PRACTICAL PORTS

Prometheus Installation & Setup

BY BENEDICT REUSCHLING

For a long time, monitoring systems and services running on them has been done as part of a sysadmin's job. This serves many purposes:

- is the system generally reachable (Availability),
- answers the question of what did the system do last week on Sunday at 4 a.m.? (Metrics)
- alerts IT personnel (often at ungodly hours) about unusually high processes and other events out of the ordinary (Alerting)

Sysadmins typically collect these metrics at a central location for further study and visualization, which helps more than logging into individual systems and running `tail -F /var/log/messages` or other logfiles. Finding when a problem started by seeing a spike in CPU usage or a dramatic decline in available disk space at the beginning of the month is clearly visible from a graph. When alerts are configured, notifications are sent about certain events (is the system reachable at all?) or if certain thresholds are reached (only 10% free disk space left). All of these have traditionally been done by software such as Munin, CheckMK, Nagios or Zabbix among others.

Prometheus is a fairly young monitoring project in the open source space. It did become a well-established solution—mainly in the Kubernetes and Cloud space—but is also usable in other environments. The setup was surprisingly easy for me, after having used a combination of telegraf, InfluxDB, and Grafana for a long time. Grafana is also used here as well for the visualization of Dashboards. InfluxDB, as the name suggests, serves as the central storage place for the collected metrics from which Grafana pulls the data. Sending the data was done by Telegraf running on each machine, sending its metrics to InfluxDB at regular intervals (i.e., every 10 seconds).

Prometheus is a fairly young monitoring project in the open source space.

Prometheus' architecture is similar: a central Prometheus server for transforming, storing, and streamlining the received data, with so-called `node_exporters` collecting the metrics on client machines. Again, Grafana uses the Prometheus data and can display them with some ready-made dashboards to impress your colleagues and be useful at the same time.

Other components include an alert manager to send various configurable notification types (email, SMS, pager, chat messages) when certain events occur. To query the data, Prometheus offers its own query language called PromQL that Grafana understands and uses for the dashboard content. Users can also write their own ad-hoc queries using PromQL, allowing for quick searches without having to build a dashboard first.

The logic by which metrics are extracted, formatted, and sent is coded into the exporters. There are several different exporters available for specific software like databases. Applications like RabbitMQ, GitLab, and Grafana itself allow the export of their own application states into a Prometheus-compatible format for monitoring. Written in Go, Prometheus is highly scalable and does not need too many resources when running.

In this article, we'll setup a FreeBSD based Prometheus server and have clients (Linux and FreeBSD) send system metrics to it via the **`node_exporter`**. We'll also use Grafana to visualize the data in an existing dashboard that we are importing for this purpose.

Users can also write their own ad-hoc queries using PromQL.

Prometheus Setup

First, we setup the Prometheus instance on a FreeBSD system. Freshly installed and connected to the network, we begin by creating the dataset where Prometheus stores its data in **`/var/db/prometheus`**. A directory is created by the port automatically, so this step is not strictly necessary. However, running it on ZFS as a separate dataset with properties like compression is good practice.

```
# zfs create -o compression=zstd sys/var/db/prometheus
# pkg install prometheus node_exporter
```

To extract system level metrics from the host, we install the **`node_exporter`** on our Prometheus host and all other machines we want to monitor. The Prometheus port installs a default configuration file called **`prometheus.yml`** in the **`/usr/local/etc`** path. We're going to modify it to fit our needs. The syntax for the configuration file is done in YAML, so be extra careful to avoid tabs and use the proper indentation with spaces.

```
prometheus.yml:
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is ev-
ery 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).

# 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"
    static_configs:
     - targets:
        - mistwood:9090

  - job_name: bdc
    static_configs:
     - targets:
        - mistwood:9100

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

Before diving into the configuration bits, we enable the **`prometheus service`** and the **`node_exporter`** on FreeBSD to start upon boot.

```
# service prometheus enable
# service node_exporter enable
```

Prometheus provides a web-based interface for querying and displaying the metrics exported by the systems (called targets in Prometheus lingo). I provide an extra argument to the start of the Prometheus service to define at which port the web interface should be reachable on my host called mistwood.

