Project 8 - Monitoring the Metrics Server

Goal:

The goal of this project is to add the grafana VM to our monitoring and alerting system, paying special attention to the services that the host provides and monitoring them accordingly.

Instructions:

First, start a command line session. Change into your `linuxclass` folder and then change into the `icinga` directory. From there, bring up the VM and connect to it.

```
cd linuxclass
cd icinga
vagrant up
vagrant ssh
```

When adding a new client to icinga, we need to create a ticket for it. Let's do that now for the grafana host.

```
sudo icinga2 pki ticket --cn 'grafana'
```

You'll need this ticket number when you configure Icinga on the grafana server.

Install the Icinga Client

Open up a new command session and connect to the grafana host.

```
cd linuxclass
cd grafana
vagrant up
vagrant ssh
```

Start off by enabling the Icinga repository.

```
sudo dnf install -y http://mirror.linuxtrainingacademy.com/icinga/icinga-rpm-release.noarch.rpm
```

Next, enabled the EPEL repository.

```
sudo dnf install -y epel-release
```

Now enable the PowerTools repository.

```
sudo dnf config-manager --set-enabled powertools
```

Finally, install Icinga and the Nagios monitors.

```
sudo dnf install -y icinga2 nagios-plugins-all
```

Run the node wizard to tell the client about the master Icinga server. Use the information in bold below to supply the node wizard with the appropriate information.

```
sudo icinga2 node wizard
```

```
Please specify if this is an agent/satellite setup ('n' installs a master
setup) [Y/n]: (press ENTER)
Please specify the common name (CN) [grafana]: (press ENTER)
Please specify the master endpoint(s) this node should connect to:
Master/Satellite Common Name (CN from your master/satellite node): icinga
Do you want to establish a connection to the parent node from this node?
[Y/n]: (press ENTER)
Please specify the master/satellite connection information:
Master/Satellite endpoint host (IP address or FQDN): 10.23.45.30
Master/Satellite endpoint port [5665]: (press ENTER)
Add more master/satellite endpoints? [y/N]: (press ENTER)
…
Is this information correct? [y/N]: y
Please specify the request ticket generated on your Icinga 2 master
(optional).
 (Hint: # icinga2 pki ticket --cn 'grafana'): (Use ticket from above.)
Please specify the API bind host/port (optional):
Bind Host []: (press ENTER)
Bind Port []: (press ENTER)
Accept config from parent node? [y/N]: y
Accept commands from parent node? [y/N]: y
Local zone name [grafana]: (press ENTER)
Parent zone name [master]: (press ENTER)
Do you want to specify additional global zones? [y/N]: (press ENTER)
Do you want to disable the inclusion of the conf.d directory [Y/n]:
(press ENTER)
```

Start icinga on the grafana host so it will load this configuration. Remember to enable the service as well.

```
sudo systemctl start icinga2.service
sudo systemctl enable icinga2.service
```

Configure the Monitors for the Client on the Server

Return to your command line session on the icinga server.   Your prompt should like this:

```
[vagrant@icinga ~]$
```

Add the following configuration on the master Icinga server in the `/etc/icinga2/zones.d/master/grafana.conf` file.

```
sudo nano /etc/icinga2/zones.d/master/grafana.conf
```

NOTE: This file will not exist, you will need to create it. Also note the following configuration spans multiple pages in this document.

[This space intentionally left blank. Instructions continue on the following page.]

```
# Zone Configuration

object Endpoint "grafana" {
  host = "10.23.45.40"
}

object Zone "grafana" {
  endpoints = [ "grafana" ]
  parent = "master"
}

# External Monitors

object Host "grafana" {
  import "generic-host"
  address = "10.23.45.40"
  vars.os = "Linux"
  vars.http_vhosts["http-grafana"] = {
    http_uri = "/"
    http_port = 3000
  }
  vars.notification["mail"] = {
    groups = [ "icingaadmins" ]
  }
}

# Internal Monitors

object Service "load" {
  import "generic-service"
  check_command = "load"
  host_name = "grafana"
  command_endpoint = "grafana"
}

object Service "swap" {
  import "generic-service"
  check_command = "swap"
  host_name = "grafana"
  command_endpoint = "grafana"
}

object Service "disk" {
  import "generic-service"
  check_command = "disk"
  host_name = "grafana"
  command_endpoint = "grafana"
}
```

