/prometheus.yml /etc/prometheus/prometheus.yml`
  - Copy the files `sudo cp -r prometheus-2.37.0.linux-amd64/consoles /etc/prometheus/consoles`
  - Copy the libs `sudo cp -r prometheus-2.37.0.linux-amd64/console_libraries /etc/prometheus/console_libraries`
  - Change ownership `sudo chown -R prometheus:prometheus /etc/prometheus`
  - Change ownership `sudo chown -R prometheus:prometheus /var/lib/prometheus`
  - Let us create Prometheus as service
    - Create a service file for Prometheus `sudo nano /etc/systemd/system/prometheus.service` **** Take the content of the file from here: prometheus.service to create the service.
    - Enable the service `sudo systemctl enable prometheus.service`
    - Start the service `sudo service prometheus start`
  - Try accessing the in the browser http://VM1_IP_ADRESS:9090

- Installing **Grafana**
  - Install required libraries: `sudo apt-get install -y libfontconfig1`
  - Get the Grafana .deb pkg `wget https://dl.grafana.com/enterprise/release/grafana-enterprise_9.0.4_amd64.deb`
  - Install the pkg `sudo dpkg -i grafana-enterprise_9.0.4_amd64.deb`
  - Restart the service `sudo service grafana-server start`
  - Try accessing the Grafana in the browser http://VM1_IP_ADRESS:3000

Exercise 14.2 Service Discovery: Configuring the consul agent, node-exporter (metric scrapper) on *System Under Test* machines

In this task, we will configure the consul agent as a client and node-exporter to scrape the metrics from the *System Under Test* machines. Here, *Consul client* connects to *Consul Server*, and helps for service discovery. In this task, node-exporter services are listed in the Consul's service list that belongs to the data center group *lab14*.

- Create VM2 with **name**: Lastname_VM2, **flavour**: `m3.nano`
  - SSH to the VM2
- Installing **Node Exporter** - Used to scrape the resource utilization metrics of the VMs
  - Download the node-exporter pkg `wget https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporter-1.3.1.linux-amd64.tar.gz`
  - Extract the pkg `tar -xvzf node_exporter-1.3.1.linux-amd64.tar.gz`
  - Copy to the bin `sudo cp node_exporter-1.3.1.linux-amd64/node_exporter /usr/bin/node_exporter`
  - Make it run as a service `sudo vi /etc/systemd/system/node_exporter.service`

```
[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target

[Service]
Type=simple
ExecStart=/usr/bin/node_exporter --collector.systemd --collector.processes

[Install]
WantedBy=multi-user.target
```

- Enable the service `sudo systemctl enable node_exporter.service`
- STart the service `sudo service node_exporter start`

- Installing **Consul client**
  - Install Consul `sudo apt-get update && sudo apt-get install consul`
  - To configure the Consul client and services to report to the server, we have to create two files consul.json (This is to tell that consul will act as a client) and services.json (This is to add node-exporter service in the consul services)
    - Create consul.json at /etc/consul.d/ `sudo nano /etc/consul.d/consul.json` (PS! Change VM1_IP_ADRESS in retry_join to your VM1 IP address)

```
{
 "datacenter": "lab14",
 "data_dir": "/var/consul",
 "encrypt": "3EY9MGe2a1ElUNrsl5E6rHAtJEL7lqe+1CDZAyY8yTU=",
 "log_level": "INFO",
 "enable_syslog": true,
 "enable_debug": true,
 "enable_script_checks": true,
 "server": false,
 "leave_on_terminate": true,
 "rejoin_after_leave": true,
 "retry_join": [
    "VM1_IP_ADRESS"
    ]
}
```

    - Create services.json at /etc/consul.d/ - `sudo nano /etc/consul.d/services.json`

```
{
  "services": [
    {
      "name": "node_exporter",
      "tags": [
        "node_exporter"
      ],
      "port": 9100,
      "checks": [
        {
          "tcp": "localhost:9100",
          "interval": "10s"
        }
      ]
    }
  ]
}
```

  - Start the agent `sudo service consul start`
  - Go to consul server UI http://VM1_IP_ADDRESS:8500. We will find our node-exporter service has been registered.