```
# sysrc prometheus_args="--web.listen-address=mistwood:9090"
```

Once we have that, we can start the **`prometheus service`** and the **`node_exporter`** like this:

```
# service prometheus start
# service node_exporter start
```

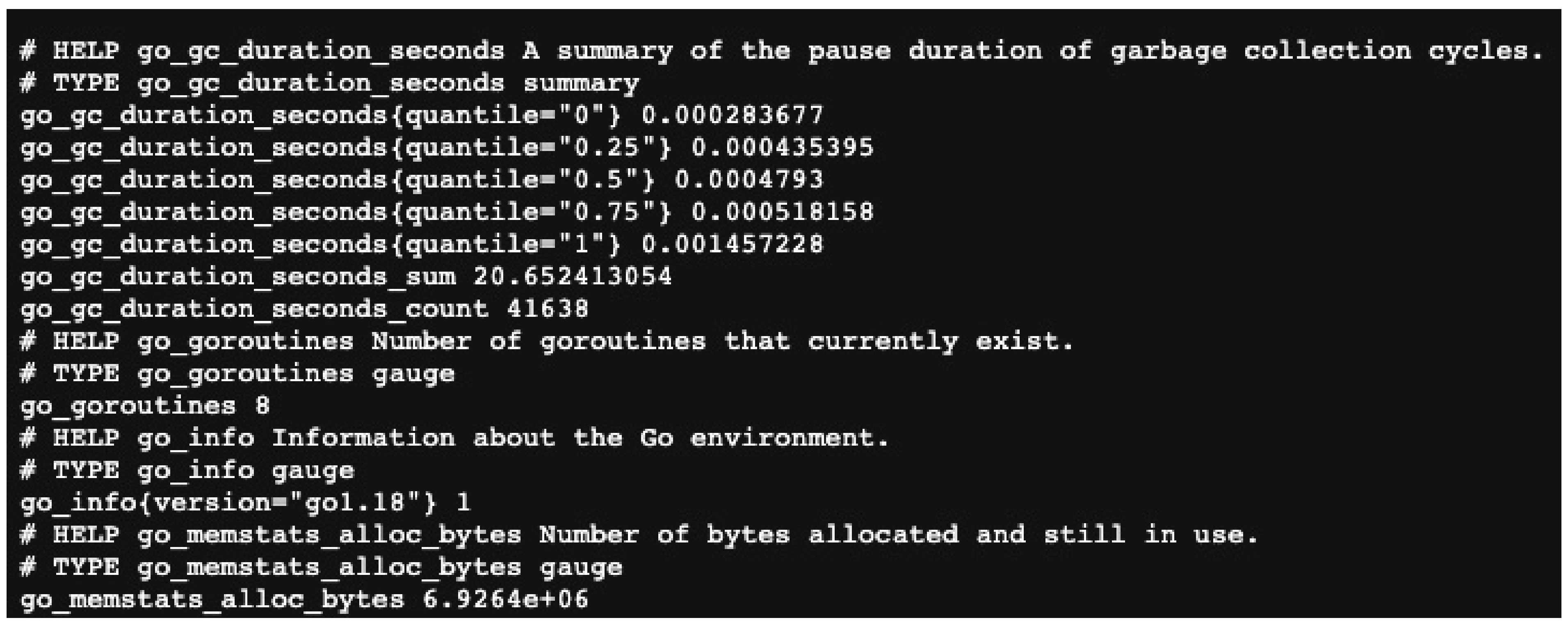
After a few seconds, the output of

