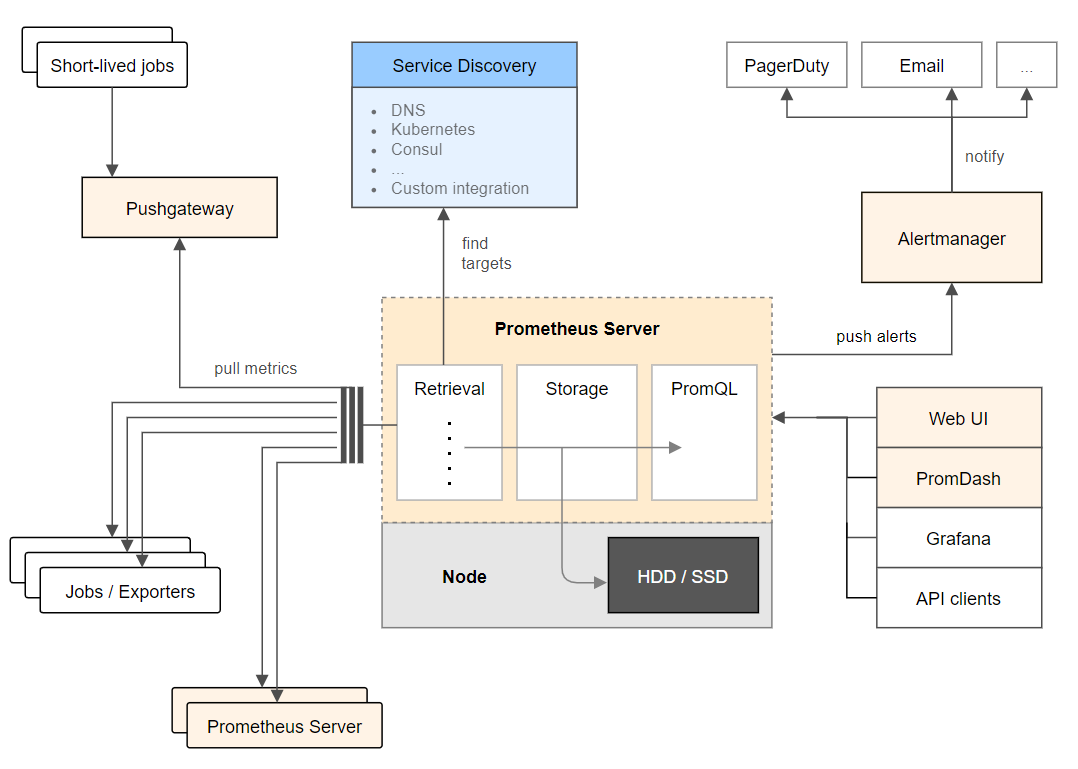
Prometheus

Open source systems monitoring and alerting toolkit.

## Prometheus Features:

* Multi-dimensional data model with time series data identified by metric name and key/value pair
* Flexible query language. (PromQL) to leverage dimensionality.
* Time series collection via pull model over HTTP.
* Pushing timeseries is supported via an intermediary gateway.
* Targets discovered via service discovery or static configuration.
* Multiple support for graphing and dashboard.
* Support for hierarchical and horizontal federation.

## Prometheus Architecture:



## Integrating Prometheus with Spring actuator:

Spring Boot uses Micrometer, an application metrics facade to integrate actuator metrics with external monitoring systems.

To integrate actuator with Prometheus, you need to add the micrometer-registry-prometheus dependency -

