Lab 9.1: Prometheus
IN719 Systems Administration

Introduction

Nagios is a widely used monitoring system. It's been around for a while, so it's pretty well proven and does it's job well enough for a lot of purposes, but it doesn't do everything you might want. Its checks basically tell you if a particular service is broken or not, but it doesn't help you track the performance of that service over time. It also mainly does *black box* monitoring of services, meaning it tells you if the service is responding properly but doesn't monitor the internal workings of the service. That black box nature of Nagios checks is really why getting NRPE checks working is a bit difficult.

In recent years we've seen a new generation of monitoring tools that allow us to track the performance of services with a high degree of detail. These monitoring tools give us increased insight into what is happening on our monitored systems, but at the cost of increased effort to set up. In this lab we will see one of these systems, *Prometheus*. Where Nagios can check on a web service and verify that it's responding correctly, Prometheus can actually monitor the internal workings of the web service software and report on its performance. To get this extra information, however, we generally have to modify the code for that service to supply data to Prometheus.

Do the tasks for this lab on your backup server to minimise the chances that you'll do anything that effects work you've done on the other servers. Also, it's not necessary to manage any of this with Puppet. Our goal for this lab is just to get some exposure to an alternative monitoring style.

Much of the content of this lab comes from the "First Steps" documentation on the Prometheus project web site.

1 Download and run Prometheus

Prometheus is written in Go and distributed as a statically linked binary, so we can basically just download and run it. Download it onto your backup server with the command

```
wget https://github.com/prometheus/prometheus/releases/download/v2.18.0-rc.1/
           prometheus-2.18.0-rc.1.linux-amd64.tar.gz
```

(The command above is broken into two lines for fit on the page, enter it as a single line.)

Unpack the downloaded file and change to the new Prometheus directory. Before we run it, let's look at the configuration file, `prometheus.yml`.

```
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

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

```
# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ['localhost:9090']
```

Prometheus works by checking, or "scraping" its targets, usually via HTTP. In the `global` section above we see that we will scrape targets every 15 seconds.

We can ignore the `alerting` and `rule_files` sections because they're not used in this lab.

In the `scrape_config` section we provide a list of services we want to monitor. The supplied configuration monitors Prometheus itself. This means that Prometheus, listening on port `9090` provides data about itself on the standard `metrics` URL endpoint. With the configuration above, Prometheus will query that URL every 15 seconds and store the data received.

Close the configuration file and run Prometheus with the command

```
./prometheus --config.file=prometheus.yml
```

Now we can see that Prometheus is working and see the raw data it shares by visiting the URL[1] `http://<server-ip>/metrics`. Of course, this URL is generally used by Prometheus itself to gather data that we can browse as we will in the next section.

2 Using the expression browser and graphing interface

Start by visiting the URL `http://<server-ip>:9090/targets`. Since our instance monitors one target, we should see it listed here. This shows us that Prometheus is successfully monitoring the target, but doesn't tell us about the target's reported data. To see that, start by selecting "Graph" from the top menu.

This view gives us two tabs: *Graph* and *Console*. We start on the Console tab. Our target supplies many different metrics and we can see the data for one by entering its name in the text entry box, for example `promhttp_metric_handler_requests_total`. Put this in the box and click "Execute" to see a summary of the data collected for this metric. Selecting the "Graph" Tab will show us a graph of how this metric has changed over time.

Play with the reporting tools a bit and then shut Prometheus down by entering `Ctrl-c` in your shell session.

3 Setting up a Node Exporter

In order for Prometheus to collect data from a target, that target needs to supply a service that provides data on request, usually on a `/metrics` URL. Much of the work in setting up a target is creating the target service to supply the data. For Linux hosts there is a standard target service, or *Node Exporter* that we can download and run.

Download the Node Exporter onto your backup server with the command

```
wget https://github.com/prometheus/node_exporter/releases/download/v1.0.0-rc.0/
             node_exporter-1.0.0-rc.0.linux-amd64.tar.gz
```

Unpack the downloaded tarball, `cd` into the unpacked directory and run the command `./node_exporter`. The exporter provides data at the url `http://<server-ip>:9100`.

To get Prometheus to use this, we need to configure the new target in its configuration file, `prometheus.yml`. Add the following to that file in the `scrape_config` section:

```
- job_name: 'node'
  static_configs:
  - targets: ['localhost:9100']
```

[1] Since you're probably off campus when you do this, you'll need to use OP's Citrix Viewer for this.

This file's contents are YAML, so be sure the indenting is consistent.

Start Prometheus and check the web UI again. You should see the new target on the targets status page, and you can query the Node Exporter's metrics in the expression browser. For example, `node_filesystem_free_bytes` shows free disk space, or `node_load1` shows the CPU load average.

4 Wrapping up

You're not necessarily expected to set up Prometheus monitoring for assessment in this paper, but you have the option to do so. If you're interested in the topic of monitoring, then looking further into Prometheus would be a good next step.