Metrics

Raúl Estrada

Octubre 2020

Metrics

For ingesting, storing and alerting on our metrics, we shall explore another, quite popular open-source project called Prometheus:

> **Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud.**
>
> **Prometheus's main features are:**
>
> **-a multi-dimensional data model (time series identified by metric name and key/value pairs)**
>
> **- a flexible query language to leverage this dimensionality**
>
> **- no reliance on distributed storage; single server nodes are autonomous**
>
> **- time series collection happens via a pull model over HTTP**
>
> **- pushing time series is supported via an intermediary gateway**
>
> **- targets are discovered via service discovery or static configuration**
>
> **- multiple modes of graphing and dashboarding support**
>
> **--https://prometheus.io/docs/introduction/overview/emphasis>**

Even though it is the kind of system that takes care of pretty much everything, the project still follows the popular UNIX philosophy of modular development. Prometheus is composed of multiple components, each providing a specific function:

- the main Prometheus server which scrapes and stores time series data

- client libraries for instrumenting application code

- a push gateway for supporting short-lived jobs

- a GUI-based dashboard builder based on Rails/SQL

- special-purpose exporters (for HAProxy, StatsD, Ganglia, etc.)

- an (experimental) alertmanager

- a command-line querying tool

--https://prometheus.io/docs/introduction/overview/

Ingesting and storing metrics with Prometheus

Our second EC2 instance is going to host the Prometheus service alongside Jenkins (we will come to that shortly), thus a rather appropriate name would be promjenkins.

As a start, download and extract Prometheus and Alertmanager in `/opt/prometheus/server` and `/opt/prometheus/alertmanager` respectively (ref: https://prometheus.io/download).

We create a basic configuration file for the Alertmanager in `/opt/prometheus/alertmanager/alertmanager.yml` (replace e-mail addresses as needed):

Note

Please refer to: https://github.com/PacktPublishing/Implementing-DevOps-on-AWS/blob/master/5585_07_CodeFiles/promjenkins/opt/prometheus/alertmanager/alertmanager.yml

Copy

```
global:
  smtp_smarthost: 'localhost:25'
  smtp_from: 'alertmanager@example.org'

route:
  group_by: ['alertname', 'cluster', 'service']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 1h
  receiver: team-X-mails

receivers:
- name: 'team-X-mails'
  e-mail_configs:
  - to: 'team-X+alerts@example.org'
    require_tls: false
```

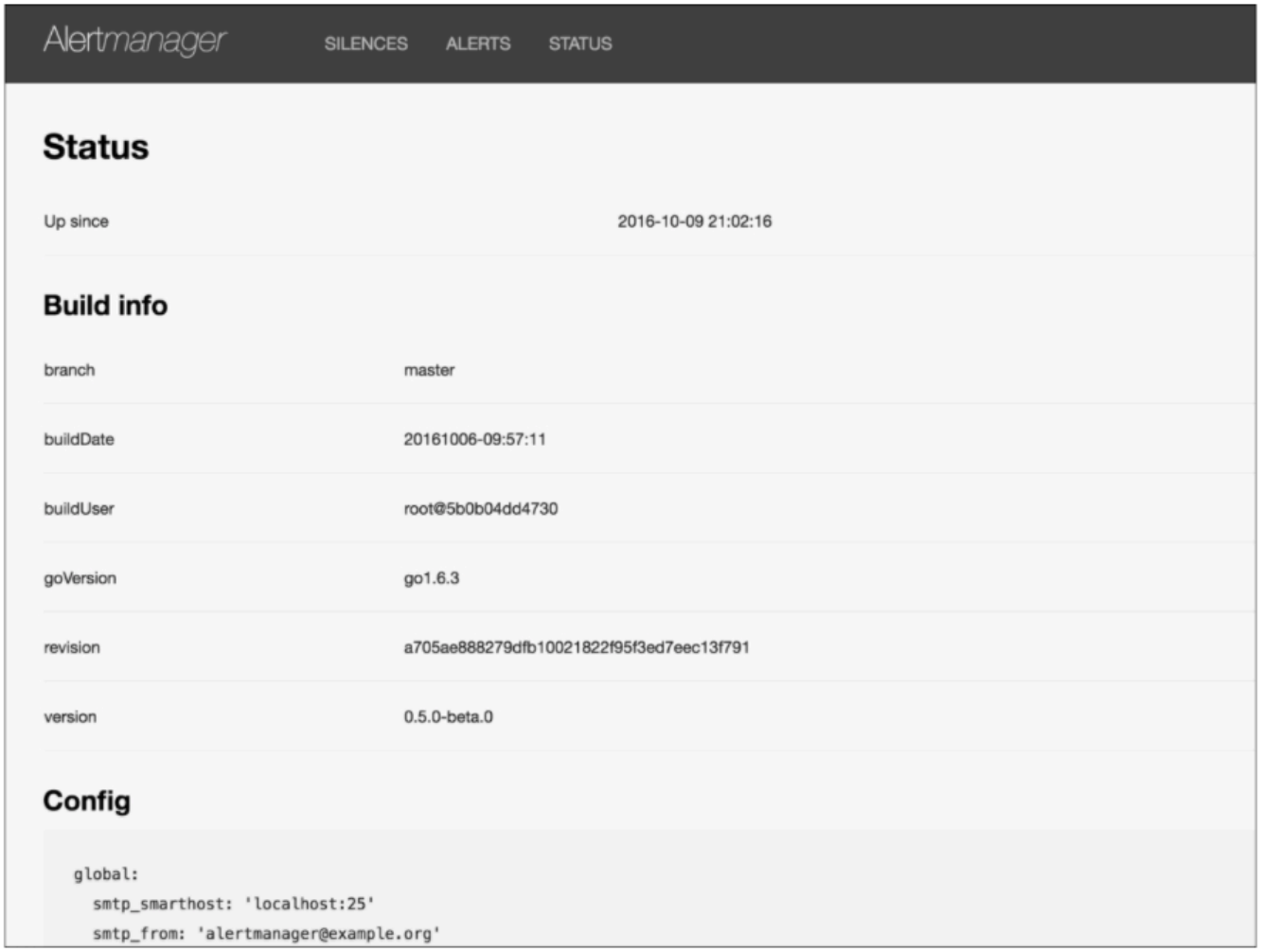
This will simply e-mail out alert notifications.