- Configure Prometheus to pull the metrics from the Consul service data center group `lab14`
  - NB! SSH to VM1
  - Edit prometheus.yml file `sudo nano /etc/prometheus/prometheus.yml`
    - Append the following inside the `scrape_configs:` block

```
  # node exporter configuration,
  - job_name: "node_exporter"
    consul_sd_configs:
      - server: "VM1_IP_ADDRESS:8500"
        datacenter: "lab14"
        services: [node_exporter]
    relabel_configs:
      - source_labels: [__address__]
        target_label: instance
```

    - Change the server IP address in the configuration (**VM1_IP_ADDRESS**) and you can see that `datacenter: "lab14"`. This indicates the data center with services group name node_exporter is used for pulling the metrics.
  - Restart the Prometheus service.
  - Now, go to the Prometheus UI http://VM1_IP_ADDRESS:9090 and check the targets, should see the node-exporter service.

- Configure Grafana data source to query the metrics from Prometheus and create a dashboard to visualize the data. This is similar to task you performed in 9.4.6 in Practice 9

- Make sure you are in VM1
- Go to the Grafana and add the Prometheus data source
  - Change URL as `http://127.0.0.1:9090`
  - Click on Test and Save
- Add the dashboard to visualize the resource utilization metrics of *System Under Machines*
  - Go to Grafana dashboards http://VM1_IP_ADDRESS:3000/dashboard/import and In import from grafana.com section, add `1860` in id and select Prometheus as data source.
- Go to the Dashboard and should see the graphs as shown below. Here, you're visualizing the data of VM2.

- Now, let us create VM3 and monitor the resource metrics. In this task, you need to configure the following and should see the service registered in the consul and directly visualize the data in the grafana. Further, you need not edit/modify the Prometheus.yml
  - Install and configure the following
    - Install Consul
    - Configure Consul client
    - Install node-exporter
  - Go to Consul UI, should see the service registered.
  - **Make a screenshot of the consul dashboard showing two node-exporter services**
  - Go to Grafana and select the Host and Visualize the resource metrics.
  - **Make a screenshot of the grafana dashboard showing VM3 metrics (you can select the Host) with a graph showing CPU, System Load etc**

Exercise 14.3 Monitoring with Prometheus

In this task, we will write the queries to get the data from the Prometheus time series database. We will use Prometheus Query Language (PromoQL) to scrape the data. Further, we will deploy the flask application and generate load on the *System Under Test* machines using locust.

- Understanding Prometheus Queries
  - Go to Prometheus service http://VM1_IP_ADDRESS:9090 and click on the graph. Now, you will get the Query Editor to write the PromoQL queries. You can read the documentation on PromoQL
  - Write a query to check the target services in the Prometheus is active or down.
    - Query is `up == 0` and click on Execute.
    - The targets that are in the UP state (running and accessible to Prometheus) will have the value 1, and targets that are not in the UP (or DOWN) state (not running or inaccessible to Prometheus) will have the value 0.
  - Write a query to compute the current average CPU Utilization of VM2 and VM3. `(1 - avg(irate(node_cpu_seconds_total{mode="idle"}[10m])) by (instance)) * 100`
    - *node_cpu_seconds_total*: CPU utilization of a node in MilliCPU measured in seconds.
    - *mode* indicates the CPU time spent in idle, iowait
    - *irate*: irate(v range-vector) calculates the per-second instant rate of increase of the time series in the range vector, for example, 10m in the query.
    - *avg*: average over idle CPU usage for 10m.
    - Here, it calculates CPU usage in percent by subtracting the idle CPU with 1.
  - Likewise, memory utilization is `100 * (1 - ((avg_over_time(node_memory_MemFree_bytes[10m]) + avg_over_time(node_memory_Cached_bytes[10m]) + avg_over_time(node_memory_Buffers_bytes[10m])) / avg_over_time(node_memory_MemTotal_bytes[10m])))`
  - Write a query to calculate the average 5 minutes load on the node `avg(node_load5{job="node_exporter"}) / count(count(node_cpu_seconds_total{job="node_exporter"}) by (cpu)) * 100`