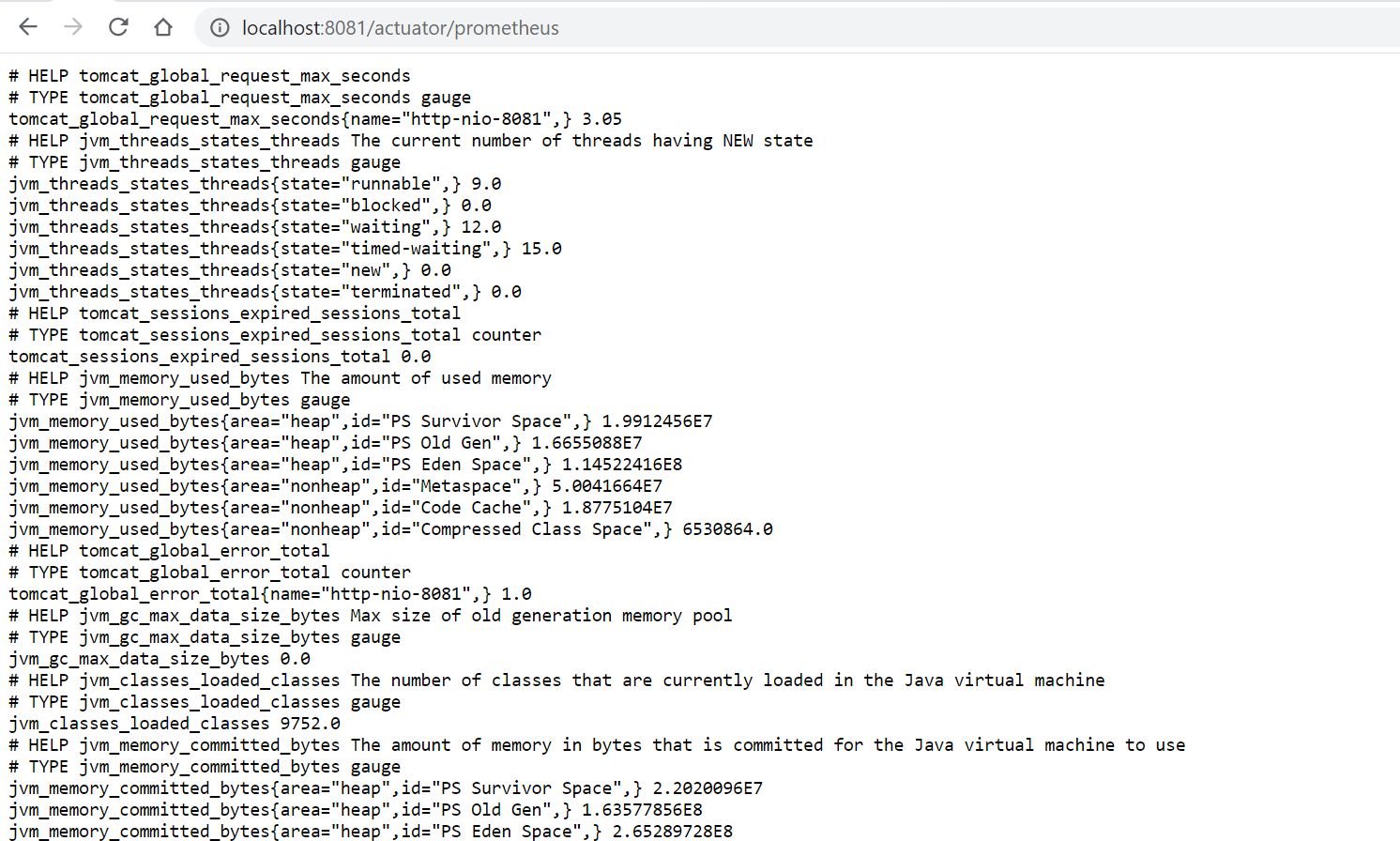
<dependency>

<groupId>io.micrometer</groupId>

<artifactId>micrometer-registry-prometheus</artifactId>

</dependency>

Adding above dependency exposes metrics on <application-host:port>/actuator/prometheus as shown below. The prometheus endpoint exposes metrics data in a format that can be scraped by a Prometheus server.



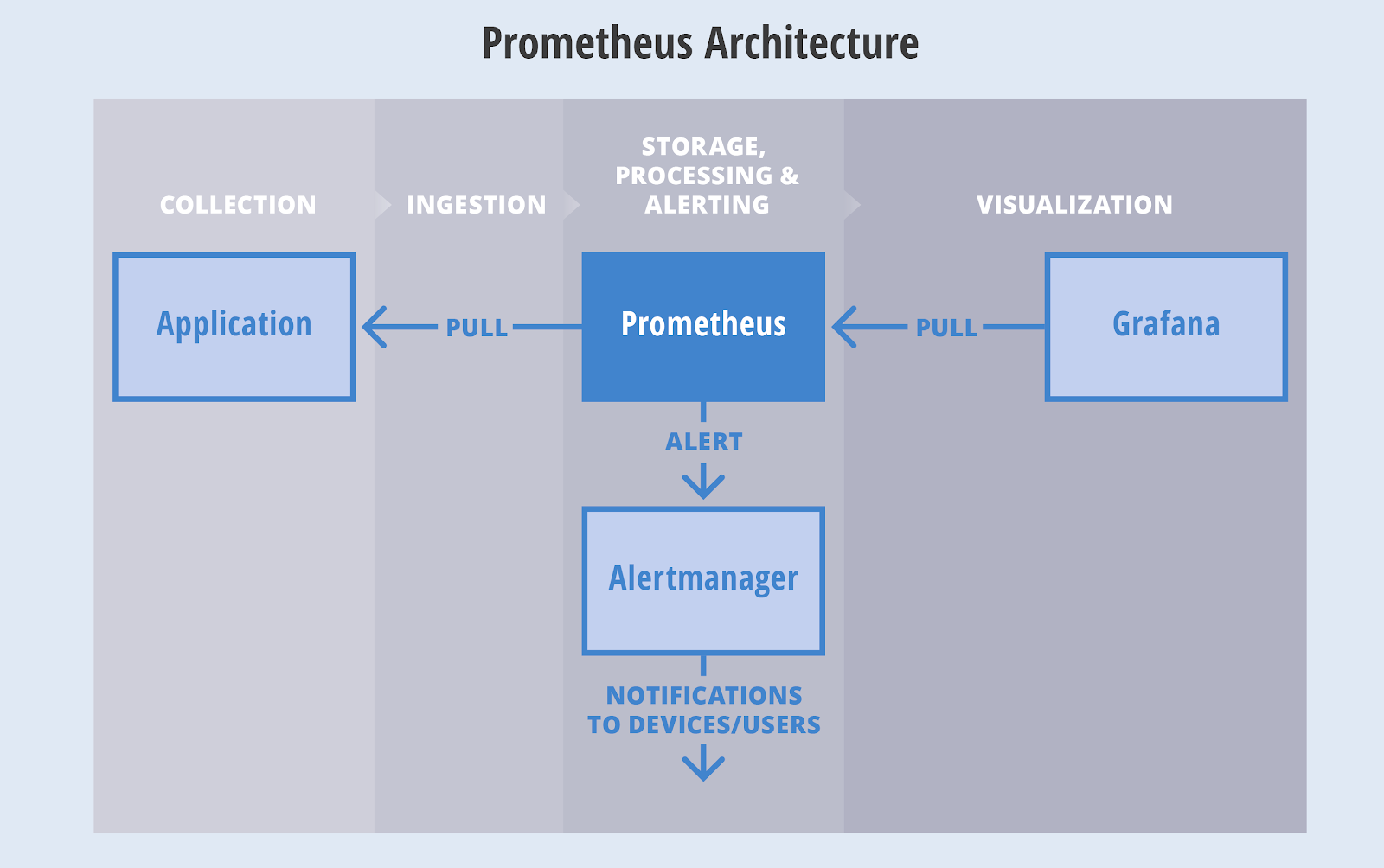
## Prometheus with Grafana and Alert Manager:

Prometheus is time-series database to store all the metrics data with simple user interface where you can visualize, query, and monitor all the metrics.

Grafana gives you rich graphical interface with beautiful graphs. It also lets you set alert rules based on your metrics data. When an alert changes state, it can notify you over email, slack, or various other channels.

AlertManager handles alerts sent by Prometheus server and notifies to end user.

Below diagram shows relationship between end application, Prometheus server, Grafana and Alert Manager.



## Setting up Prometheus:

Prometheus can be downloaded using docker image as given below:

$ docker pull prom/prometheus

## Setting Prometheus config files:

We need to configure Prometheus to scrape metrics data from Spring Boot Actuator’s /prometheus endpoint. This is done with help of *prometheus.yml*.

A sample *prometheus.yml* file is shown below, details will be explained in further sections.

# 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: ['host.docker.internal:9093']

# Load rules once and periodically evaluate them according to the global 'evaluation\_interval'.

rule\_files:

- "alert.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: ['127.0.0.1:9090']

- job\_name: 'spring-actuator'

metrics\_path: '/actuator/prometheus'

scrape\_interval: 5s

basic\_auth:

username: admin

password: admin

static\_configs:

- targets: ['host.docker.internal:8081']

To make this file accessible to Prometheus server, we need to rebake standard Prometheus image. This can be done using *Dockerfile.*

Create a *Dockerfile* with following content in same directory as of *prometheus.yml*

FROM prom/prometheus

ADD prometheus.yml /etc/prometheus/

ADD alert.yml /etc/prometheus/

Before baking image create one more file called *alert.yml* in same directory as that of *prometheus.yml* and *Dockerfile*. Its details and requirements will be explained in subsequent sections.

Content of *alert.yml* are as below:

groups:

-

name: demo-alert

rules:

-

alert: high\_load

annotations:

description: "Instance CPU usage exceeded"

summary: "Instance under high load"

expr: "system\_cpu\_usage > 0.2"

labels:

severity: page

-

alert: service\_down

annotations:

description: "Service instance is down"

summary: "Service instance is down"