Start the service:

Copy

```
# cd /opt/prometheus/alertmanager
# (./alertmanager 2>&1 | logger -t prometheus_alertmanager)&
```

Ensure the default `TCP:9093` is allowed, then you should be able to get to the dashboard at `http://$public_IP_of_promjenkins_node:9093/#/status` :



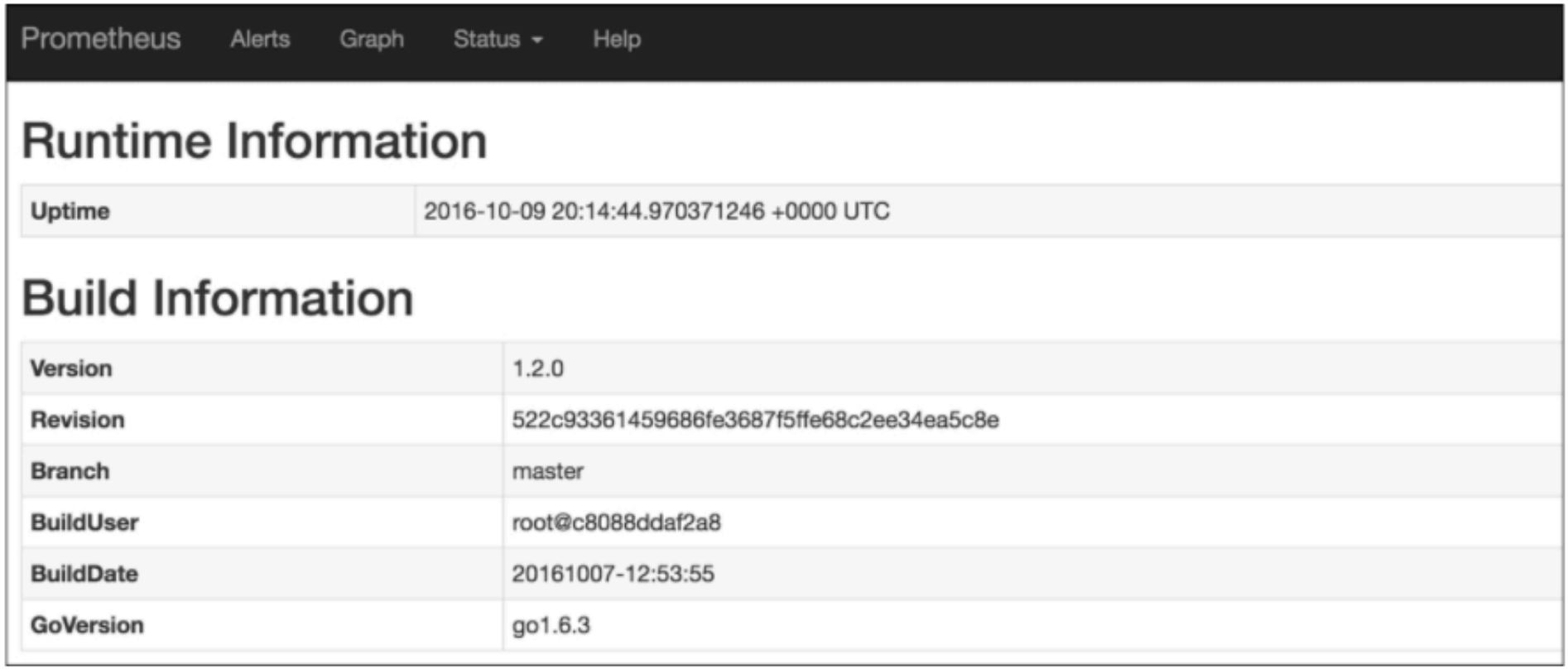
Back to the Prometheus server, the default `/opt/prometheus/server/prometheus.yml` will suffice for now.