```
# sockstat -l
```

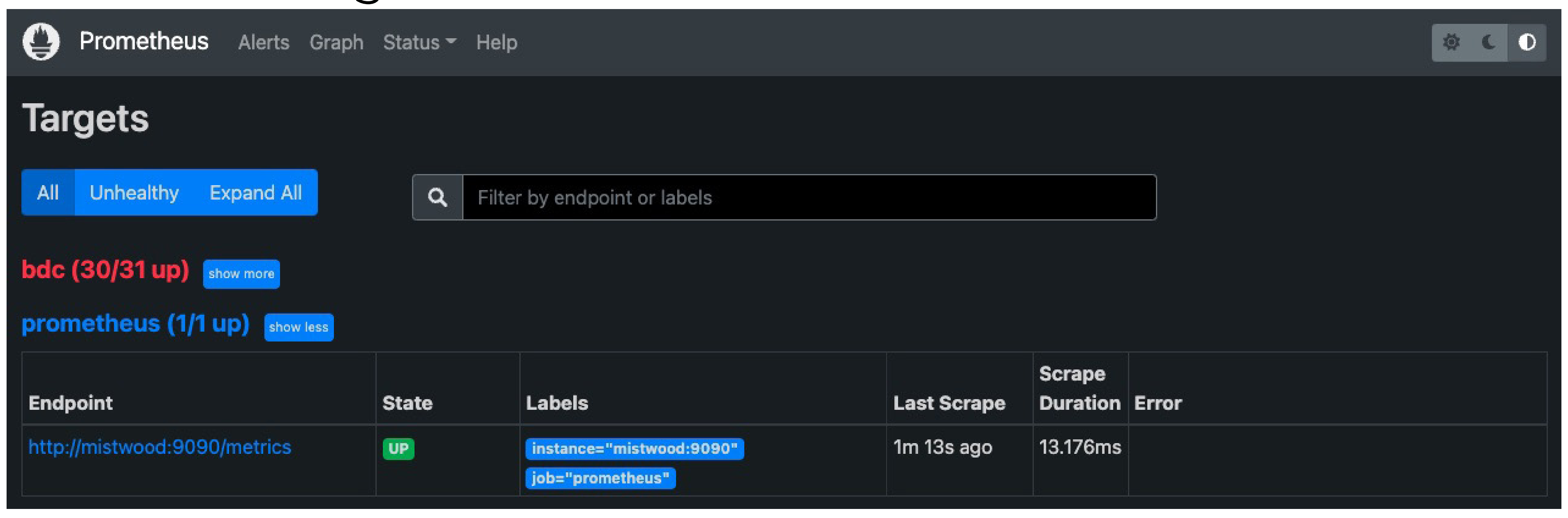
should have the following lines in it, confirming both services started successfully:

User Address	Command	PID	FD	PROTO	Local Address	Foreign
prometheus	prometheus	70027	8	tcp4	mistwood:9090	*:*
nobody	node_exporter	2950	3	tcp46	*:9100	*:*

First, let's see if the `node_exporter` is extracting some metrics. We can do that by pointing our browser to the URL of the host running the `node_exporter` service (mistwood in my case), adding the port 9100 and /metrics to the end to form this URL: http://mistwood:9100/metrics


