LS Lab 5 - Monitoring

> Group of two, one report for group (please mention in the report both students' names and contribution). The subtopic depends on your student number (stX):
>
> stX mod 3 = 0 -> subtopic 3
> stX mod 3 = 1 -> subtopic 2
> stX mod 3 = 2 -> subtopic 1

In this lab, you will get familiar with various types of infrastructure monitoring, and then learn how to deploy, configure and operate them.

Subtopic 1 - Infrastructure Monitoring

Task 1 - Preparation

Take a pick from the list proposed below:

- Zabbix (preferred option)
- Nagios XI + Performance Graphs
- Icinga
- M/Monit
- your suggestion (if you know any other open source tool that can handle next tasks)

After you select the monitoring instrument

- Install a new guest system
- Install and configure web-server + DBMS (e.g. NGINX + MariaDB)
- Install and configure a CMS. Create a page with some heavy content
- Install another guest system and deploy the monitoring facility (and agents, if required)

Task 2 - Status alerts

Setup a few alerts including

- against the guest system itself (ping)
- against available RAM or disk space (with threshold about 90%)
- against the web service
- and against two web pages, a simple one and the heavy-duty one

Validate that your monitoring displays an alert once you destroy the service (what is the delay, how long does it take for it to appear?)

Bonus: configure and validate that you receive an email (you can use Telegram instead of email (at least it is possible in Zabbix)) for alerts.

Task 3 - Stress & performance graphs

Take a pick for stress benchmark

- yandex.tank
- AB
- Autocannon
- Siege

- K8
- ...

Then use your load-testing tool of choice and perform a few different load tests

- Define performance graphs for AT LEAST the four different kinds of resources (CPU, RAM, DISK I/O, Network TX/RX)
- Play with the number of threads, number of clients, and request bodies while performing requests
- Look at the monitoring screen to see how bad your system is. What *resource* is the more impacted?..
- Configure an alert threshold for one of those and validate it *(Note: it does not make sense to define a threshold for CPU or Network unless you can define a timer within it)*

Bonus: deal with the resource metrics from the virtualization host instead of guest agents.

Bonus: enable SSL on your web service and evaluate the impact on delay and performance.

Questions:

- Do the resulting metrics match with your stress load-test?
- Are your metrics representative of the actual state of affairs? Do they reflect reality?
- Are some of those irrelevant and should be omitted?
- So is your system high-load capable? If not, what would it take to make it happen?

Task 4 - Trends & business metrics

Trends - collect and store metric results for some period of time to see the analytic and forecasts. Define a few ones, for example

- RPC (requests per second) on web-server, DBMS, etc
- worker threads and processes
- queue size
- reply time (web-page generation time).
- MySQL/MariaDB types of operations, operations per second, db bandwidth, db disk i/o

and eventually define an alert threshold for one of those

Business metrics - the above metric does not mean or cannot prove that you application works well in terms of functionality. Proceed with more fine-grained data, for example

- user activity (number of visits/views)
- how many views vs how many applicants
- page load time
- login/sign ups/buys/comments/etc
- ad conversions
- high-level metrics

Subtopic 2 - Time-Series based Monitoring: Prometheus + TSDB + Grafana

Task 1 - Prepare monitoring environment

Choose and deploy your monitoring stack. Default choice is:

- *Prometheus* as core monitoring system for metrics processing
- *InfluxDB* as TSDB for time series metrics storing
- *Grafana* as software for metrics visualization and reports generation

Task 2 - Configure monitoring agents

You have to deploy and configure metrics exporters on your monitoring agents (targets instances). Targets might be your VMs and applications/services that you have used in previous labs on this course. Try to set up different types of exporters and as much as possible, but usually core exporters are:

- exporter for instance metrics (*nodexporter*)
- exporter for containers metrics (*cadvisor* is one of the most popular solution)
- exporter for containers states (*dockerstatesexporter*)
- *bonus: integrated exporter of your application/service metrics (here you need to write a logic to expose your service metrics on particular port and enable* `Service discovery` *Prometheus feature)*
- ...

Hint: you might use docker-compose.

Task 3 - Processing with Prometheus

1. Since your monitoring infrastructure and agents metrics exporters are configured, log in Prometheus console, and make sure that you are getting metrics:
    - go to `Targets` and confirm that you can see your target instances
    - go to `Graph` and confirm that you can see coming metrics via PromQL
2. Create and write your alerts rules in `prometheus.rules.yml` file. Try to play with different alerts types and test and as much as possible. Some common alerts are:
    - high instance memory usage
    - high instance cpu usage
    - high instance disk usage
    - docker container restarting
    - docker container exited
    - docker container dead
    - exporter down
    - kubernetes pod crashed
    - *specific alerts that related to your app/service: high heap usage, high cache hit rate usage...*
    - ...
3. Apply your alerts rules and provoke some alerts conditions. Confirm that you can see alerts on Prometheus dashboard.

    Bonus: configure `Alertmanager` *for alerts visualization. Group alerts by instances, types, environments... Provoke false-positive alerts and silence them.*

Task 4 - Processing with Grafana

After you set up metrics catching and alerts rules, it it time to visualize your metrics and deploy elegant charts.

1. Log in Grafana console and create several *dashboards* that are separated by metrics types and logic.
2. Deploy some *panels* in dashboards. Write expressions (*metrics query*) for them and name them according to panels purposes.
3. Make sure that your dashboards are ready and work properly.

   Bonus: use dynamic variables within Dashboards.

Bonus 1: wrap your deployment in tasks 1-2 as ansible roles.

Bonus 2: organize a process of cross-systems notifications about your alerts.

Subtopic 3 - Logging system: ELK cluster

Task 1 - Prepare logging environment

1. Choose and deploy your monitoring stack. Default choice is:

- *logstash* is a server-side data processing engine that receives data from multiple sources, parses the log, and then sends it to the Elasticsearch database
- *elasticsearch* is the core of the entire system, which combines the functions of a database, search engine and analytical system
- *kibana* allows users to visualize data using charts and graphs in Elasticsearch. You can also administer the database through Kibana
- *filebeat agents* is logging agent that nstalled on the machine generating the log files

2. Try to deploy different roles within your cluster: *coordinator, master node, data node…*
3. Configure basic indices politics: *shards allocation politics, numbers of primary & replica shards, jvm heap usage configuration…*

> Keep in mind that usually ELK cluster requires quite high system resources.

Task 2 - Configure logging targets

Prepare your target logging instances to collect logs from them. Targets might be your VMs and applications/services that you have used in previous labs on this course.

Task 3 - Processing with logging

1. Log in Kibana console and discover coming logs.
2. Play with different filters to learn how to quickly get necessary logs.
3. Try and test popular ELK API commands. Examples:
   - get the current heap.percent for each node
   - calculate the current JVM memory pressure for each node
   - check your cluster health status
   - view unassigned shards
   - get cluster settings

- get index settings
- check the current disk space of your nodes
- get indices
- get nodes
- get shards
- ...

4. Generate reports and make conclusions based on the data generated in the reports.

Bonus 1: wrap your deployment in tasks 1-2 as ansible roles.

Bonus 2: configure and test extended index management politics: clean old logs, backup logs, time period to create & update indexes...

Bonus 3: use your ELK cluster as a base to turn this system into IDS/IPS