We can start the service:

Copy

```
# cd /opt/prometheus/server
# (./prometheus -alertmanager.url=http://localhost:9093 2>&1 | logger -t prometheus_server)
```

Open up `TCP:9090` , then try `http://$public_IP_of_promjenkins_node:9090/status` :



We are ready to start adding hosts to be monitored. That is to say targets for Prometheus to scrape.

Prometheus offers various ways in which targets can be defined. The one most suitable for our case is called `ec2_sd_config` (ref: https://prometheus.io/docs/operating/configuration/#<ec2_sd_config>). All we need to do is provide a set of API keys with read-only EC2 access (**AmazonEC2ReadOnlyAccess** IAM policy) and Prometheus will do the host discovery for us (ref: https://www.robustperception.io/automatically-monitoring-ec2-instances).

We append the `ec2_sd_config` settings to: `/opt/prometheus/server/prometheus.yml` :

Note

Please refer to: https://github.com/PacktPublishing/Implementing-DevOps-on-AWS/blob/master/5585_07_CodeFiles/promjenkins/opt/prometheus/server/prometheus.yml

Copy

```
- job_name: 'ec2'
  ec2_sd_configs:
    - region: 'us-east-1'
      access_key: 'xxxx'
      secret_key: 'xxxx'
      port: 9126
  relabel_configs:
    - source_labels: [__meta_ec2_tag_Name]
      regex: ^webserver
      action: keep
```

Copy

```
- job_name: 'ec2'
  ec2_sd_configs:
    - region: 'us-east-1'
      access_key: 'xxxx'
      secret_key: 'xxxx'
      port: 9126
  relabel_configs:
    - source_labels: [__meta_ec2_tag_Name]
      regex: ^webserver
      action: keep
```

We are interested only in any instances in the `us-east-1` region with a name matching the `^webserver` regex expression.

Now let us bring some of those online.

Gathering OS and application metrics with Telegraf

We will be using the pull method of metric collection in Prometheus. This means that our clients (targets) will expose their metrics for Prometheus to scrape.

To expose OS metrics, we shall deploy InfluxData's Telegraf (ref: https://github.com/influxdata/telegraf).

It comes with a rich set of plugins, which will provide for a good deal of metrics. Should you need more, you have the option to write your own (in Go) or use the `exec` plugin which will essentially attempt to launch any type of script you point it at.

As for application metrics, we have two options (at least):

- Build a metrics API endpoint in the application itself
- Have the application submit metrics data to an external daemon (StatsD as an example)

Incidentally, Telegraf comes with a built-in StatsD listener, so if your applications already happen to have StatsD instrumentation, you should be able to simply point them at it.

Following on from the ELK example, we will extend the EC2 user data script to get Telegraf on our the Auto Scale Group instances.

Note

Please refer to: https://github.com/PacktPublishing/Implementing-DevOps-on-AWS/blob/master/5585_07_CodeFiles/webserver/user_data.sh