expr: up==0

for: 1m

## Bake custom Prometheus image:

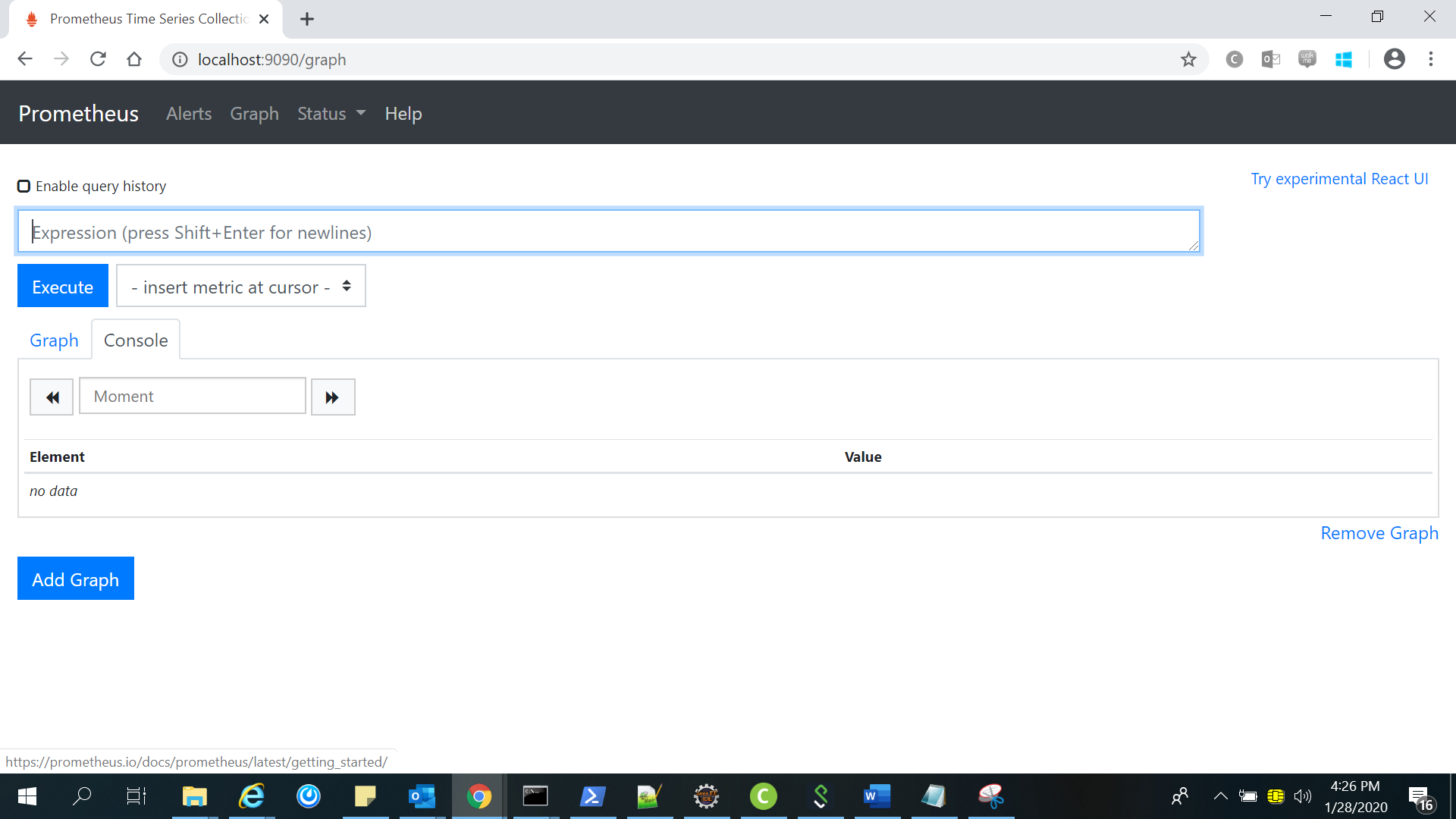
* The above *prometheus.yml* file tells Prometheus the details about application where it can find /actuator/Prometheus endpoint. It also specifies the scraping interval, authentication details (if any) needed for accessing the endpoint and IP and port of endpoint. ‘host.docker.internal’ tells docker to find endpoint on host machine rather than on container network.
* Above *alert.yml* defines two alerts for our Prometheus server.
  + The first alert is ‘high\_load’ which raises an alert if cpu usage exceeds more than 20%.
  + Second alert is ‘service\_down’ which raises an alert if our service endpoint is unavailable for more than 1 minute.
* *Dockerfile* will place *prometheus.yml* and *alert.yml* into our custom image of Prometheus*.*

Commands to create custom image and run it are given below:

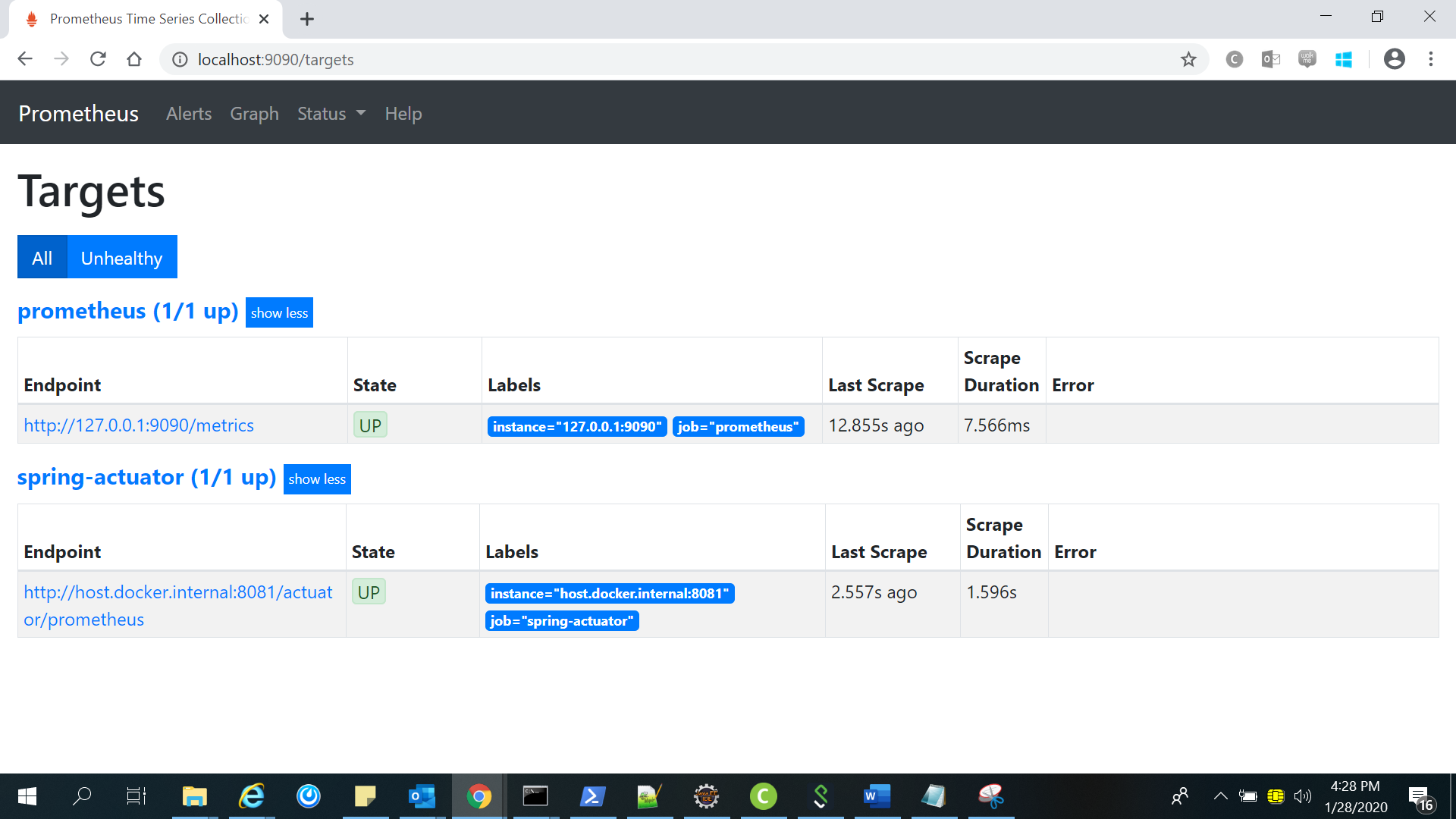
docker build -t my-prometheus .

docker run -p 9090:9090 my-prometheus

Second command will make Prometheus accessible on port 9090.