```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0.000283677
go_gc_duration_seconds{quantile="0.25"} 0.000435395
go_gc_duration_seconds{quantile="0.5"} 0.0004793
go_gc_duration_seconds{quantile="0.75"} 0.000518158
go_gc_duration_seconds{quantile="1"} 0.001457228
go_gc_duration_seconds_sum 20.652413054
go_gc_duration_seconds_count 41638
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 8
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.18"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 6.9264e+06
```

You can see a list of exported metrics in a namespace separated by underscores. Refresh this page every 10 seconds and you'll see updated data collected by the `node_exporter`. Since we're already in the browser, we can also check the status of Prometheus. The URL is very similar but using the port 9090 (no `/metrics` at the end) to get to the Prometheus web interface. Go to the Status pulldown and select Targets to see all configured hosts from the `prometheus.yml` above. In this example setup, I have extracted pieces of configuration for our big data cluster (bdc). Of course, you can pick your own labels instead of "bdc" and the "mistwood" host is also exchangeable.



Targets

bdc (30/31 up)

prometheus (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://mistwood:9090/metrics	UP	instance="mistwood:9090" job="prometheus"	1m 13s ago	13.176ms	

Prometheus checks if the hosts are reachable. The `node_exporter` that we added to the `prometheus.yml` file with port 9100 is also listed here and can be reached from here by clicking on the URL as well. Prometheus also checks the availability of the host, indicated by the UP in the State column of the endpoint.

Tags can be assigned to a certain host group to logically group them together. All host metrics collected receive this tag and can be filtered later using the PromQL language or directly within Grafana. My job name here is called bdc and all the machines that belong to that group are listed under targets. (I abbreviated it here to have only one FreeBSD and one Linux host in it.)

Before we dive into visualizations with Grafana, we can also create simple graphs from the Prometheus web interface by going to the Graph tab. At the top, there is an input field. On the left the blue Execute button, there is smaller one called the metrics explorer. Click on it and a search field opens, containing all the names of the metrics collected so far. Pick the one that you want to see. After selecting the metric, click the Graph tab next to Table to see a visualization of the metric over all your hosts. Click and select a portion to zoom into that timeframe or use the controls above to zoom out. This is already good to get a quick overview, but may not be as visually appealing as a full-blown dashboard. We add Grafana to the mix as it has more capabilities and different ways to display the data in various forms.

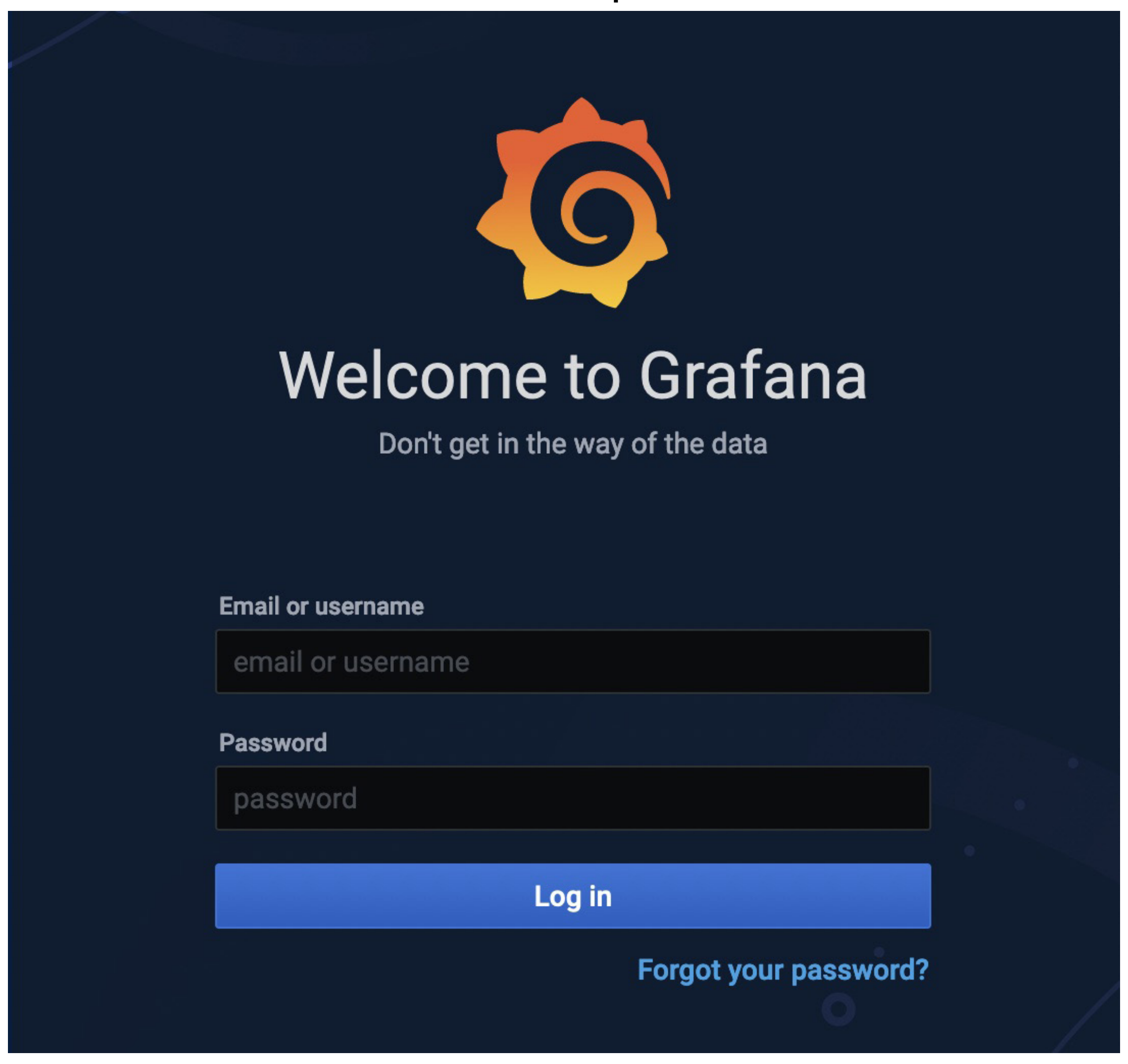
At the time of writing this article, Grafana 8 is the current version. Older versions work just as well, so there is no need to always chase the latest version to get pretty pictures.

```
# pkg install grafana8
```

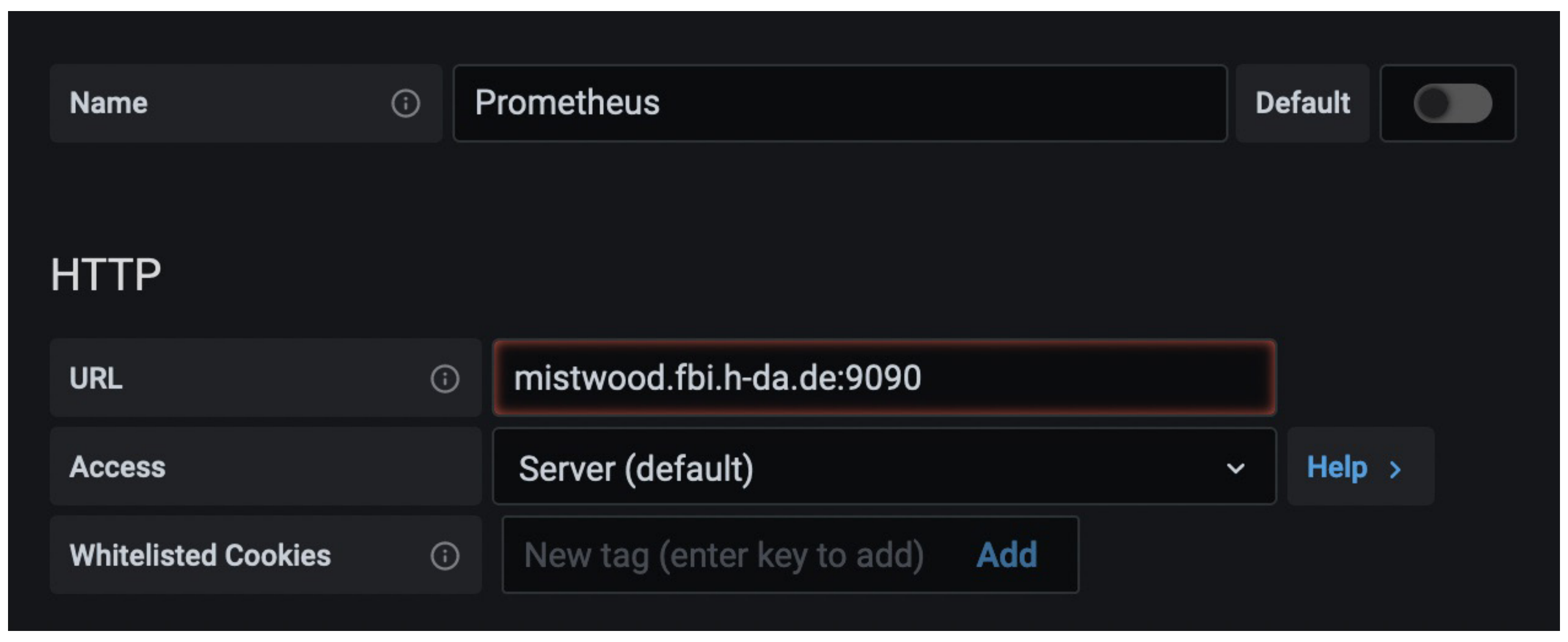
Like before, we activate the `grafana service` to run upon boot and start it right away with the following two lines:

```
# service grafana enable
# service grafana start
```

Grafana does have a configuration file under `/usr/local/etc`, but we do not need to modify it here. Make sure to visit and read the documentation on the Grafana homepage to change the file for your environment. Wait a little for Grafana to start (check the `sockstat` output for a Grafana line listening on port 3000 by default). Browse to the login page for Grafana on the host that you installed it on with port 3000.



On a fresh installation, Grafana has a default user **`admin`** with password **`admin`** that needs to be changed right after the first login to something else. You can also add more users with different privileges to see only certain dashboards, but right now we need to connect to our Prometheus metrics first. This is done by adding a data source under the gear icon on the left (Configuration -> Data Sources). Click the blue button on the left labeled "Add data source".



Name: Prometheus
Default

HTTP

URL: mistwood.fbi.h-da.de:9090
Access: Server (default)
Whitelisted Cookies: New tag (enter key to add)

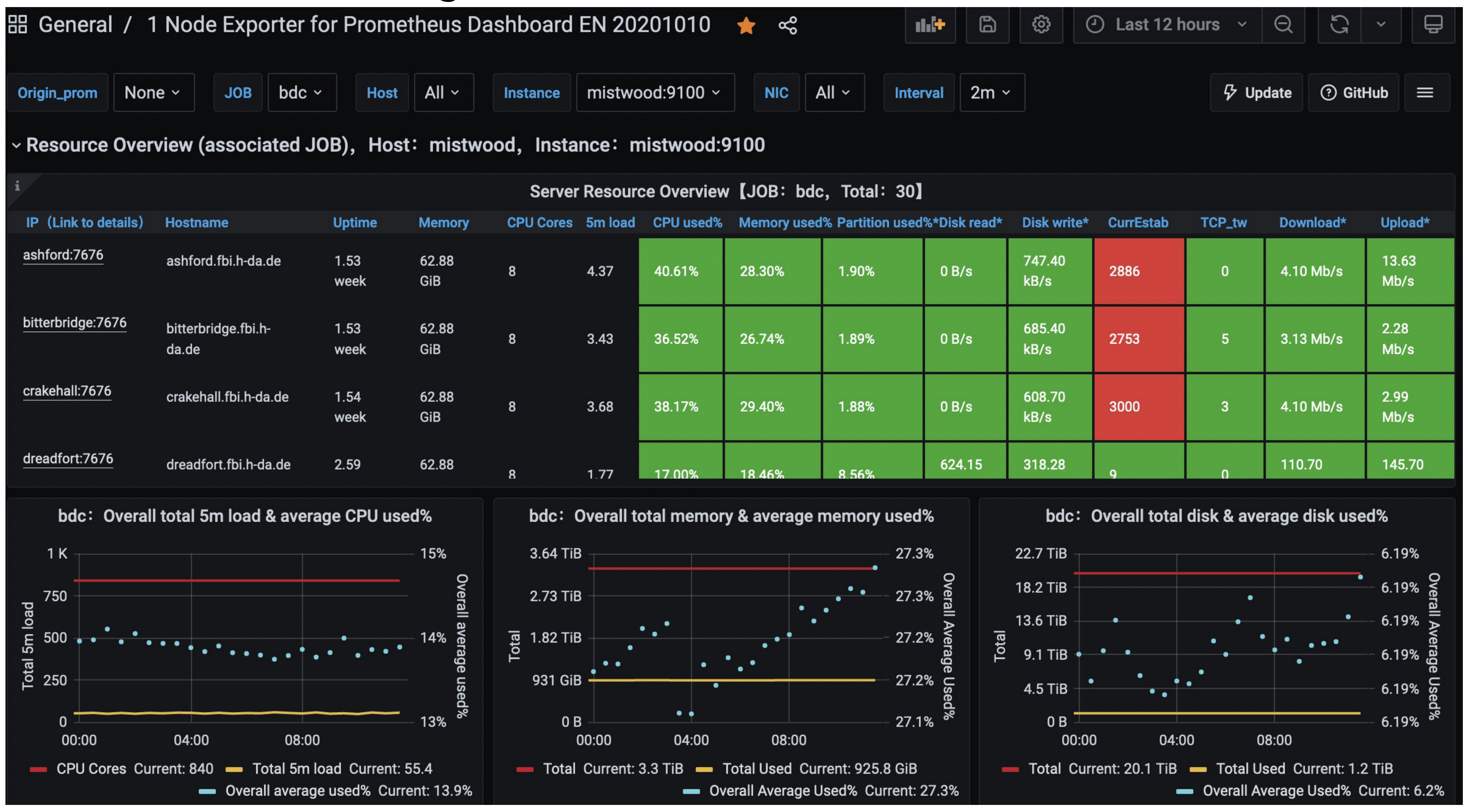
We give our datasource a descriptive name and provide the URL that we used earlier to access the Prometheus web UI on port 9090. At the bottom, click the "Save & Test" button to check if Grafana can reach your data source. Note: The configuration provided here is the most basic, which means it is focused on functionality and less on security. In production environments, you definitely need to have authentication and encryption of your metrics to not give attackers a clue about your infrastructure by reading the metrics. The Prometheus and Grafana webpages both provide documentation on how to do so.

Now that we have a datasource, we want to visualize the data coming from it. We can design our own dashboards, but my artistic talents go only so far. Other people have put in time and talent to create beautiful dashboards and provided them for everyone to use at the Grafana website. You can find them on https://grafana.com/grafana/dashboards/ along with filters on the left side to only show dashboards based on Prometheus data sources. Filtering further in the Collector Types pull-down to have Node exporter, the right side of the page automatically updates based on your filter criteria. Click on one of the search results to see a preview as well as additional information about it. On the right side, copy the dashboard ID to the Clipboard and change back to your Grafana browser tab.

Go to the dashboards tab on the left and select "Manage". On the left, there is a button labelled "Import". When clicked, you're brought to a screen that lets you paste the dashboard ID you selected earlier and load the dashboard. It's easy and convenient. Assign the data source created earlier and finish the import. For your convenience, here are a couple of dashboard IDs that I use (the last one is even built for FreeBSD use):

- 1860
- 11074
- 4260

You can find the dashboards listed on the Dashboards tab on the left and get to them by clicking their name. Some have filters at the top to pick a single host to display or select other criteria like network card or storage media.



General / 1 Node Exporter for Prometheus Dashboard EN 20201010

Origin_prom: None · JOB: bdc · Host: All · Instance: mistwood:9100 · NIC: All · Interval: 2m

Resource Overview (associated JOB), Host: mistwood, Instance: mistwood:9100

Server Resource Overview【JOB: bdc, Total: 30】

IP (Link to details)	Hostname	Uptime	Memory	CPU Cores	5m load	CPU used%	Memory used%	Partition used%*	Disk read*	Disk write*	CurrEstab	TCP_tw	Download*	Upload*
ashford:7676	ashford.fbi.h-da.de	1.53 week	62.88 GiB	8	4.37	40.61%	28.30%	1.90%	0 B/s	747.40 kB/s	2886	0	4.10 Mb/s	13.63 Mb/s
bitterbridge:7676	bitterbridge.fbi.h-da.de	1.53 week	62.88 GiB	8	3.43	36.52%	26.74%	1.89%	0 B/s	685.40 kB/s	2753	5	3.13 Mb/s	2.28 Mb/s
crakehall:7676	crakehall.fbi.h-da.de	1.54 week	62.88 GiB	8	3.68	38.17%	29.40%	1.88%	0 B/s	608.70 kB/s	3000	3	4.10 Mb/s	2.99 Mb/s
dreadfort:7676	dreadfort.fbi.h-da.de	2.59	62.88	8	1.77	17.00%	18.46%	8.56%	624.15	318.28	9	0	110.70	145.70

Some of the dashboards can be a bit overwhelming in the amount of data they display at once. I find that an overview dashboard showing me all machines is a good start to see what is going on. When I identify something out of the ordinary, I drill down into that host with an-other dashboard that shows me that machine in more detail. Especially the long term trends that Prometheus provides this way give me a good understanding of whether a certain spike in memory usage is expected and normal.

For each host that should be monitored, install and start the `node_exporter` on it. On the Prometheus host, add the URL to the targets under the `static_configs` in `prometheus.yml` and then restart the prometheus service. That's fairly straightforward and is easily automated for a large number of hosts using configuration management tools like Ansible and others. Try out other `node_exporters` available on FreeBSD and find a good dashboard (or create one yourself) that fits your monitoring needs. I find that Prometheus can show me a lot more metrics than my previous setup. Alerting about certain events is also possible and there are packages available on FreeBSD to do so. I'll leave that as a learning exercise for you.

Prometheus is straightforward to set up and extend with more hosts to monitor. It's time for you to steal a little bit of fire from the gods to get better insight into the dark depths of your hosts and services.

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. Benedict is one of the hosts of the weekly bsdnow.tv podcast.