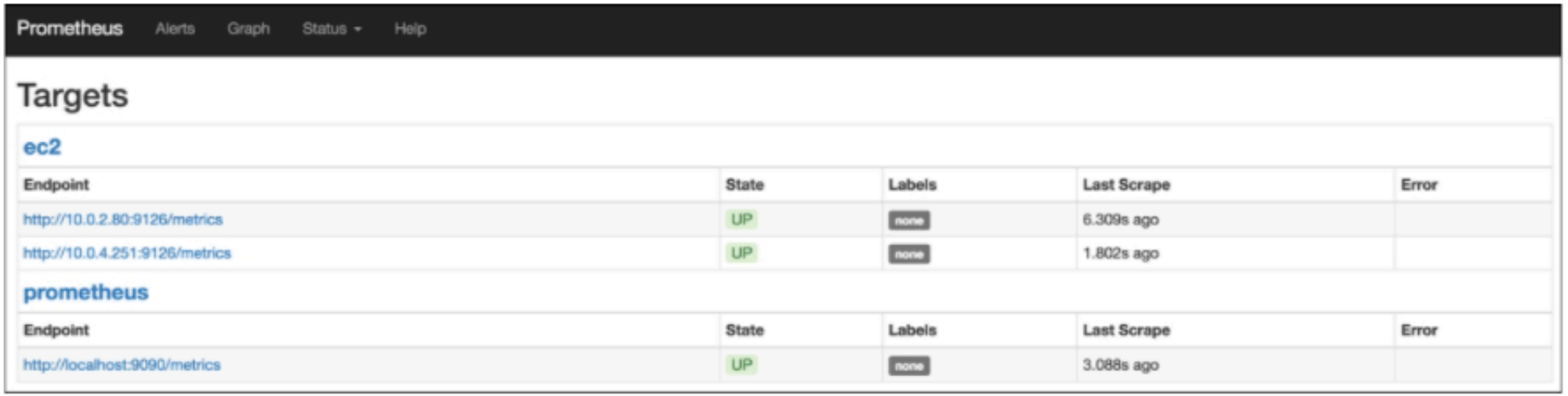
We append:

```
yum -y install https://dl.influxdata.com/telegraf/releases/telegraf-1.0.1.x86_64.rpm

cat << EOF > /etc/telegraf/telegraf.conf
[global_tags]
[agent]
  interval = "10s"
  round_interval = true
  metric_batch_size = 1000
  metric_buffer_limit = 10000
  collection_jitter = "0s"
  flush_interval = "10s"
  flush_jitter = "0s"
  precision = ""
  debug = false
  quiet = false
  hostname = ""
  omit_hostname = false
[[outputs.prometheus_client]]
  listen = ":9126"
[[inputs.cpu]]
```

```
  percpu = true
  totalcpu = true
  fielddrop = ["time_*"]
[[inputs.disk]]
  ignore_fs = ["tmpfs", "devtmpfs"]
[[inputs.diskio]]
[[inputs.kernel]]
[[inputs.mem]]
[[inputs.processes]]
[[inputs.swap]]
[[inputs.system]]
EOF

service telegraf start
```

The important one here is `outputs.prometheus_client` with which we turn Telegraf into a Prometheus scrape target. By all means check the default configuration file if you'd like to enable more metrics during this test (ref: https://github.com/influxdata/telegraf/blob/master/etc/telegraf.conf)