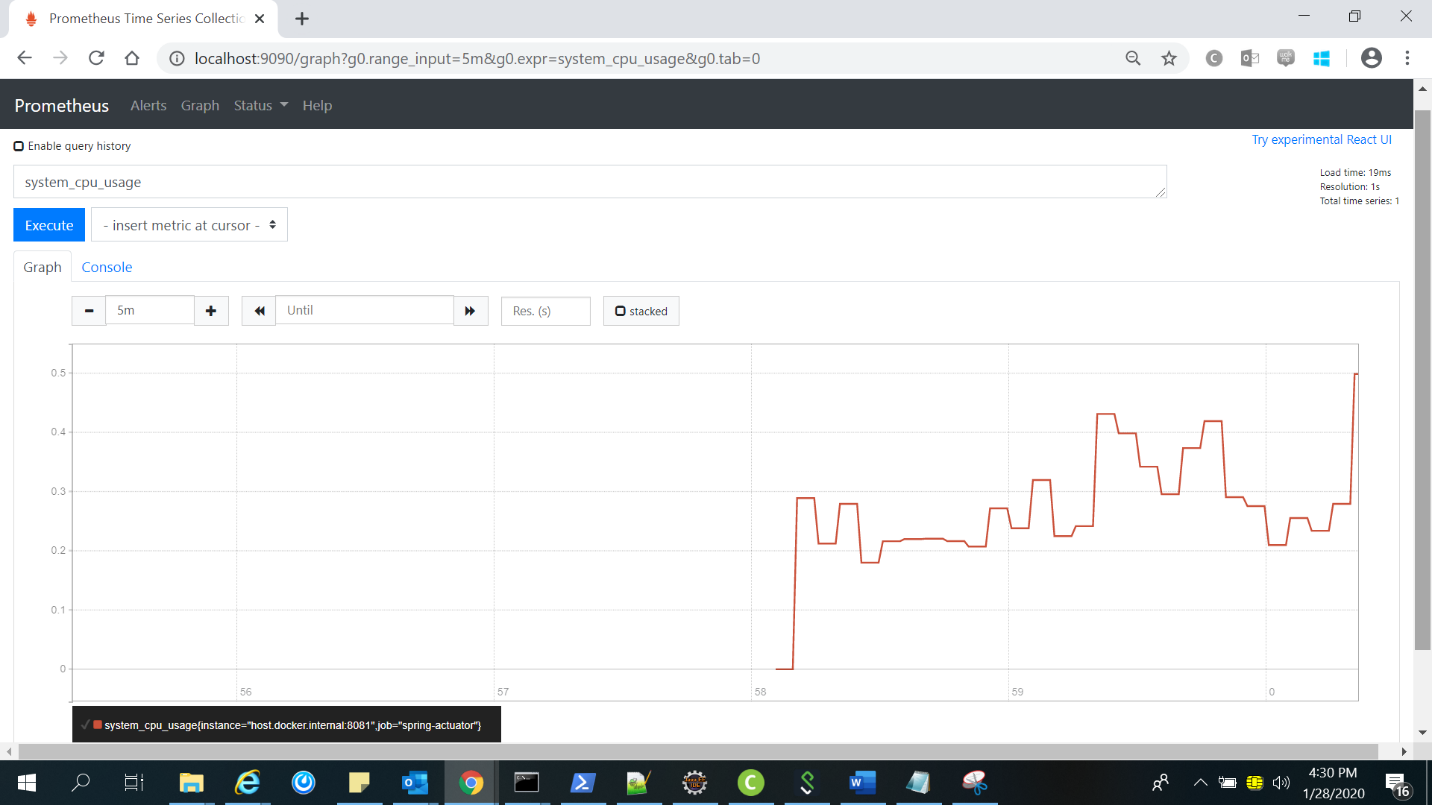


Navigate to Status > Targets



If you see State down, ensure that application endpoint is accessible over given URL.

You can view different metrics by selecting and executing metrics as shown below. Selecting graph and appropriate time scale will display a graph.

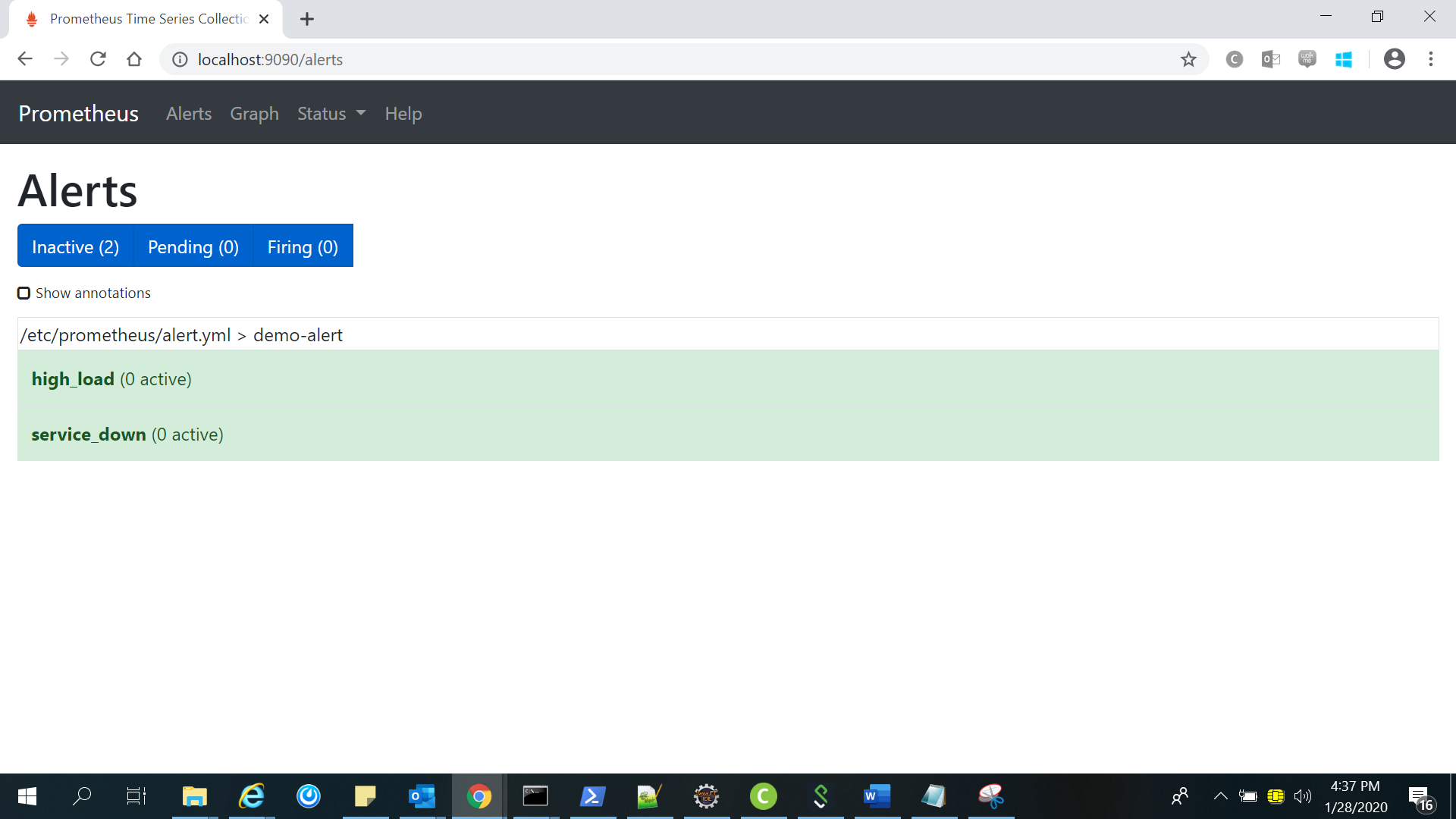


## Prometheus Alerts:

The alert section will show the alerts configured by us via *alert.yml* file. Following section in *Prometheus.yml* tells Prometheus the details of different alerts.