- Now, let us make the systems busy or to consume more resources using the Flask app and locust load testing tool.
  - Clone the flask app on VM2 `git clone https://shivupoojar87@bitbucket.org/lab2app/lab14.git`
  - Go to the folder flaskapp, run the application in the background `nohup flask run --host=0.0.0.0 &` (PS!! create venv, install the required packages in requirements.txt. This is similar to Practice Session 1, Task 1.4)
    - This flask application computes the value of Pi up to 5000 decimal points when a user request is invoked.
  - Now, Install locust `pip install locust`.
  - Run the load test `locust --headless --host http://127.0.0.1:5000 -f locustfile.py` (PS!! locustfile.py is located outside the flaskapp folder)
    - This load test is carried out for 20 seconds.
  - Meanwhile, run the Prometheus queries for CPU utilization and System Load of 5m average, in Prometheus Query Editor.
  - **Make a screenshot of the Prometheus query execution with a graph showing results and IP should be visible**

Exercise 14.4 Alerting: Configuring alert manager to generate alerts and notify to Zulip channel

In this task, we will study alerts and notifications using Prometheus Alertmanager. Prometheus generates the alerts based on the query and the alertmanager triggers the event to send notifications via webhooks, or other endpoints. Here, we will notify via Zulip Streams.

- Installing AlertManager service on VM1
  - Download the alertmanager `wget https://github.com/prometheus/alertmanager/releases/download/v0.22.2/alertmanager-0.22.2.linux-amd64.tar.gz`
  - Extract the tar file `tar xzf alertmanager-0.22.2.linux-amd64.tar.gz`
  - Move to required location `sudo mv -v alertmanager-0.22.2.linux-amd64 /opt/alertmanager`
  - Change the ownership `sudo chown -Rfv root:root /opt/alertmanager`
  - Create a data directory `sudo mkdir -v /opt/alertmanager/data`
  - Change the ownership of data directory `sudo chown -Rfv prometheus:prometheus /opt/alertmanager/data`
  - Create as a service `sudo nano /etc/systemd/system/alertmanager.service`

        - Add the script

```
[Unit]
Description=Alertmanager for Prometheus

[Service]
Restart=always
User=prometheus
ExecStart=/opt/alertmanager/alertmanager --config.file=/opt/alertmanager/alertmanager.yml --storage.path=/opt/alertmanager/data
--log.level=debug
ExecReload=/bin/kill -HUP $MAINPID
TimeoutStopSec=20s
SendSIGKILL=no

[Install]
WantedBy=multi-user.target
```

    - Reload the daemon `sudo systemctl daemon-reload`
    - Start the service `sudo systemctl start alertmanager.service`
- Now, let us update the alertmanager service in Prometheus.yml
    - Edit the Prometheus.yml to add the alertmanager `/etc/prometheus/prometheus.yml`
        - Append in scrape_configs with following

```
  - job_name: 'alertmanager'
    static_configs:
    - targets: ['VM1_IP_ADDRESS:9093']
```

    - Update the *targets* element under *alerting*, `targets: ['VM1_IP_ADDRESS:9093']`
        - It looks like

```
alerting:
  alertmanagers:
    - static_configs:
        - targets: ['VM1_IP_ADDRESS:9093']
```

    - Restart the Prometheus service
- Writing Alert rules
    - We will write an alert to notify the status of the targets (services) in the Prometheus
        - We will use the *up* expression to find the state of the targets.
        - Write the rules `sudo nano /etc/prometheus/rules.yml`. (PS!! Replace Lastname with your lastname)

```
groups:
 - name: test
   rules:
   - alert: InstanceDown
     expr: up == 0
     for: 1m
     labels:
       name: Lastname_Node_Exporter
     annotations:
       summary: Node Exporter running in {{$labels.instance}} is down.
```

            - The expression *up == 0* is executed for every 1m, and alerts are generated if succeeded to true.
    - Add the rules.yml file in the *rule_files* section of the prometheus.yml configuration file
        - `sudo nano /etc/prometheus/prometheus.yml`

```
# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
   - "rules.yml"
```

    - Restart the Prometheus service
    - Now, navigate to the URL http://VM1_IP_ADDRESS:9090/rules from your browser, and you should see the rule InstanceDown that you've just added.

- To validate, for a while stop the node-exporter service of VM2.
- Navigate to the URL http://VM1_IP_ADDRESS:9090/alerts , and you should see the state of the alert InstanceDown.

- You can check the logs of the alertmanager using the command `journalctl -u alertmanager --since "1 hour ago"` for debugging.

- Sending alerts to Zulip Stream
    - Join the Zulip Stream **Cloud Comuting Lab 14-Monitoring**.
    - Create a Zulip Bot as mentioned in the document.
        - Bot Type: Incoming Webhook
        - Full Name: "YOUR_Last_name cloud computing bot" (NB! Replace last name with yours)
        - Cookie: Lastname
        - Click on create
        - After creating, notedown the API KEY which is required to construct the webook.
    - Create the Webhook URL as mentioned in the document
        - URL should look like. (PS!! Replace API_KEY with your API KEY) `https://zulip.cs.ut.ee/api/v1/external/alertmanager?api_key=APIK_KEY&stream=Cloud%20Comuting%20Lab%2014-Monitoring&name=name&desc=summary`
    - Configure webhook in the alertmanger `sudo nano /opt/alertmanager/alertmanager.yml`

```
route:
  group_by: ['alertname']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 1h
  receiver: 'ops-zulip'
receivers:
- name: 'web.hook'
  webhook_configs:
  - url: 'http://127.0.0.1:5001/'
- name: 'ops-zulip'
  webhook_configs:
    - url: "https://zulip.cs.ut.ee/api/v1/external/alertmanager?api_key=API_KEY&stream=Cloud%20Comuting%20Lab%2014-
Monitoring&name=name&desc=summary"
inhibit_rules:
  - source_match:
      severity: 'critical'
    target_match:
      severity: 'warning'
    equal: ['alertname', 'dev', 'instance']
```

- Alerts to Zulip are fired every 5m as described in `group_interval: 5m`
- Restart the alertmanager service
- You can check the status of the alerts in the Prometheus http://VM1_IP_ADDRESS:9090/alerts
  - PS!! You can also check the logs of alertmanager for debugging.
- Finally, you should see the notification Zulip

Exercise 14.5 Some more alert rules.

In this task, you will write some more alert rules for CPU utilization. This document provides a good overview of various alert rules for host or nodes

- Create a rule to monitor the CPU utilization of nodes that reaches above 50%. To create a new alert rule in `/etc/prometheus/rules.yml`, you need to add it under `rules`.
  - Create a new rule with `CPU OverUtilzation` as similar to `InstanceDown`
  - For *expr*: Use the same CPU utilization PromoQL query from Exercise 3. But add the `>50`.
  - Change the *summary* as *CPU utilization of the node {{$labels.instance}} had reached the threshold* .
- Restart the Prometheus.
- Generate the load using the locust to flask app.
- You should see the Zulip notifications. (PS!! You can also check the logs of alertmanager for debugging)

Deliverables:

1. Screenshots from Exercise 14.2-2, 14.3-1
2. Prometheus configuration file (/etc/prometheus/prometheus.yml) and rules file(/etc/prometheus/rules.yml)
3. Delete all the VMs

Task	Lab 14 - Monitoring
Current submission	ZIP (15:54 09.05.2023)
	If your homework consists of multiple files (for example HTML + CSS + JS) it should be archived before submitting.
Report	Choose File No file chosen
File	Choose File No file chosen
Comment	

Submit

Possible solutions to potential issues

- If your facing the problem while sending notification to zulip stream by alertmanager, the logs of alertmanager can help for debugging `journalctl -u alertmanager --since "1 hour ago"`

In case of technical problems or questions write to:ati.error@ut.ee
Contact the course organizers with the organizational and course content questions.