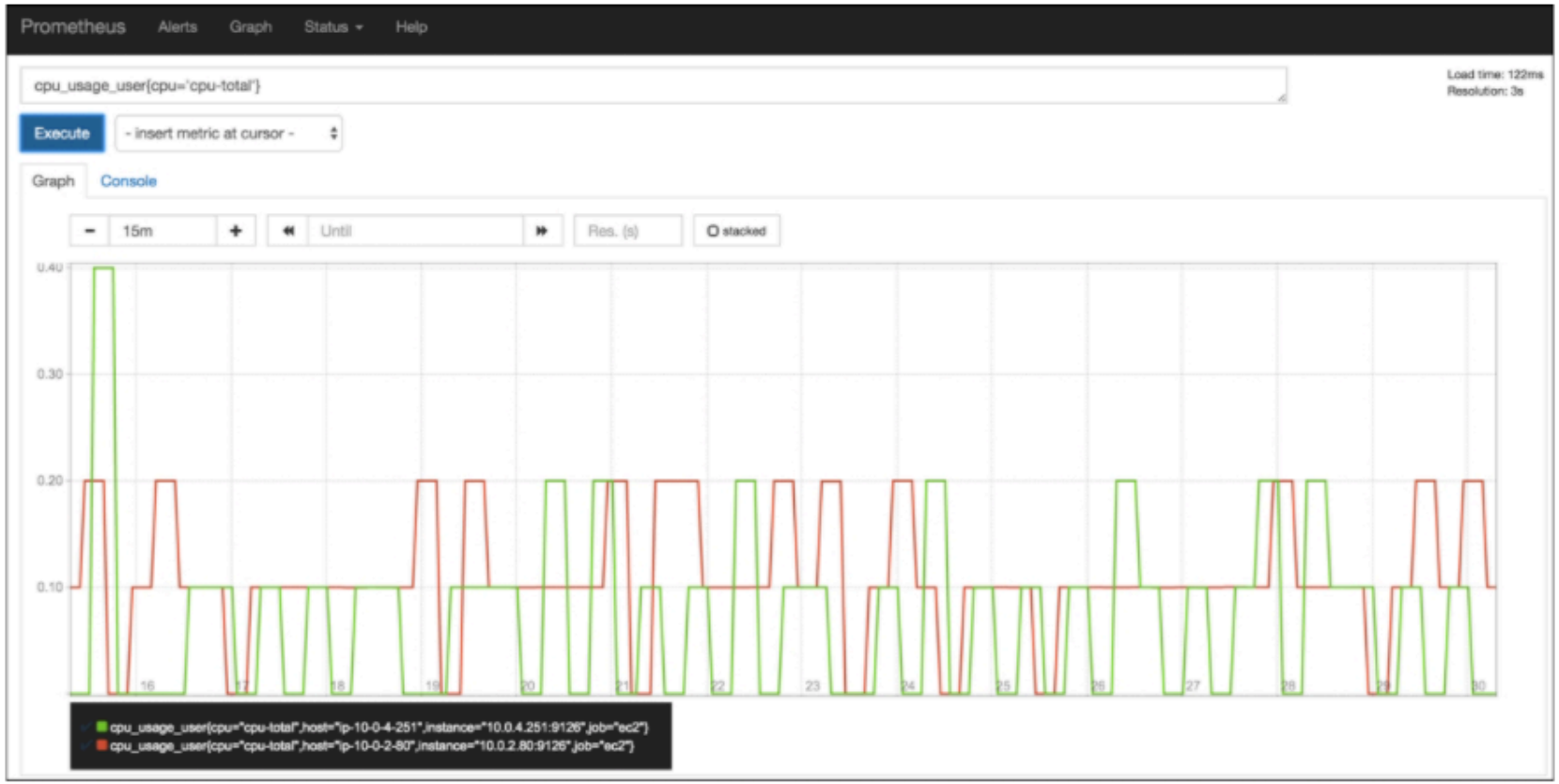
Next, check that TCP: `9126` is allowed into the Auto Scale Group security group, then launch a couple of nodes. In a few moments, you should see any matching instances listed in the targets dashboard (ref: `http://$public_IP_of_promjenkins_node:9090/targets`):



We see the new hosts under the **ec2** scrape job which we configured earlier.

Visualizing metrics with Grafana

It is true that Prometheus is perfectly capable of visualizing the data we are now collecting from our targets, as seen here:



In fact, this is the recommended approach for any ad-hoc queries you might want to run.

Should you have an appetite for dashboards however, you would most certainly appreciate **Grafana - The 8th Wonder** (ref: http://grafana.org)

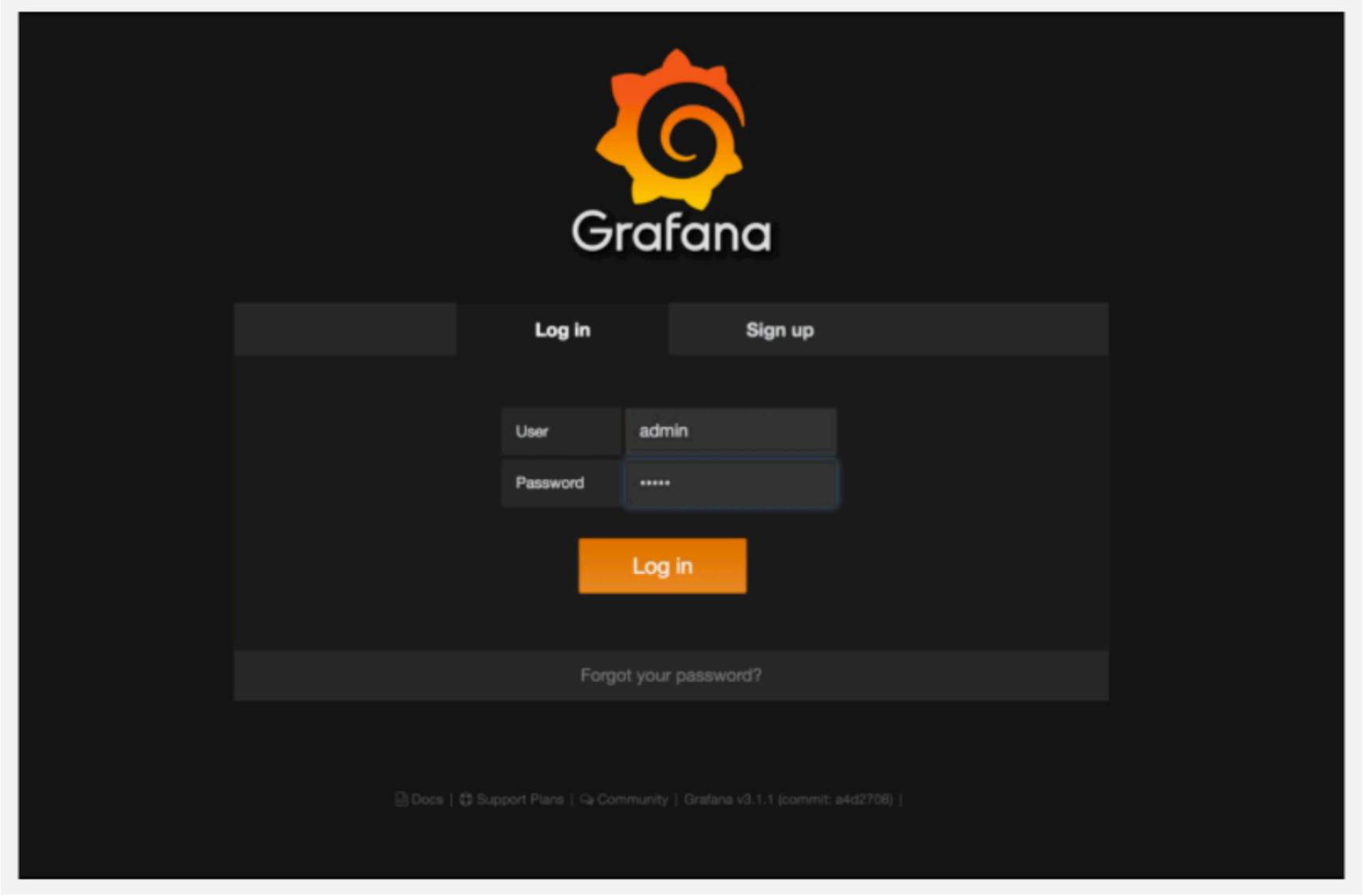
Check this out to get a feel for the thing: http://play.grafana.org

1. So, yes, Grafana, let us install the service on the promjenkins node:

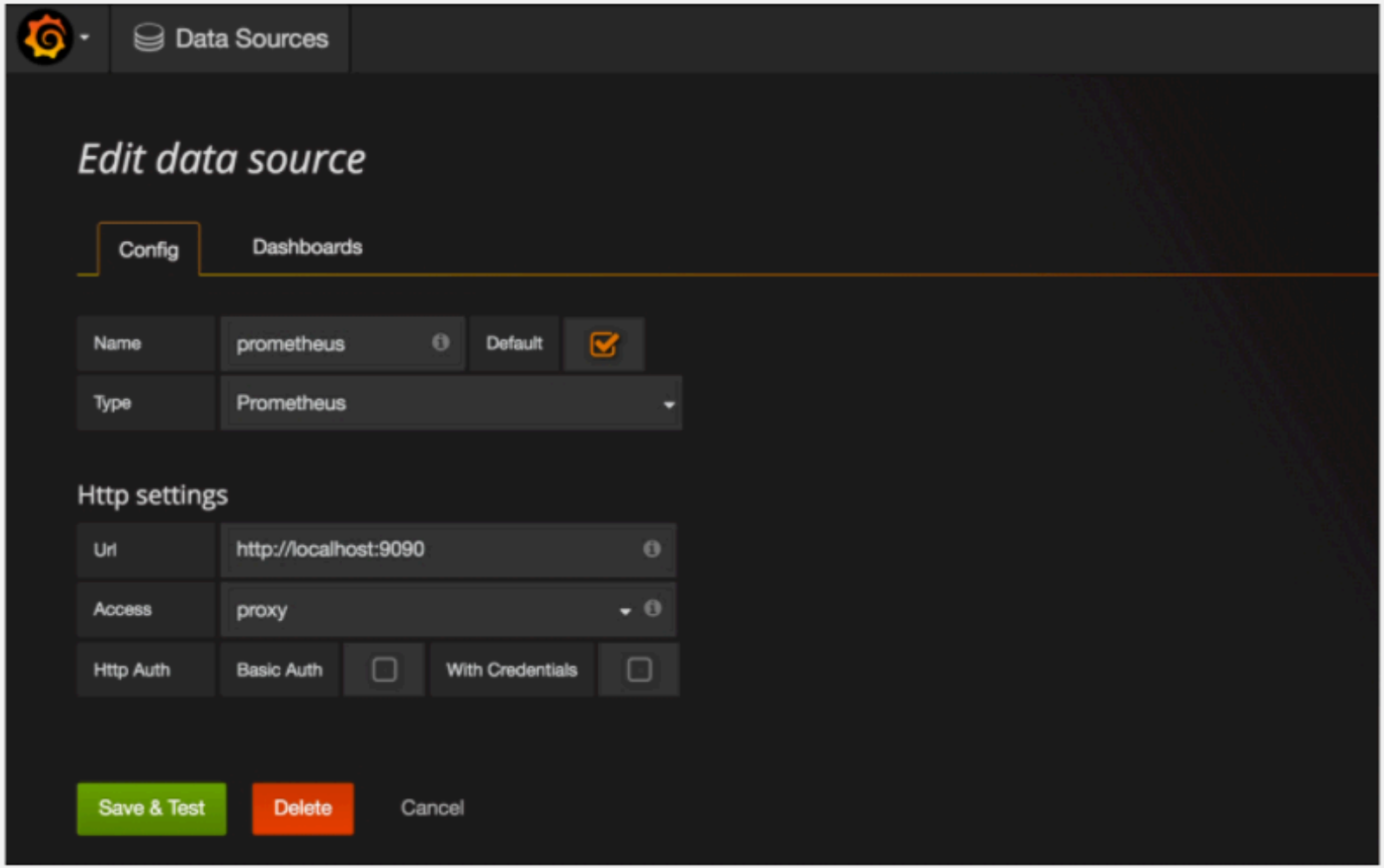
Copy

```
# yum -y install https://grafanarel.s3.amazonaws.com/builds/
        grafana-3.1.1-1470047149.x86_64.rpm
# service grafana-server start
```

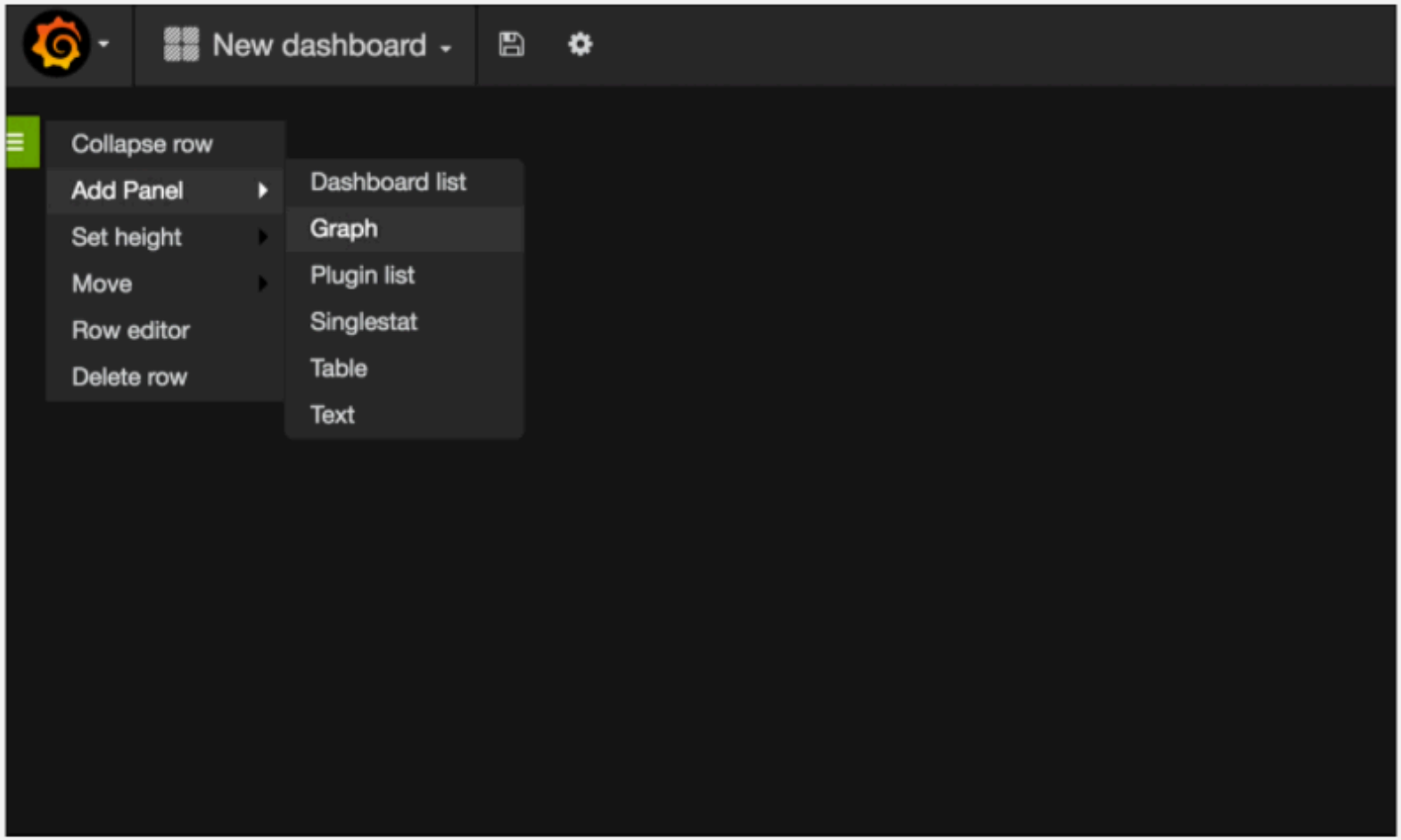
The default Grafana port is TCP: `3000` , auth `admin:admin` . After updating the relevant security group, we should be able to see the screen at: `http://$public_IP_of_promjenkins_node:3000` :


Grafana
Log in
Sign up
User
admin
Password
Log in
Forgot your password?
Docs | Support Plans | Community | Grafana v3.1.1 (commit: a4d2708) |

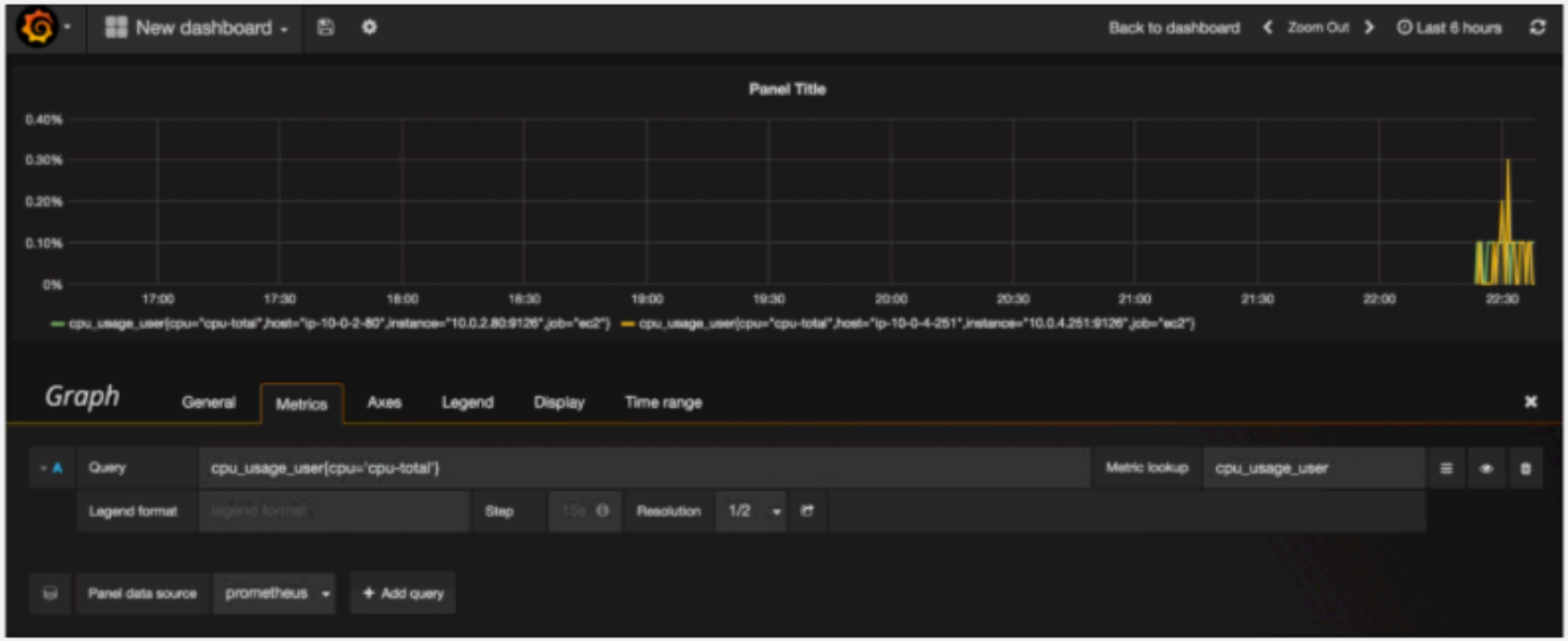
2 After logging in, first we need to create a **Data Sources** for our **Dashboards**:



3. Back at the home screen, choose to create a new dashboard, then use the green button on the left to **Add Panel** then a **Graph**:



4. Then, adding a basic CPU usage plot looks like this:



At this point I encourage you to browse http://docs.grafana.org to find out more about templating, dynamic dashboards, access control, tagging, scripting, playlist, and so on.