rule\_files:

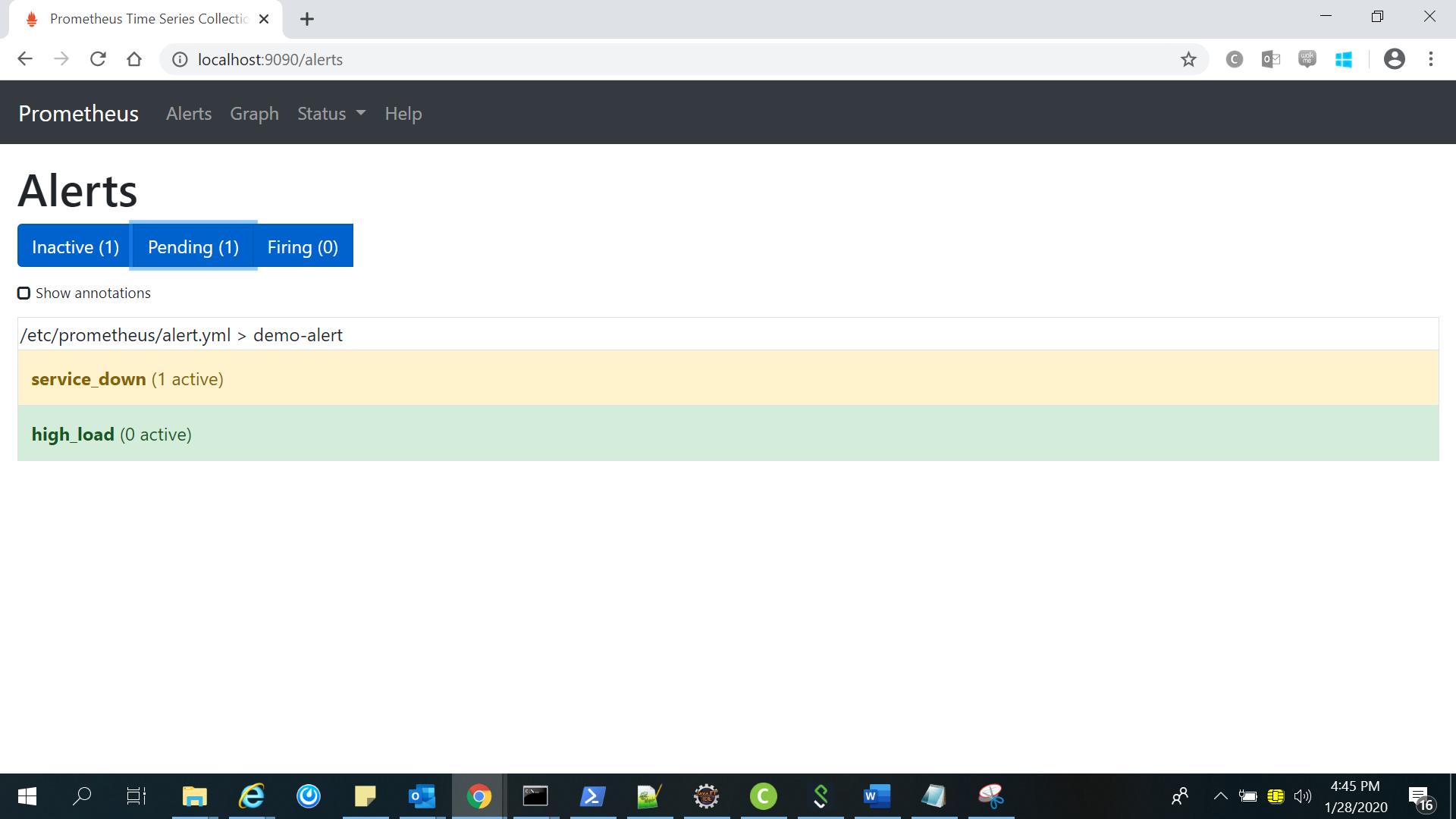
- "alert.yml"