}

```
object Service "proc-sshd" {
  import "generic-service"
  check_command = "procs"
  vars.procs_command = "sshd"
  vars.procs_critical = "1:"
  host_name = "grafana"
  command_endpoint = "grafana"
}

object Service "proc-rsyslog" {
  import "generic-service"
  check_command = "procs"
  vars.procs_command = "rsyslogd"
  vars.procs_critical = "1:1"
  host_name = "grafana"
  command_endpoint = "grafana"
}

# Telegraf

object Service "proc-telegraf" {
  import "generic-service"
  check_command = "procs"
  vars.procs_command = "telegraf"
  vars.procs_critical = "1:1"
  host_name = "grafana"
  command_endpoint = "grafana"
}

# InfluxDB specific monitors

object Service "proc-influxdb" {
  import "generic-service"
  check_command = "procs"
  vars.procs_command = "influxd"
  vars.procs_critical = "1:1"
  host_name = "grafana"
  command_endpoint = "grafana"
}

object Service "port-8086-influxdb" {
  import "generic-service"
  check_command = "tcp"
  vars.tcp_port = "8086"
  host_name = "grafana"
}

# Grafana specific monitors
```

```
object Service "proc-grafana-server" {
  import "generic-service"
  check_command = "procs"
  vars.procs_command = "grafana-server"
  vars.procs_critical = "1:1"
  host_name = "grafana"
  command_endpoint = "grafana"
}
```

You've seen many of the checks before. However, we added a few checks that are specific to this particular server. In the main host stanza we make sure to see if we can access the Grafana web interface. There, we had to specify a port as Grafana runs on port 3000.

Also, we are checking that the "telegraf" process is running. If this process were to stop, then we would fail to collect metrics while it was down. We are also ensuring that the "influxd" process is running. Also, the influxd process listens on port 8086, so we added a check for that as well. This could come in handy in the real world where a well-meaning network engineer might create a new firewall rule that blocks port 8086 traffic. Just because a process is running on a server, doesn't mean that the service is actually functioning or that the service can be reached over the network.

For Grafana, we added a check for its process, grafana-server. We're already testing for it's port in the main host stanza, so there is no need to check for it again.

By the way, you could go back and add a check for the telegraf process on all the hosts that will be sending data to InfluxDB on the grafana VM.

Since we've updated the configuration we need to restart Icinga.

```
sudo systemctl restart icinga2.service
```

Now we can visit the Icinga web front end (http://10.23.45.30/icingaweb2) and check the status of our new host and its services. (Remdiner the username and password are both "admin" for the Icinga web front end.)

If you want, you can test each monitor to make sure it's working properly. Let's switch back to the grafana server. Be sure to bring up the command line session that is connected to grafana. From here, let's stop InfluxDB to test our monitors.

```
sudo systemctl stop influxdb
```

Go back to the Icinga web interface and watch as two of our checks fail. The process check will fail as will the TCP port check.

We can start InfluxDB and watch our monitors succeed.

```
sudo systemctl start influxdb
```

From here, you can test all the services in a similar fashion if you would like. (IE, stop a service, check to see that an alert is triggered, etc.)

Determining What to Monitor

When you are adding new systems to monitoring, it's important to think about what function the server plays. For example, if it is a web server then you know you should monitor the websites that will be hosted on the server. If it's a service like InfluxDB, then it can be more complex.

One key area to check is for running processes. Ideally you would have an idea of what processes are running on your system before you install and configure the applications running on the server. If you don't, you can always compare the process listing with another server.

```
ps -ef | less
```

Look for anything associated with the service you are monitoring. Many times you'll find a process with the name of the service. Sometimes the name will be slightly different. You can also look for processes running as an application user.

Look at any recent users added to the system.

```
tail /etc/passwd
```

See what processes those users are running.

```
ps -fu influxdb
```

In addition to finding processes, you can see if those processes are listening on any ports. We use the netstat command to do this. We'll use the "-n" option to display port numbers and IP addresses instead of service names and host names. We'll limit our results to udp ("-u") and tcp ("-t"). Otherwise, we'll see internal system connections that we're not interested in; specifically IPC (inter-process communication) sockets. The "-l" option will only display listening processes. Finally, the "-p" option will display PID and process name that is doing the listening.

```
sudo netstat -nutlp
```

You should see a process listening on port 8086. Specifically, it will be a process named "influxd." Now we know InfluxDB listens on port 8086. (Actually, we already knew that because we read the documentation and performed the installation... but if for some strange reason we forgot, we now know the process of determining this information.)

You'll also find that InfluxDB listens on port 8088 as well. However, this is for receiving commands from the command line using the influx client. On a rare occasion, a system administrator would use the command-line client. I wouldn't monitor this port as it's rarely used. Plus, when it is used, it is used by someone who can troubleshoot any underlying issues.

This brings up an important point. How did I know what port 8088 did? Answer: I read the documentation. You'll not only find documentation on a service's website, but you'll also find it in configuration files, in README's, etc. Reading documentation can take a lot of the guesswork out of what to monitor.

One final point. As you are installing and configuring a service, get into the habit of making notes -- mental or written -- about things you can monitor. For example, when we were installing telegraf, we configured it to talk to InfluxDB on port 8086. That should have been a big clue that we'll want to monitor for the availability of port 8086 on the Grafana host.