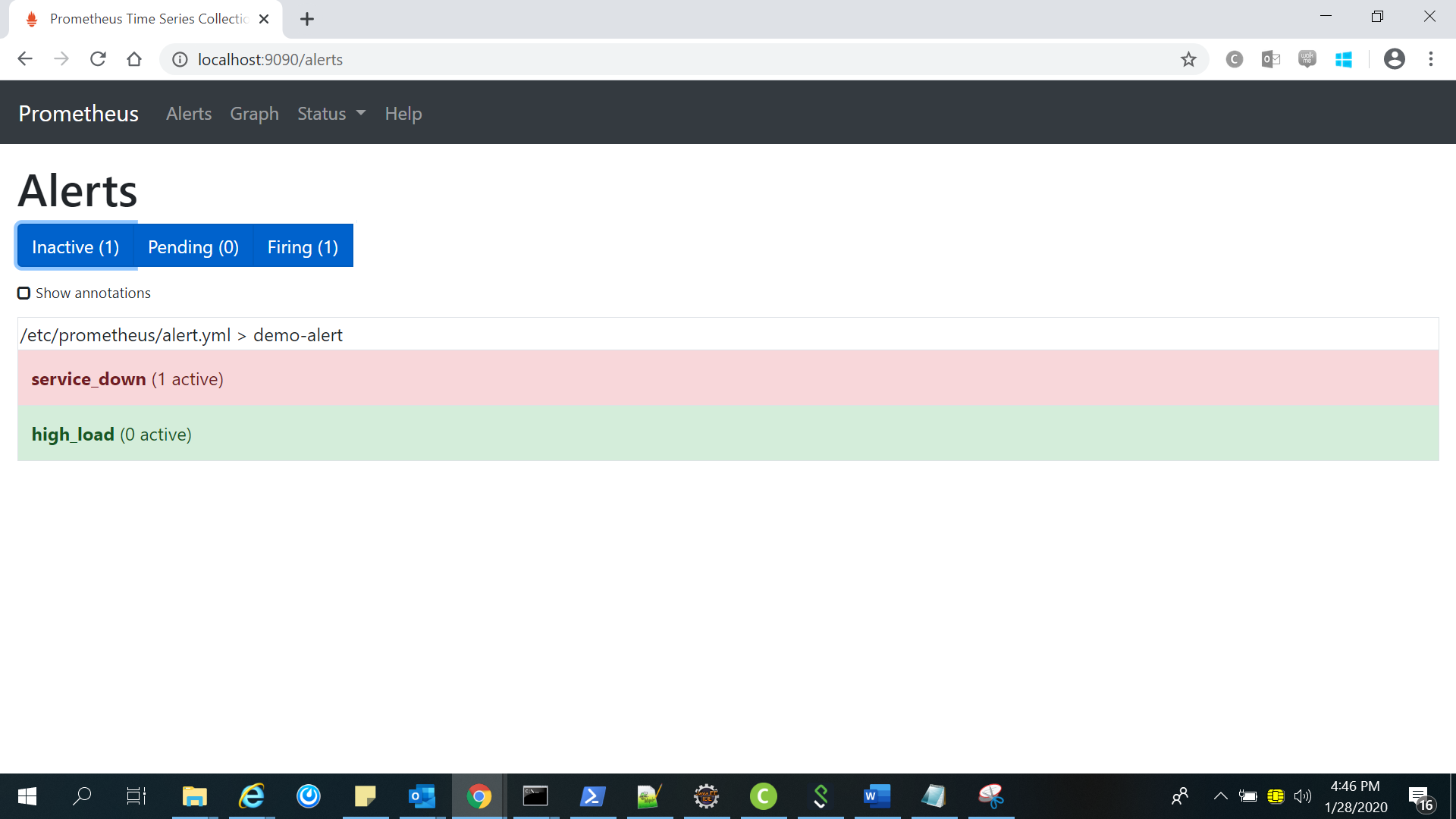


Alerts have three states:

1. Inactive: The alert is not active.
2. Pending: The alert has met test condition but is waiting for the duration specified in the ‘for’clause.
3. Firing: The alert has met the test expression and has been Pending for longer than the duration of the ‘for’ clause.

Turning down the application can make ‘service\_down’ alert to Pending and then to Firing state.





## Alert Manager:

Alert manager helps alert users as soon as error has occurred. Alerts are essential for monitoring. If any alert condition is met, Prometheus sends alert to alert manager. Alert manager manages alerts through its pipeline of silencing, inhibition, grouping and sending out notifications.

There are different ways to configure alert manager on different platforms. It can be downloaded from: <https://prometheus.io/download/>

Alerting with Prometheus setup steps are mentioned below:

1. Setup and configure AlertManager:

Download binary from given URL based on your environment.

2. Configure the config file on Prometheus so it can talk to the AlertManager:

In our case it has been achieved with following section in *Prometheus.yml*.

#Alertmanager configuration

alerting:

alertmanagers:

- static\_configs:

- targets: ['host.docker.internal:9093']

3. Define alert rules in Prometheus server configuration: This has been already done using *alert.yml* file.

4. Define alert mechanism in AlertManager to send alerts via Slack, Email, PagerDuty etc.:

This is done using *alertmanager.yml* file. A sample file is given below:

global:

smtp\_smarthost: <smtp-server>

smtp\_from: "alerts@gmail.com"

smtp\_auth\_username: "test@gmail.com"

smtp\_auth\_identity: " test@gmail.com "

smtp\_auth\_password: "!@@#$%%DFDSFG"

route:

group\_by: ['alertname']

group\_wait: 10s

group\_interval: 5m

repeat\_interval: 3h

routes:

receiver: alert-team

receivers:

- name: 'alert-team'

email\_configs:

- to: support-team@siemens.com

inhibit\_rules:

- source\_match:

severity: 'critical'

target\_match:

severity: 'warning'

equal: ['alertname', 'cluster', 'service']

Above configuration will send email alerts to [support-team@siemens.com](mailto:support-team@siemens.com) from account [test@siemens.com](mailto:test@siemens.com). The alerts will be grouped by alertname for every 10 seconds before sending out an email alert.

Sample email alert:

