

PSAwise2324Team10Aufgabe10

Network Monitoring

Our network monitor is on VM9.

We decided to use Prometheus in combination with Grafana for network monitoring. Prometheus is an open-source network monitoring tool, which we used for data collection. While Prometheus has a UI, it's somewhat limited, so we set up Grafana for the UI instead. Grafana also has inbuilt support for Prometheus, meaning the two work well together. Furthermore, it can handle user authentication, which we need since we want the network monitoring system to be remotely accessible.

Prometheus

Installation

Navigate to the directory /opt, where we decided to put our installation files, since it's the directory intended for storing external applications. Download Prometheus here by running `sudo curl -LO <download URL>` for both applications. You can find the URL on the Prometheus website (<https://prometheus.io/>) under "Download". After downloading, unpack the files using `sudo tar xvfz <directory name>`.

Prometheus needs some places to store its files, so create /var/lib/prometheus and /etc/prometheus.

```
sudo mkdir /etc/prometheus /var/lib/prometheus
```

Change into the prometheus directory you have unpacked in /opt and move "prometheus" as well as "promtool" to /usr/local/bin.

Additionally, move the prometheus configuration file prometheus.yml, as well as the consoles and console_libraries directories to /etc/prometheus.

In order for Prometheus to start automatically during start-up, set up a service for it. Create the file "prometheus.service" in /etc/systemd/system and give it these contents:

```
[Unit]
Description=Prometheus
Wants=network-online.target
After=network-online.target

[Service]
User=prometheus
Group=prometheus
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/bin/prometheus \
    --config.file /etc/prometheus/prometheus.yml \
    --storage.tsdb.path /var/lib/prometheus/ \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries \
    --web.listen-address=0.0.0.0:9090 \
    --web.enable-lifecycle \
    --log.level=info

[Install]
WantedBy=multi-user.target
sudo mkdir /etc/prometheus /var/lib/prometheus
```

This service should be run by a user called prometheus in the group prometheus, so you have to create both:

```
sudo useradd --user-group -rs /bin/false prometheus
```

(The option --user-group causes useradd to create not only a user but also a group of the same name and put the user in the group.)

Change the permission of all prometheus directories to match this user/group:

```
sudo chown -R prometheus:prometheus /etc/prometheus
/var/lib/prometheus /opt/<prometheus dir name; version dependant>
```

Once this is done, reload the systemctl daemon and start and enable the prometheus service:

```
sudo systemctl daemon-reload  
  
sudo systemctl start prometheus  
sudo systemctl enable prometheus
```

The endpoints that Prometheus monitors are referred to as "targets". In order to monitor the system, we need to create targets on each VM. The targets are provided by so called "exporters", which present Prometheus with certain metrics. Depending on the exporter, different metrics are collected. We used several different exporters for our network monitoring.

node_exporter

This exporter collects some basic data about the machine, like CPU usage and storage usage. `node_exporter` has to be installed on every VM. Furthermore, it has to be made accessible. It uses port 9100 by default, so add rules for it on every VM (input dport 9100, output sport 9100). On the VM that hosts prometheus (VM5 in our cases), rules for both dport and sport 9100 are required in the input and output chain.

The installation process of `node_exporter` is very similar to how Prometheus itself was installed.

On each VM, do the following:

Download and unpack `node_exporter` (URL can be found on same page as Prometheus download URL):

```
cd /opt  
sudo curl -LO <download URL>  
sudo tar xvfz <node exporter dir name>
```

Create a user and group for the service:

```
sudo useradd --user-group -rs /bin/false node_exporter
```

Change the owner/group of all `node_exporter` files to this user/group:

```
sudo chown node_exporter:node_exporter <node exporter dir name> -R
```

Move the executable `node_exporter` to `/usr/local/bin`.

```
sudo mv node_exporter /usr/local/bin
```

The remaining files in the `node_exporter` download directory are just NOTICE and LICENSE, which we aren't relevant going forward. You can delete the directory if you wish.

To have `node_exporter` start at start-up, set up a service for it. Go to `/etc/systemd/system` and create the file `node_exporter.service`:

```
[Unit]
Description=Node Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=node_exporter
Group=node_exporter
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/bin/node_exporter

[Install]
WantedBy=multi-user.target
```

And finally, reload the `systemctl` daemon, start and enable the service:

```
sudo systemctl daemon-reload

sudo systemctl start node_exporter
sudo systemctl enable node_exporter
```

blackbox_exporter

This exporter allows you to probe machines from the outside using protocols such as HTTP, ICMP and several others. `blackbox_exporter` only needs to be installed on the network manager VM. It uses port 9115 by default, so add rules permitting traffic across that port to your firewall.

The installation process is almost identical to the installation of `node_exporter`, so we won't explain it again in detail. Just use the same commands but replace the name of the exporter and the download URL.

The only significant difference is that `blackbox_exporter` has its own configuration file, `blackbox.yml`, so we suggest creating a directory `/etc/blackbox_exporter` for it and moving the file there. The file and directory should be owned by the new `blackbox_exporter` group and user, permission

755. Then the configuration file can be referenced easily when creating the blackbox service:

```
[Unit]
Description=Blackbox Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=blackbox_exporter
Group=blackbox_exporter
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/bin/blackbox_exporter \
    --config.file /etc/blackbox_exporter/blackbox.yml # <-- config
file

[Install]
WantedBy=multi-user.target
```

process_exporter

We used this exporter to count the running processes on each machine. To access it, add rules for port 9256 to the firewall. The installation is once again almost identical to that of node_exporter. The service configuration file is slightly different:

```
[Unit]
Description=Process Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=process_exporter
Group=process_exporter
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/bin/process-exporter \
    --web.listen-address=0.0.0.0:9256 \
    --config.path=/etc/process_exporter/process_exporter.yml

[Install]
WantedBy=multi-user.target
```

process_exporter it needs its own configuration file, process_exporter.yml, for

which we created the directory `/etc/process_exporter`. In our case, the only purpose of this file is to specify which processes are supposed to be monitored. This simple configuration file makes it monitor **all** processes:

```
process_names:
- name: "{{.Comm}}"
  cmdline:
- '.*'
```

Alerts

In order to set up alerts, we used Prometheus' Alertmanager. It uses port 9093 by default, so add firewall rules for it on the host machine of the network monitor. The installation process is nearly identical to the installation of `node_exporter`, so we won't be repeating it step by step.

The only differences are that there is a configuration file, `alertmanager.yml`, which needs to be moved to its own directory `/etc/prometheus_alertmanager` and the Alertmanager requires a data directory. For the data directory, create `/var/lib/prometheus_alertmanager`, set its owner/group to the newly created `prometheus_alertmanager` user and group and permissions to 755. You can then specify this storage location in the service configuration file using `--storage.path`, as seen below:

```
[Unit]
Description=Alertmanager
Wants=network-online.target
After=network-online.target

[Service]
Type=simple
User=prometheus_alertmanager
Group=prometheus_alertmanager
ExecReload=/bin/kill -HUP $MAINPID
ExecStart=/usr/local/bin/alertmanager \
  --config.file=/etc/prometheus_alertmanager/alertmanager.yml \
  --storage.path=/var/lib/prometheus_alertmanager

SyslogIdentifier=prometheus_alertmanager
Restart=always

[Install]
WantedBy=multi-user.target
```

Next, the Alertmanager has to be tied into Prometheus, to do this, add the following changes to prometheus.yml:

```
# <...>

rule_files:
  - "alertrules.yml"

# Alerting specifies settings related to the Alertmanager
alerting:
  alertmanagers:
    - static_configs:
      - targets:
          # Alertmanager's default port is 9093
          - localhost:9093

# <...>
```

And create the a rule file, titled alertrules.yml in /etc/prometheus. Give it these contents:

```
groups:

# alerts if service is down for > 1m
- name: state_alerts
  rules:
    - alert: InstanceDown
      # Condition for alerting
      expr: up == 0
      for: 1m
      # Annotation - additional informational labels to store more
      information
      annotations:
        title: 'Instance {{ $labels.instance }} down'
        description:
          '{{ $labels.instance }} of job {{ $labels.job }} has been down for m
          ore than 1 minute.'
      # Labels - additional labels to be attached to the alert
      labels:
        severity: 'critical'

# alerts if storage space on file server is running low
- name: storage_low
  rules:
    - alert: DiskWillFillIn24Hours
      expr:
        predict_linear(node_filesystem_avail_bytes{job="router_db_dhcp_files
        erver",mountpoint="/mnt/md0",fstype!="rootfs"}[1h], 24 * 3600) < 0
      for: 5m
```

```
labels:  
  severity: 'high'
```

These rules specify alerts for

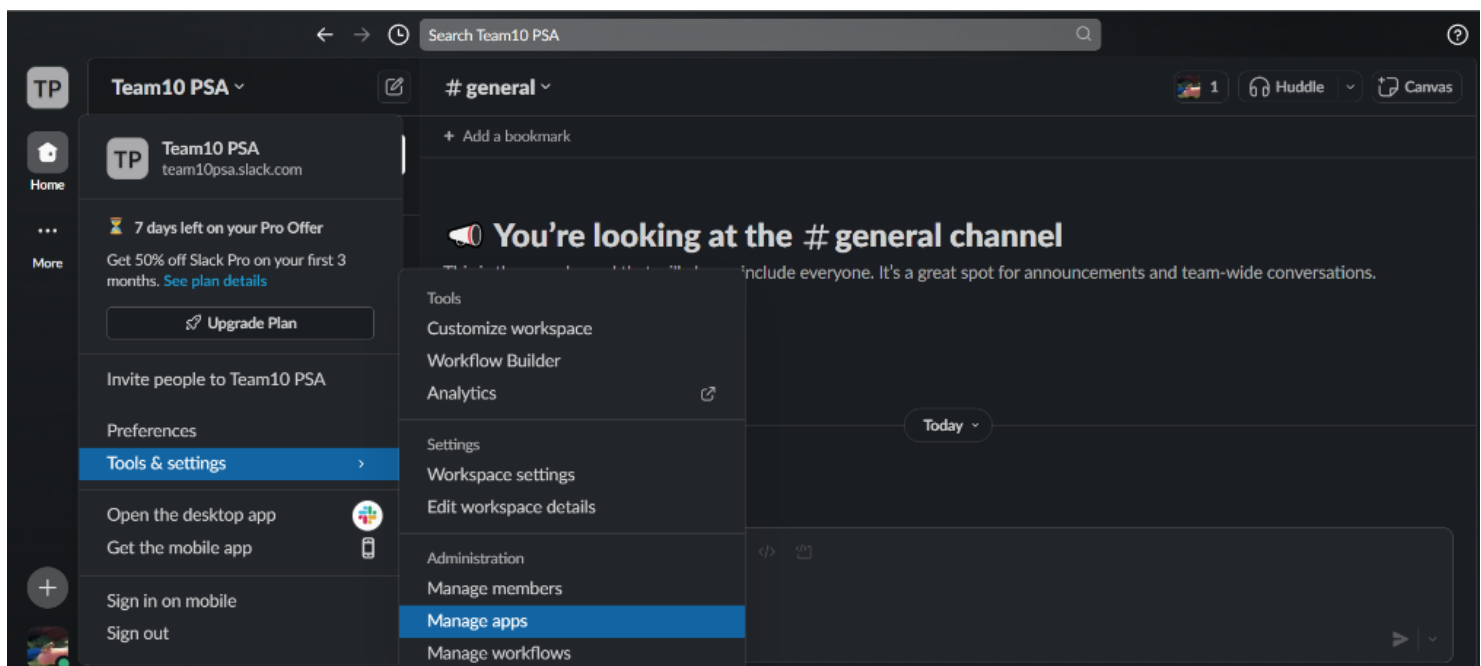
- services being down
- storage space running out (relative to the speed that the space is currently filling up! alerts 24 hours before predicted issue)

Run `sudo systemctl daemon-reload` and restart prometheus as well as prometheus_alertmanager after this. Now, Alertmanager should be active.

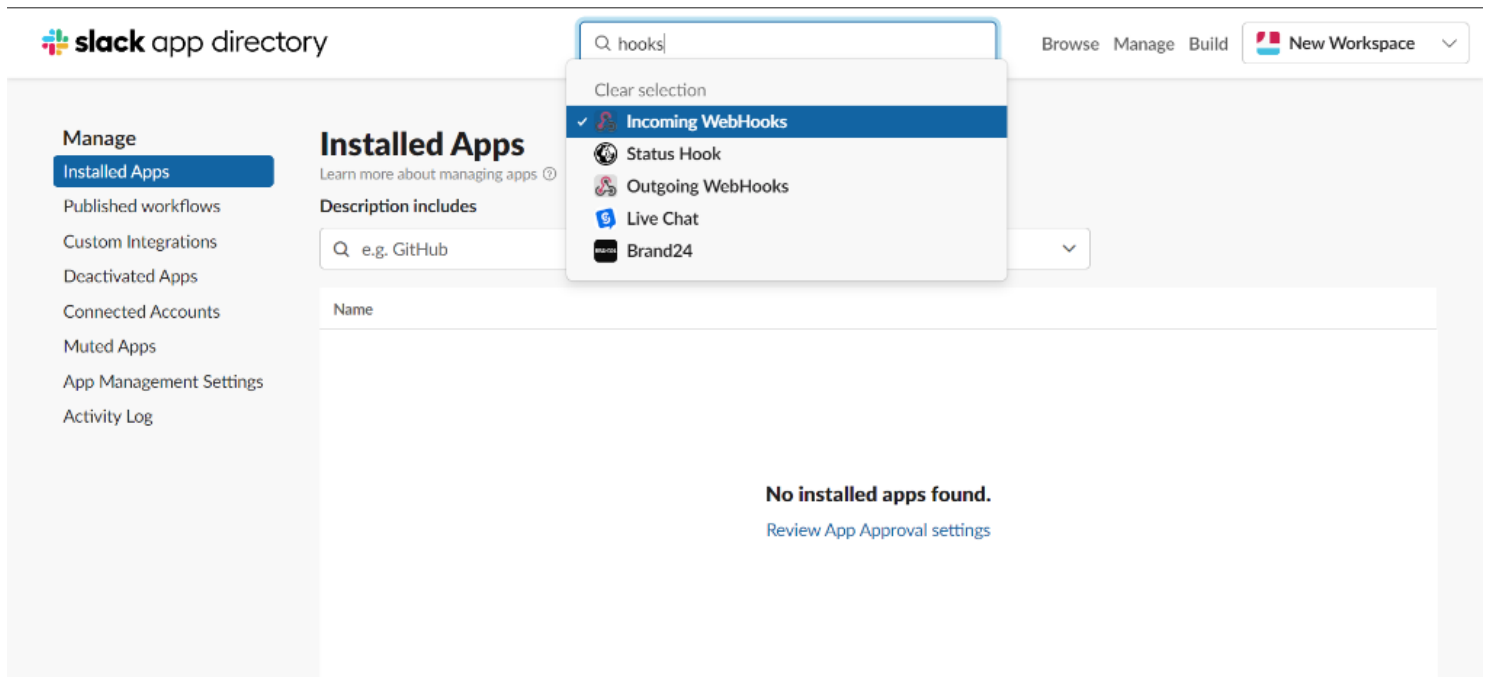
Alert Notifications through Slack

Since Slack is often used in professional settings, we thought it made sense to use it as the alternative alert notification system. First, create a workspace in Slack, as well as a channel that will be used for notifications. We named ours `#network-monitoring`.

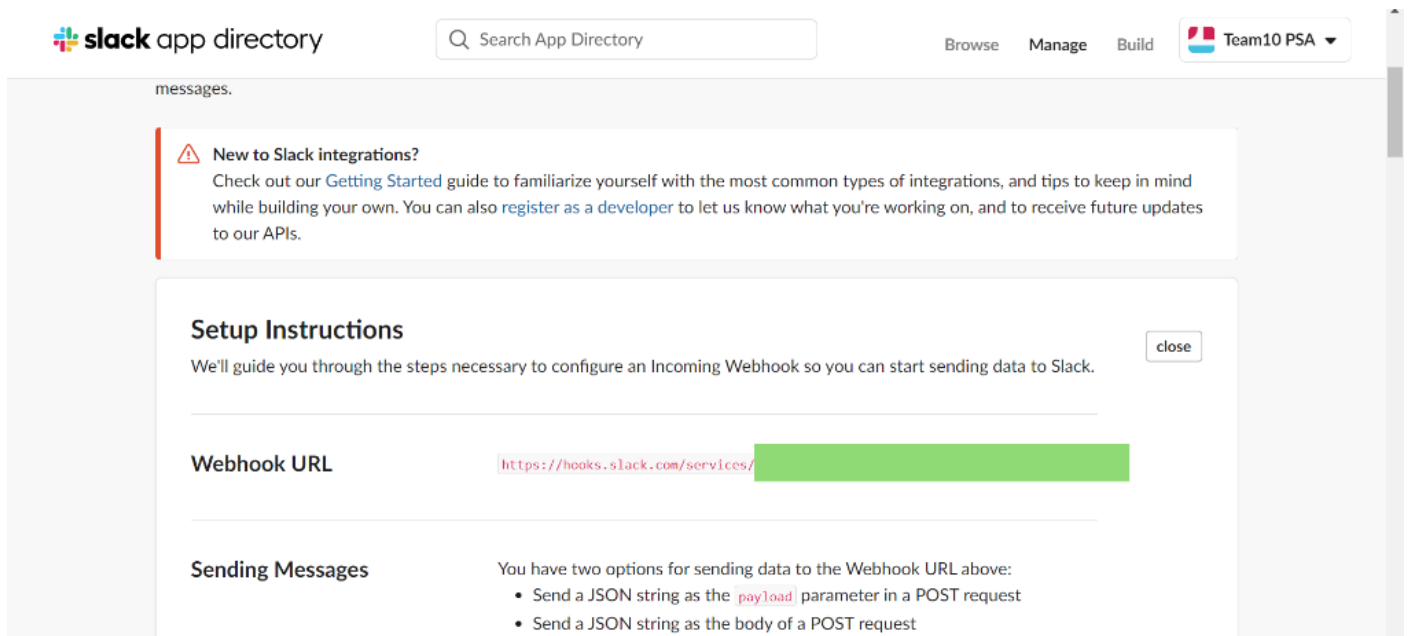
Then, click on the workspace name and navigate to Tools & Settings > Manage Apps:



Search for "Incoming WebHooks" and add this app.



Input #network-monitoring in the field where it asks what channel hooks should be added to. The app will create the hooks and display a link which can be used to tell Alertmanager where to send its notifications.



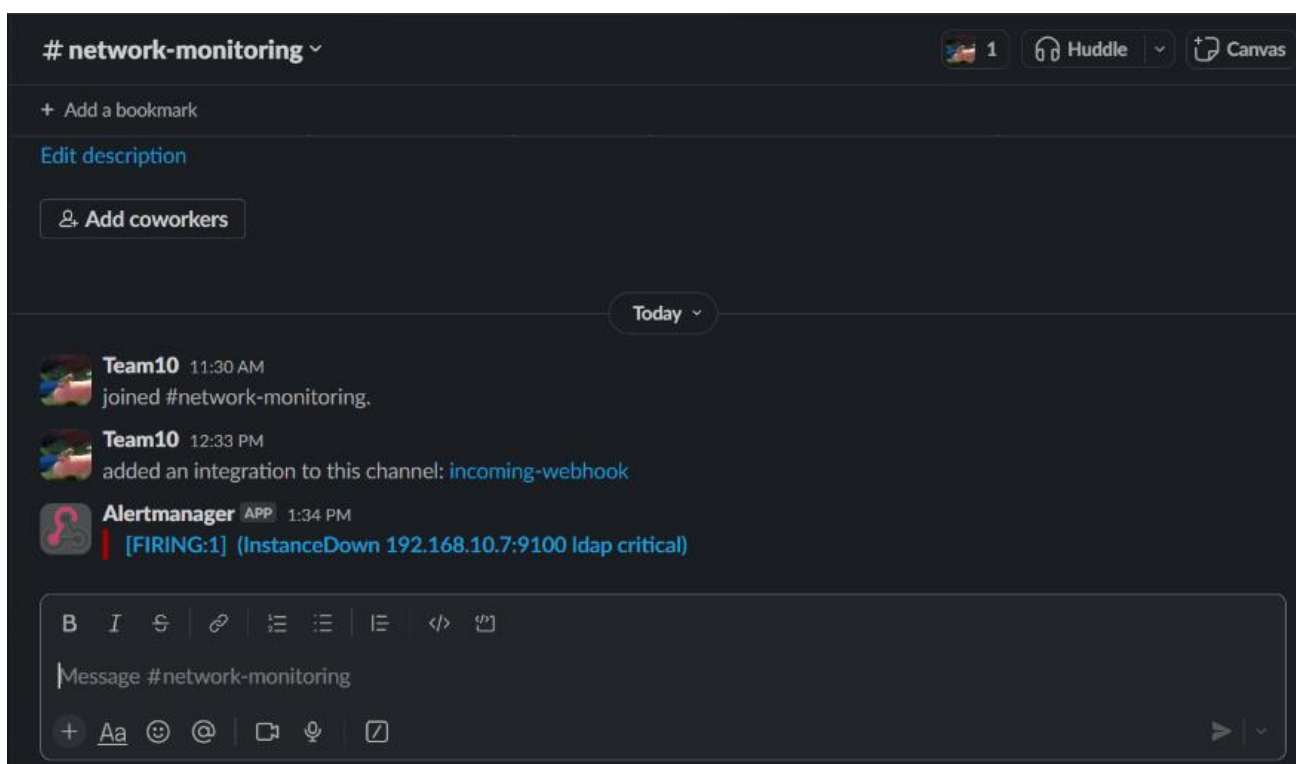
Copy the link and replace the contents of alertmanager.yml with this. Make sure to add the link in the specified position.

```

global:
  resolve_timeout: 1m
  slack_api_url: '<LINK YOU COPIED>'
route:
  receiver: 'slack-notifications'
receivers:
- name: 'slack-notifications'
  slack_configs:
  - channel: '#network-monitoring'
    send_resolved: true

```

Run `sudo systemctl daemon-reload` and restart prometheus as well as prometheus_alertmanager. Now alerts should be sent to your Slack channel:



Grafana

Installation

Next, you need to install Grafana.

First of all, install a few necessary dependencies:

```

sudo apt install -y apt-transport-https software-properties-common
wget

```

Then import the CPG key:

```
sudo mkdir -p /etc/apt/keyrings/  
  
wget -q -O - https://apt.grafana.com/gpg.key | gpg --dearmor | sudo  
tee /etc/apt/keyrings/grafana.gpg > /dev/null
```

Add stable a release and a beta release repository:

```
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg]  
https://apt.grafana.com stable main" | sudo tee -a  
/etc/apt/sources.list.d/grafana.list  
  
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg]  
https://apt.grafana.com beta main" | sudo tee -a  
/etc/apt/sources.list.d/grafana.list
```

And finally, you can actually install Grafana using apt:

```
sudo apt update  
sudo apt install grafana
```

If everything worked correctly, Grafana is now installed. Start it and enable it so that it get's started at start-up:

```
sudo systemctl start grafana-server  
sudo systemctl enable grafana-server
```

Since Grafana provides our UI, this is the service we want to make accessible across the internet. To do this, go to the settings of the current VM (in our case VM5) > Network > NAT Adapter > Port Forwarding and set up port forwarding like this (Grafana uses port 3000 by default):

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
grafana	TCP	0.0.0.0	61056		3000

You also need to update the firewall with rules for input dport 3000 and output sport 3000.

With this, Grafana should be reachable through the URL `http://131.159.74.56:61056/`. However, we want our communications to be encrypted, so we also need to set up an https connection.

SSL

We decided to use a self-signed certificate for encryption. All SSL files are in the `/etc/grafana` directory.

First of all, generate the private key:

```
sudo openssl genrsa -out /etc/grafana/grafana.key 2048
```

Then a certificate signing request:

```
sudo openssl req -new -key /etc/grafana/grafana.key -out  
/etc/grafana/grafana.csr
```

After running this command, you will be prompted for some basic information. Leave the challenge password empty.

Next, create the self-signed certificate by running:

```
sudo openssl x509 -req -days 365 -in /etc/grafana/grafana.csr -  
signkey /etc/grafana/grafana.key -out /etc/grafana/grafana.crt
```

And finally, set all the permissions correctly:

```
sudo chown grafana:grafana /etc/grafana/grafana.crt  
sudo chown grafana:grafana /etc/grafana/grafana.key  
sudo chmod 400 /etc/grafana/grafana.key /etc/grafana/grafana.crt
```

With all that in place, configure Grafana to use https with the newly created certificate by adding these settings to `/etc/grafana/grafana.ini`:

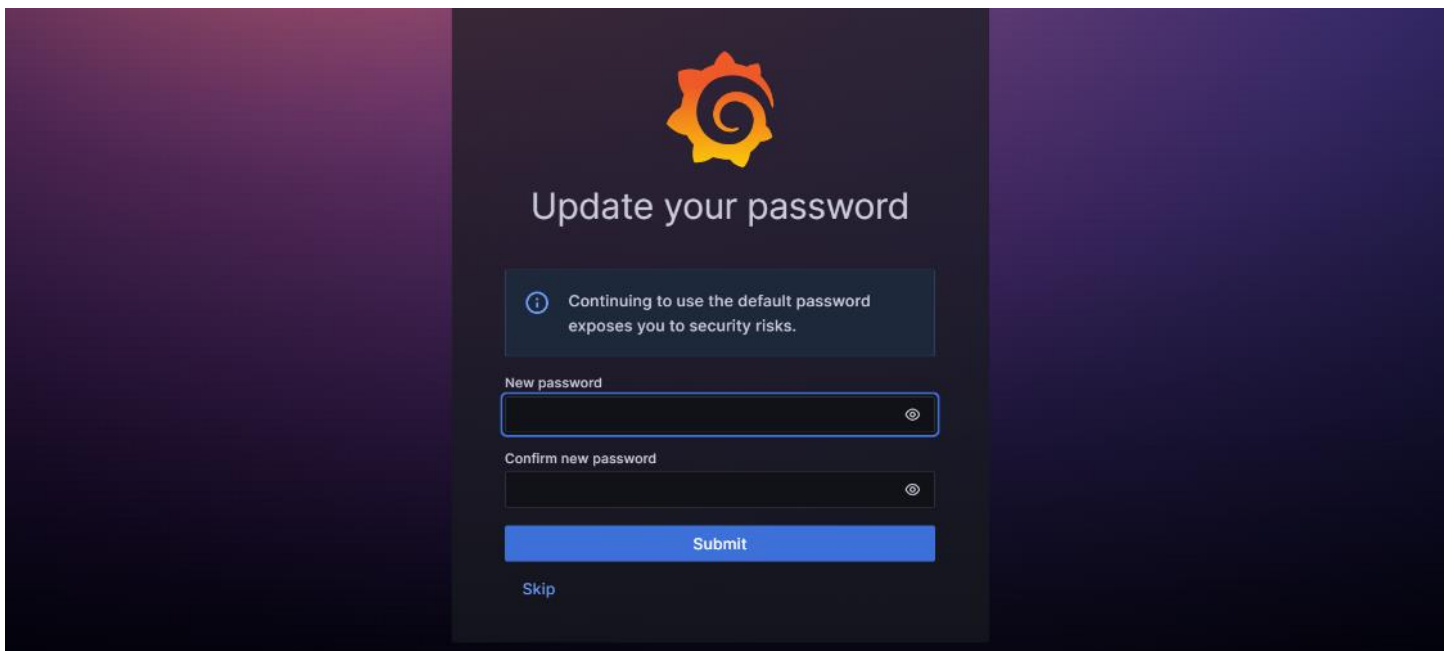
```
protocol = https  
cert_file = /etc/grafana/grafana.crt  
cert_key = /etc/grafana/grafana.key
```

Authentication

We don't want anyone but ourselves to have access to our network monitoring, so we only needed one account, with admin privileges. Grafana comes with one such account already created. The credentials are:

```
username: admin  
password: admin
```

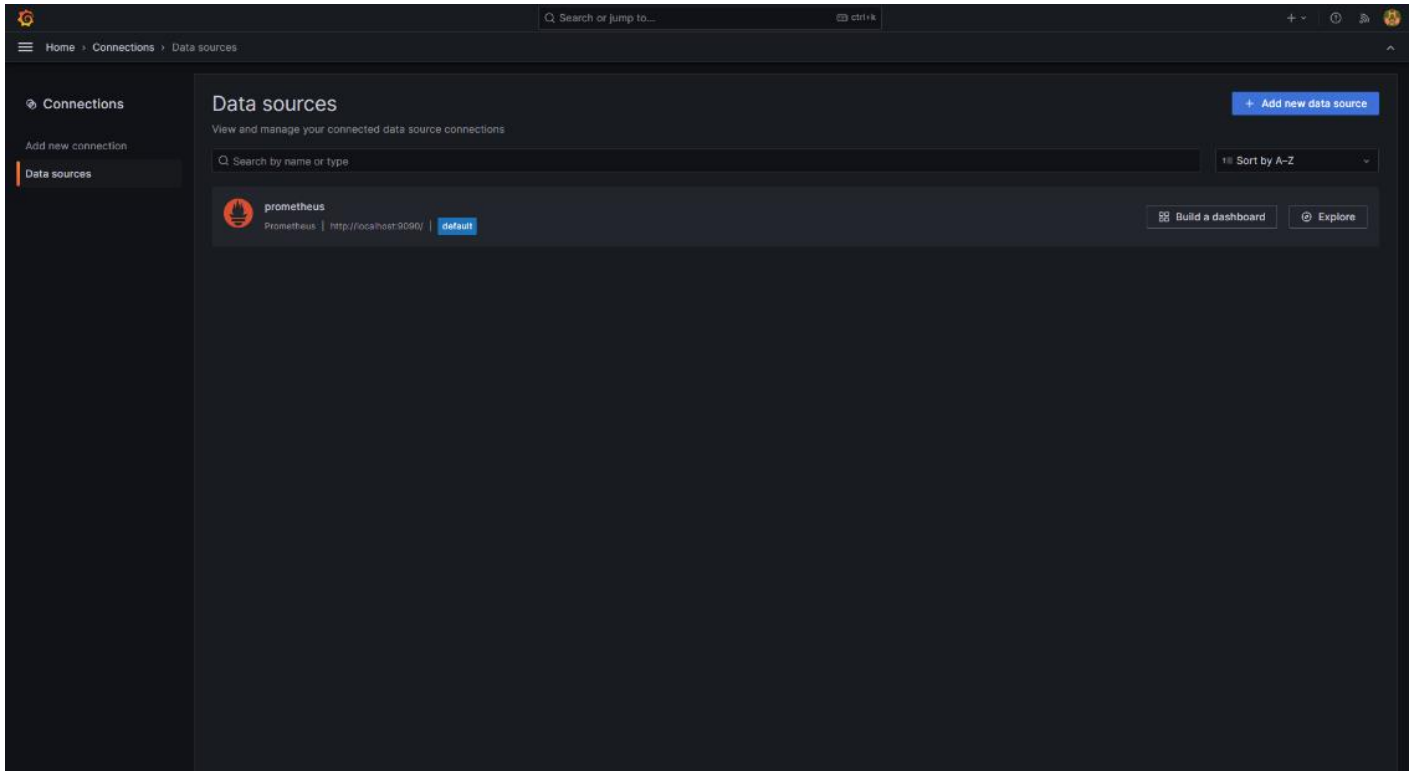
Simply enter these in the login page of your Grafana website. Upon logging in, you can select a new, more secure password.



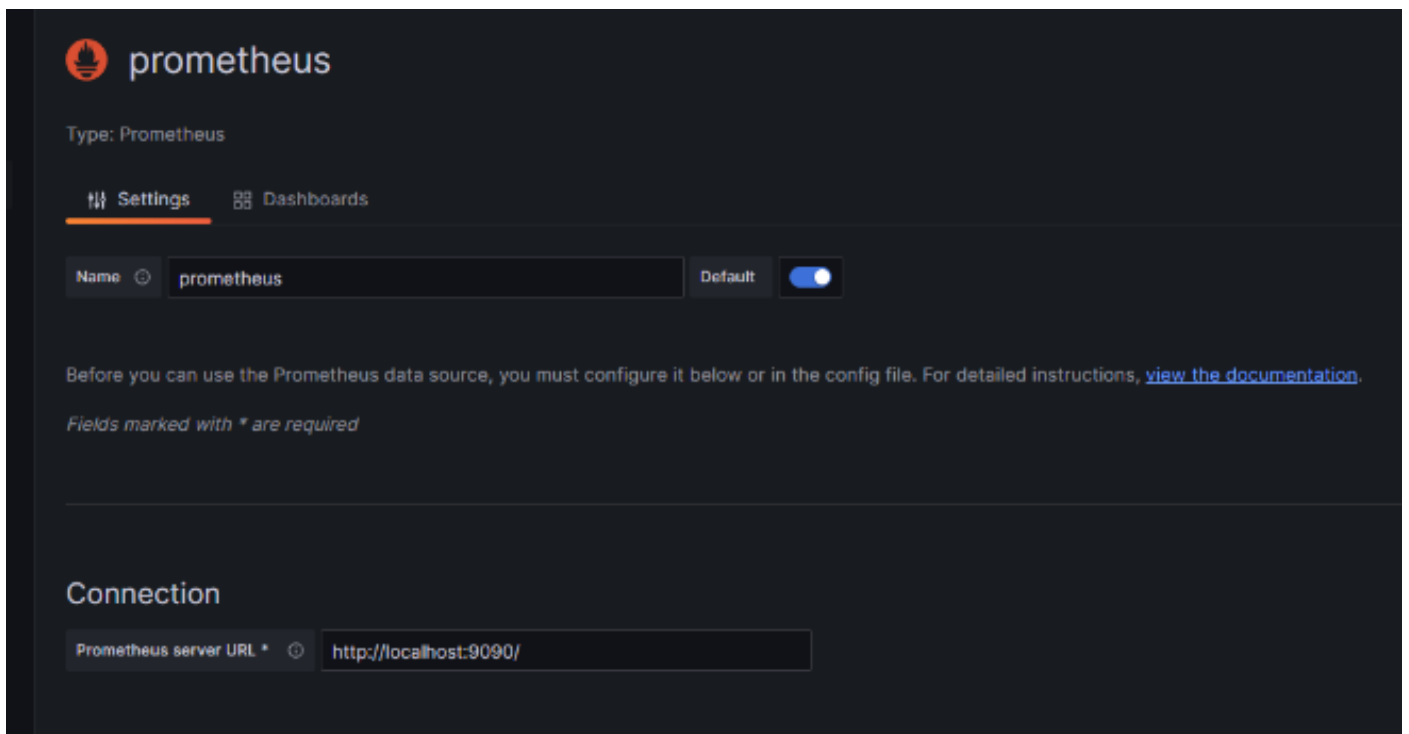
Connect to Prometheus

Open Grafana in a web browser and log in with the admin account. Then, open the menu in the sidebar and navigate to "Connections > Data Sources."

Click on "+ Add new data source" in the top right corner and select "prometheus",

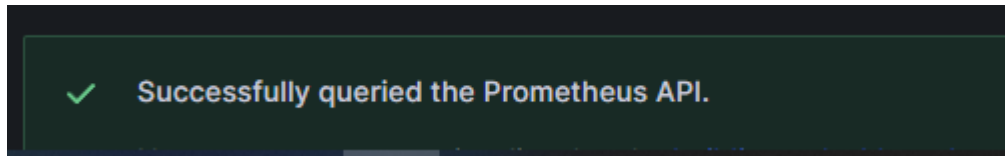


Now, you can enter the address and name of the data source as follows:

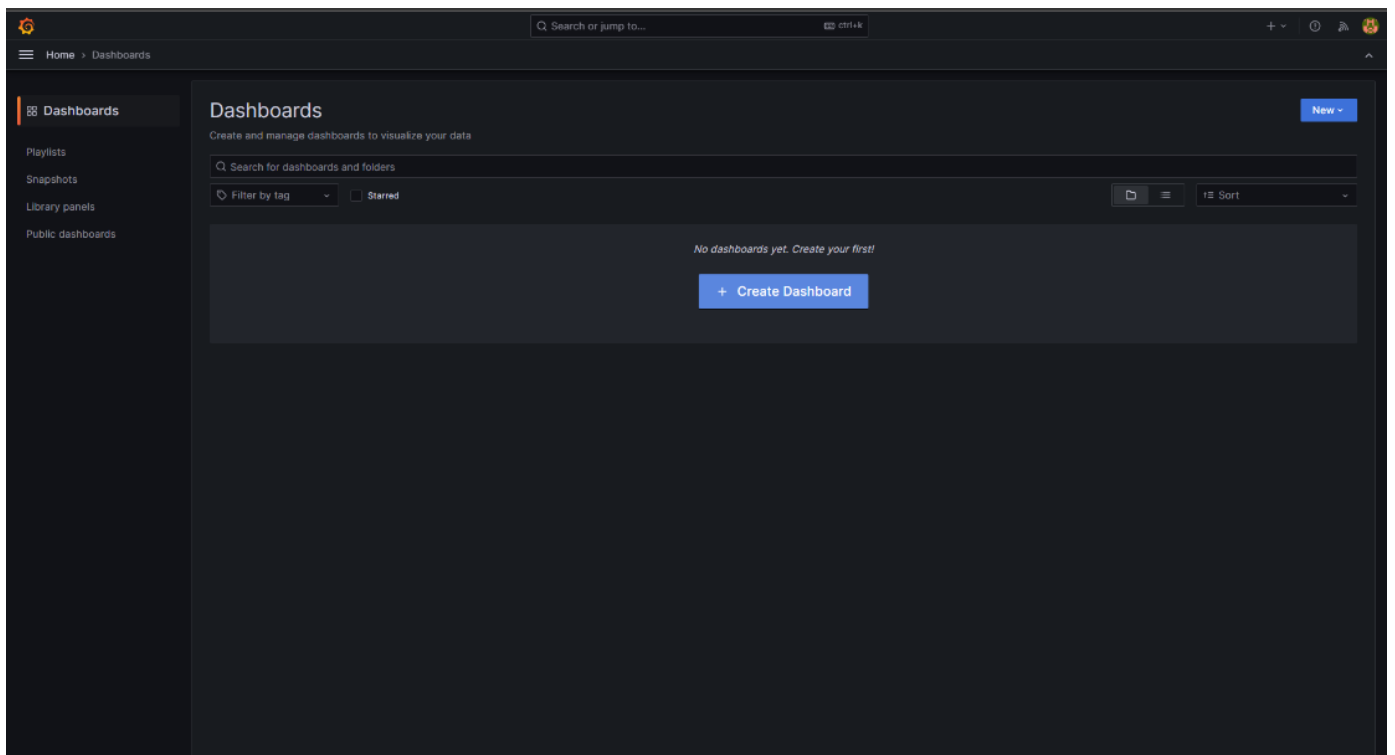


We also set the default query editor to "Code" for convenience. This is the editor you will see when you write prompts to access data from this data sources.

Do this and then click "Save & test" at the bottom of the page. The following message should appear:



Grafana can now query data from the Prometheus service. To visualize the data, you need to create a dashboard. Navigate to "Dashboard" and click "+ Create Dashboard".

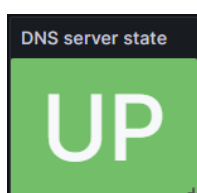


Click on the save icon in the top right to save the dashboard and give it a name.

Now, you can start adding everything you want to visualize to the dashboard. Click "+ Add visualization" and select Prometheus as the data source

Here is a list of all the panel types we used:

State



These panels are for monitoring whether the targeted services are up and running. For this very simple panel, you need to add a job like this to prometheus.yml and restart the prometheus service:

```
scrape_configs:
# <...>

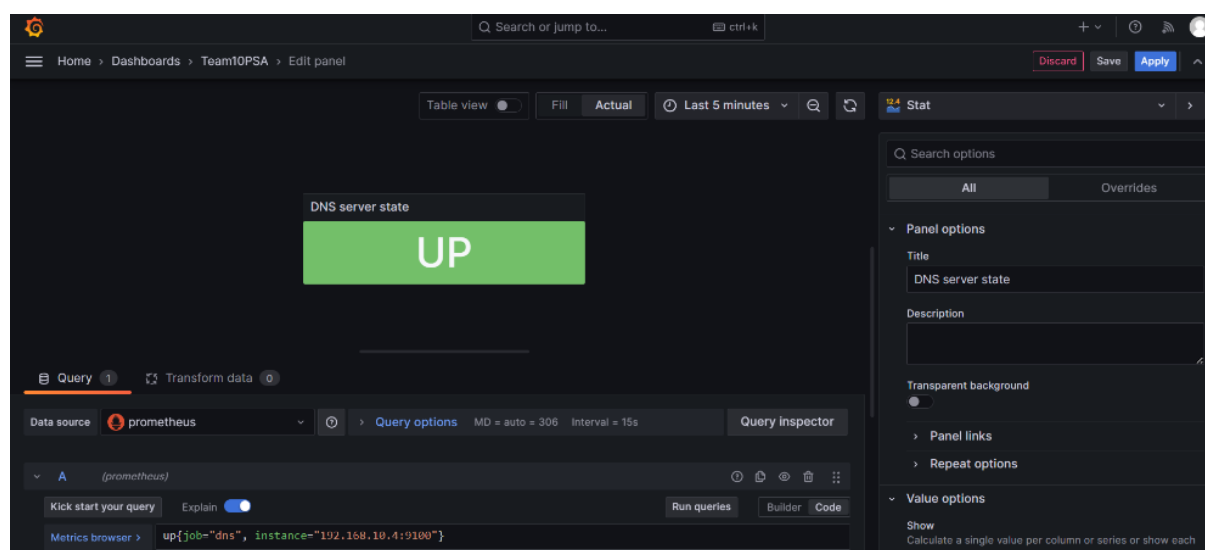
- job_name: "dns"
  scrape_interval: 10s # how long to wait between scrapes

static_configs:
  - targets: ["192.168.10.4:9100"] # where the endpoint is
```

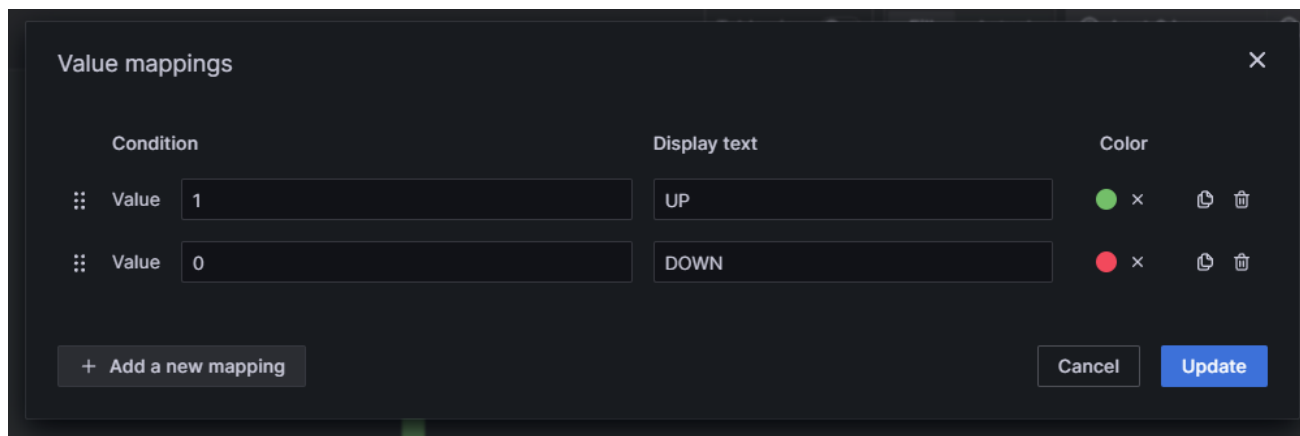
This is a node_exporter endpoint, so node_exporter needs to be installed on the target VM (see installation guide further up in this document) and be listening to port 9100.

Then you can query the state using the PromQL query "up", which returns 0 for DOWN and 1 for UP. This query is typed into the query input field at the bottom in Grafana. To find the service, we identify it through it's "job" title, which is part of the prometheus configuration (see Prometheus section). Then, the return value of the prompt execution will be displayed in the panel.

As you can see in the top right, we chose the panel type "stat" to display this simple value. We also gave the panel a title, "DNS server state", which is displayed above the actual context, to differentiate it from the other target states. In order to make the background of the panel one solid color, we selected "Background solid" and set graph mode to "None" in the "Stat styles" section.



In order for the panel to display not just 1 and 0, but UP and DOWN instead, we also added a value mapping in the menu on the right. We configured it as follows:



Condition	Display text	Color
Value 1	UP	Green
Value 0	DOWN	Red

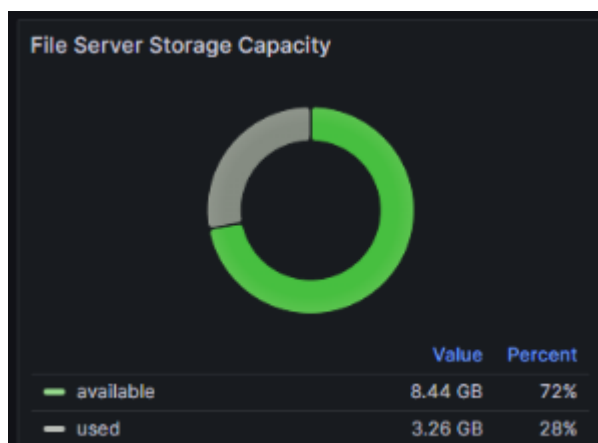
+ Add a new mapping

Cancel Update

Once you're done configuring a panel, click "Apply" in the upper right corner and save the dashboard to make the change permanent.

We created one of these panels for each of our VMs.

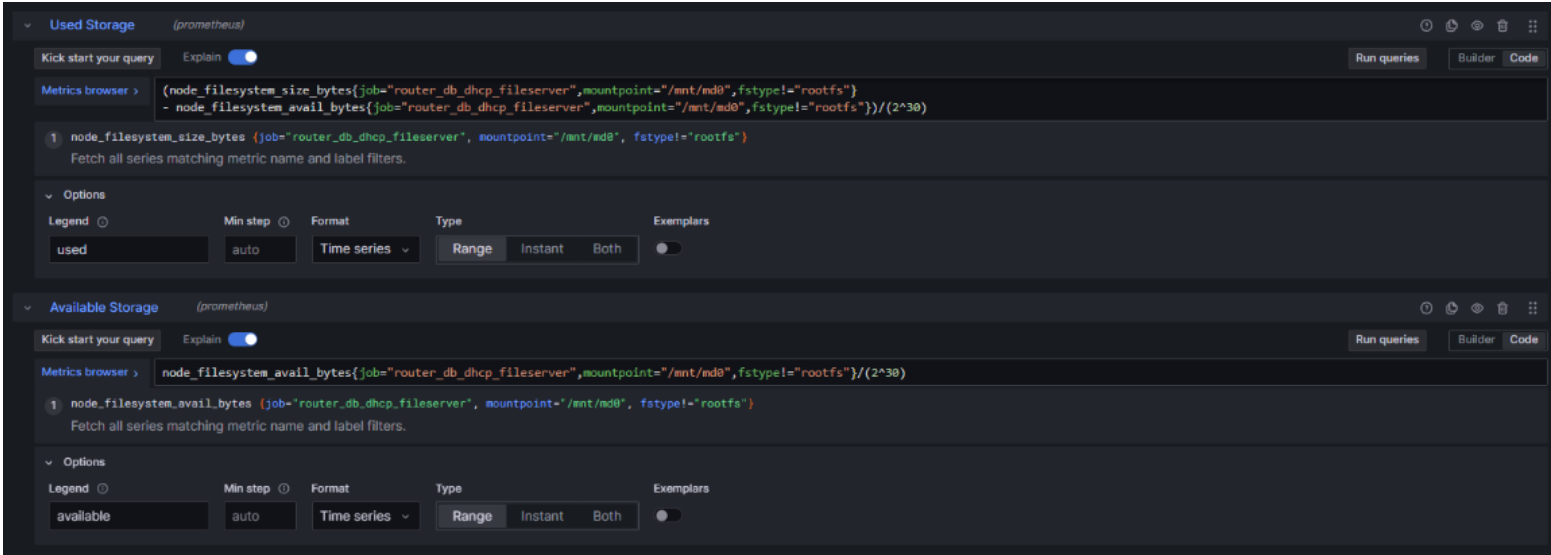
Disk Storage Monitor Individual



We want to know the remaining storage capacity of our file server. For this, a "pie chart" type visualization is useful.

To collect the necessary metrics, a `node_exporter` endpoint needs to be present on the fileserver. (For configuration of endpoint, see the section about the *State* panel.)

Create a new visualization in the dashboard and set the type to pie chart. We want to know how much storage is available and how much is already in use, so we need two queries:

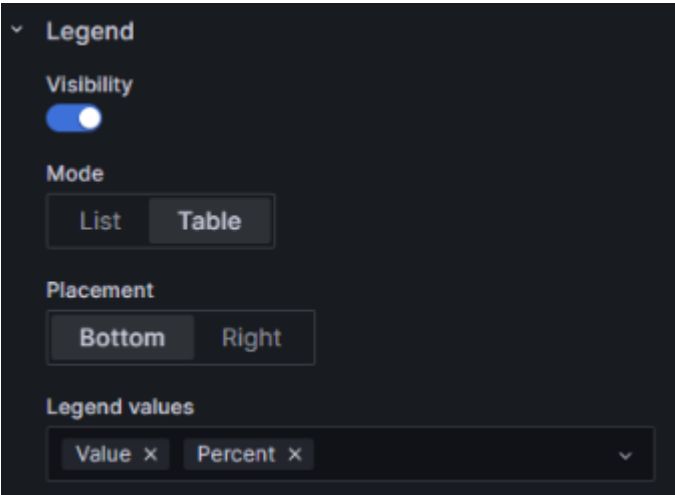


The first query calculates the used storage by subtracting the number of available bytes from the total amount. The result is also scaled by 2^{30} to transform it to GB. As you can see, the job and mountpoint need to be specified to target the storage space that actually belongs to the fileserver. In our case, this is mountpoint /mnt/md0 on VM1.

The second query queries the number of available bytes and, again, scales them to GB.

Below the queries, in the options section, you can enter a custom name for the query result which will be displayed in the legend of the pie chart. To change the color of the pie chart sections, simply click on the color in the legend and pick a new color.

In the sidebar, we also changed the pie chart type to donut, selected a unit in the "Standard Options" section (gigabytes) and modified the legend to display the return value of the queries, as well as the percentage they take up in the pie chart.



Disk Storage Monitor Overview

Disk Usage All VMs					
Machine	/home	/	/boot/efi	/mnt/md0	/var/lib/postgresql
VM1	26.1%	20.0%	1.47%	26.1%	26.1%
VM4	26.1%	29.5%	1.28%		
VM7		43.2%	1.47%		
VM5	26.1%	56.4%	1.47%		
VM9	26.1%	52.6%	1.47%		

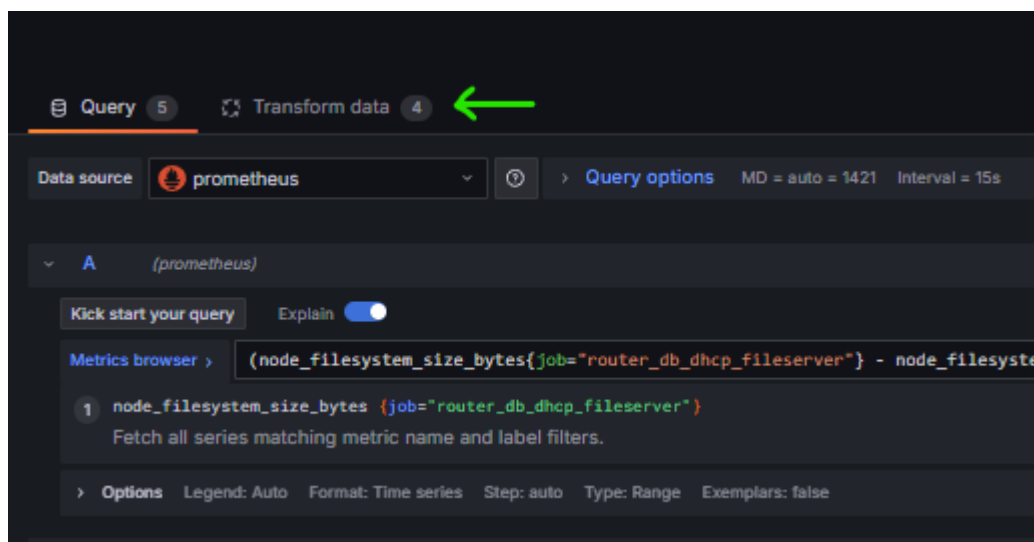
Similar to the panel described above, this one also uses node_exporter to collect its metrics. But its purpose is to give an overview of the storage usage of all VMs.

The query used in Grafana is as follows:

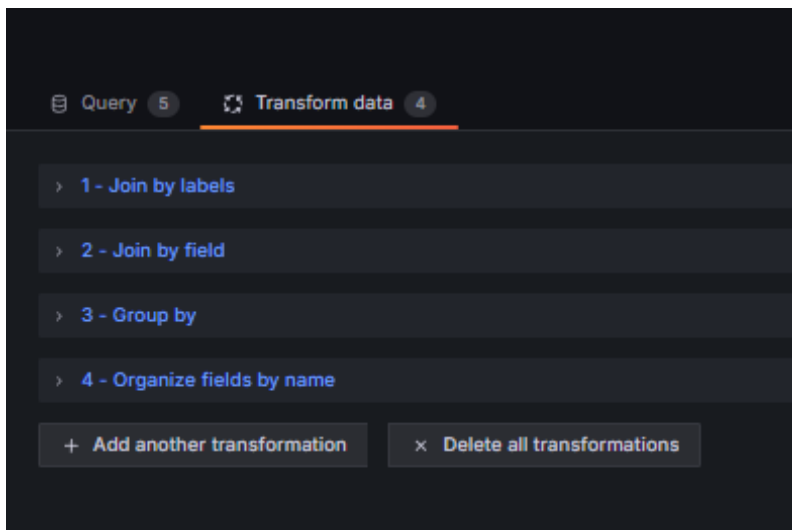
```
Kick start your query Explain ☒  
Metrics browser > (node_filesystem_size_bytes{job="router_db_dhcp_fileserver"} - node_filesystem_avail_bytes{job="router_db_dhcp_fileserver"})/(node_filesystem_size_bytes{job="router_db_dhcp_fileserver"}/100)
```

One query is needed per VM.

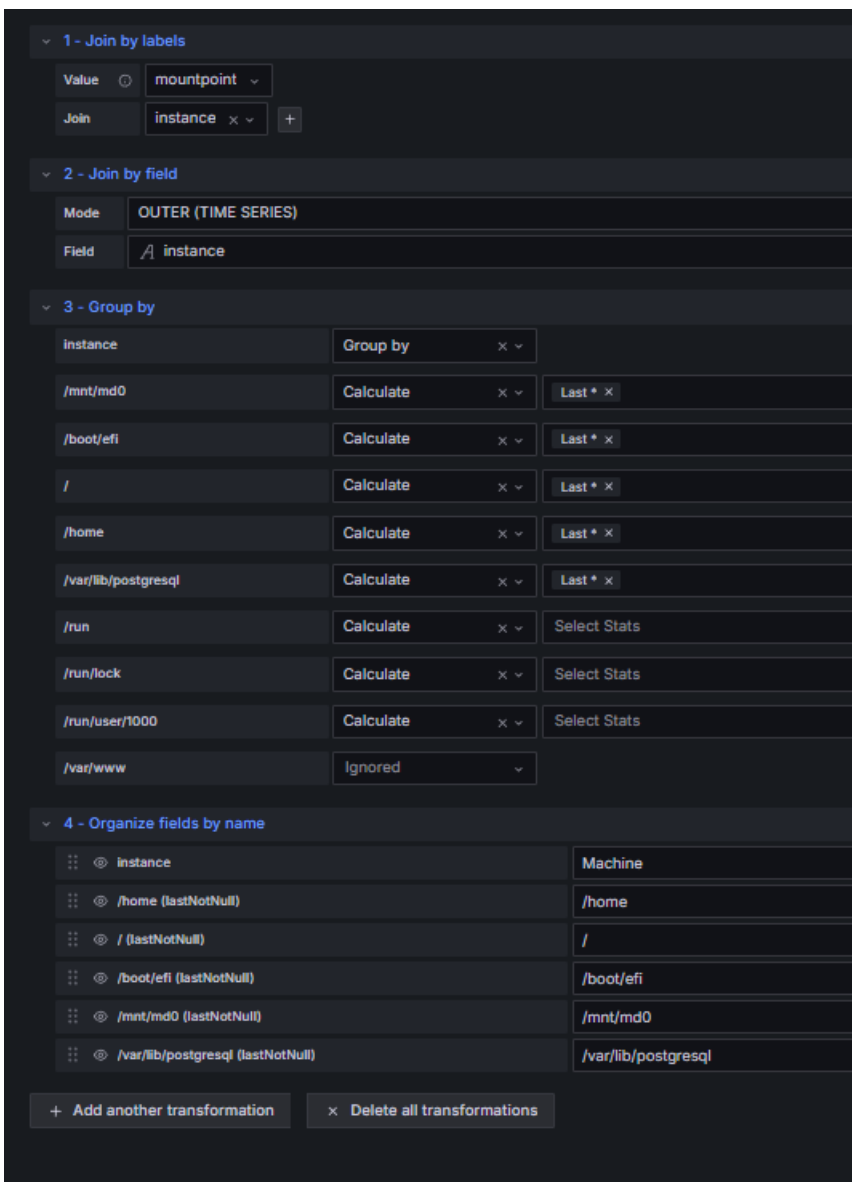
In order for the data to be displayed in a organized table, you need to switch the visualization type to "table" and navigate to the "Transform Data" tab:



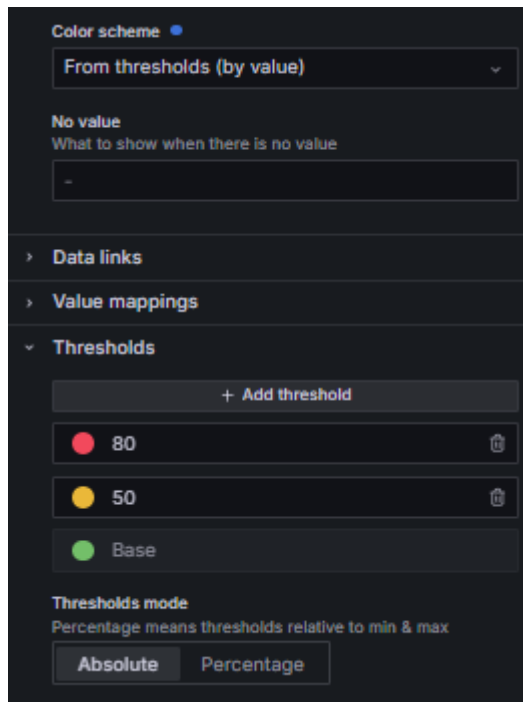
Add these four query types, in this order



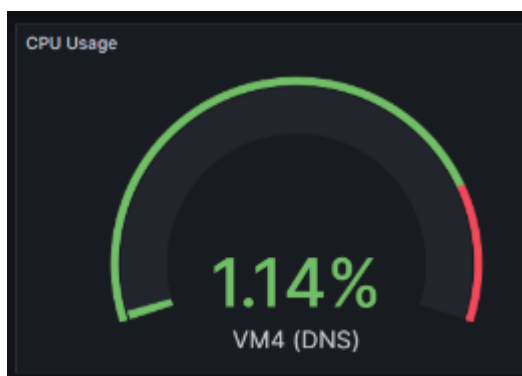
and configure them as follows:



To get the value fields to be displayed in percentages, select "Percent" as the value unit. Lastly, to get the percentages to change color depending on how full the part in question is, configure threshold colors and select "color scheme: from thresholds" in the standard options section:



CPU Usage



This metric also requires a `node_exporter` endpoint. (For configuration, see the section about the *State* panel.)

For visualization, we suggest using a gauge type panel. To calculate the CPU usage, we queried the percentage of the CPU that is currently in idle mode and subtracted it from 100%. The query looks as follows:



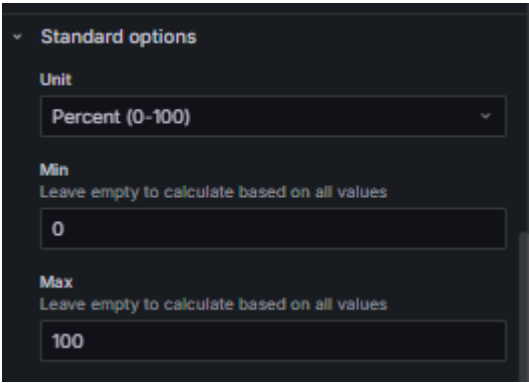
The rate function calculates a rate given the beginning and the end of the interval in brackets, in this case one minute. This prevents momentary spikes from being displayed. As usual, you have to indicate the job name to get the metrics for that specific target.

You can also change the name of the gauge in the Options > Legend section below the query.

CPU Usage Graph

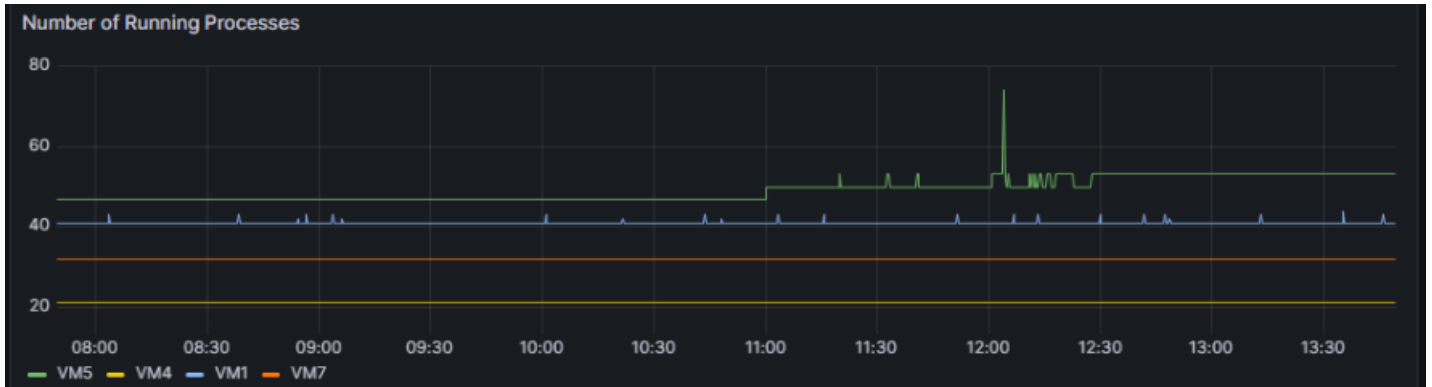


This panel uses the same query and endpoints as the previous panel, but it's a timeseries visualization type. We also picked units for the y-axis and limited to a 0%-100% range:



The reason we added a min and a max percentages is because the CPU usage, as calculated by the query we used, falls into the negative when a VM is down, which does not carry much information and also distorts the graph.

Number of Running Processes



Every machine whose processes you want to monitor needs to have `process_exporter` installed.

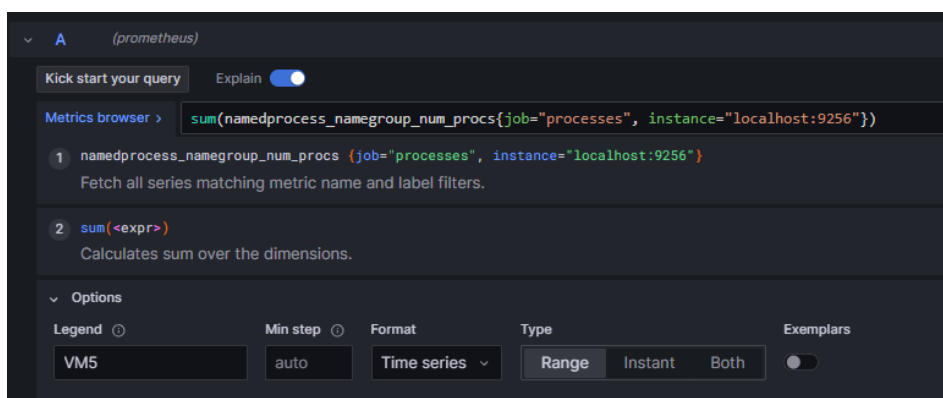
On the network monitoring VM, add this new job to `prometheus.yml` and restart prometheus:

```
scrape_configs:
# <...>

- job_name: "processes"
  scrape_interval: 10s

  static_configs:
    - targets:
      - "localhost:9256"
      - "192.168.10.4:9256"
      - "192.168.10.1:9256"
      - "192.168.10.7:9256"
```

Create a graph type visualization and add one query like this for for each target:



Network Usage Graph



This metric can be read using `node_exporter`. The endpoints as installed for the state panels are sufficient.

The query in Grafana is this:

Kick start your query ☒ Explain

Metrics browser > `rate(node_network_transmit_bytes_total{job="dns",device="enp0s8"}[5m])`

- `node_network_transmit_bytes_total {job="dns", device="enp0s8"}`
Fetch all series matching metric name and label filters.
- `rate(<expr>[5m])`
Calculates the per-second average rate of increase of the time series in the range vector. Breaks in monotonicity (such as resets) are handled by the range vector, allowing for missed scrapes or imperfect alignment of scrape cycles with the range's time period.

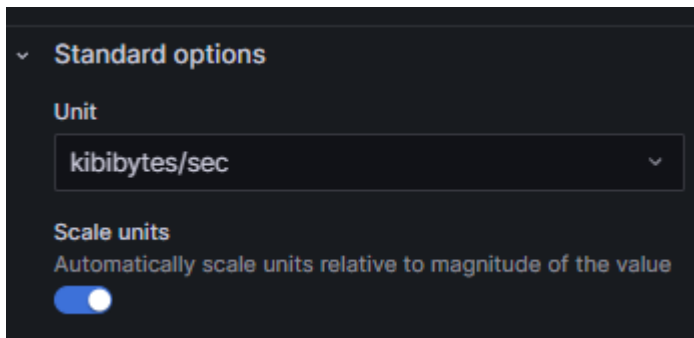
Options

Legend ⓘ Min step ⓘ Format Type Exemplars

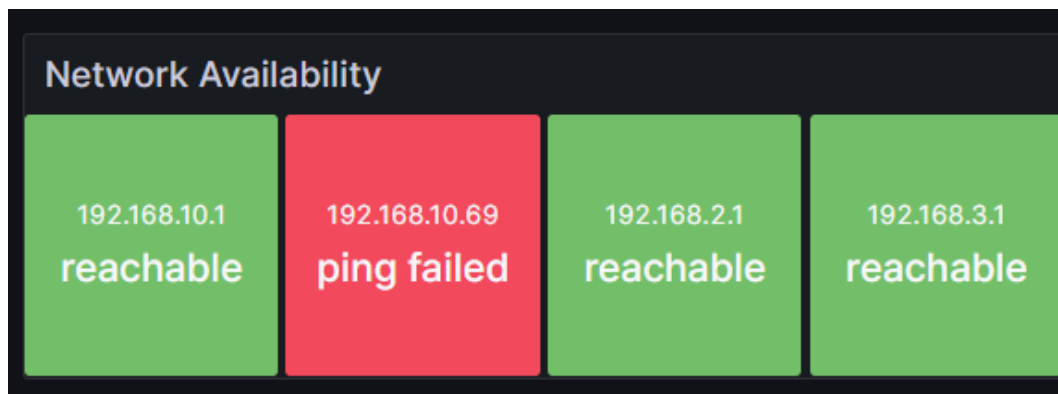
VM4 (DNS) auto Time series Range Instant Both ☐

Add one query for each VM. The query calculates the rate of network traffic per second. We made a panel for each network adapter (device). All of our machines have the same network adapters with the same purposes, since they're all made from the same template.

The panel is simply a time series visualization, the only special thing is that we specified a unit in the Standard Options section. We also set it to scale automatically:



Ping



To check the network availability, we added one ping test per team using blackbox. This is done using a blackbox probe. To establish this probe, add the following job to `prometheus.yml` and restart prometheus:

```
scrape_configs:
  # <...>

  - job_name: "network_ping"
    metrics_path: /probe
    params:
      module: [icmp]
    static_configs:
      - targets:
        - "8.8.8.8" # external IP address
        - "192.168.1.1"
        - "192.168.2.1"
        - "192.168.3.1"
        - "192.168.4.1"
        - "192.168.5.1"
        - "192.168.6.1"
        - "192.168.7.1"
        - "192.168.9.1"
        - "192.168.10.1"
      relabel_configs: # this is required according to
blackbox_exporter README
```

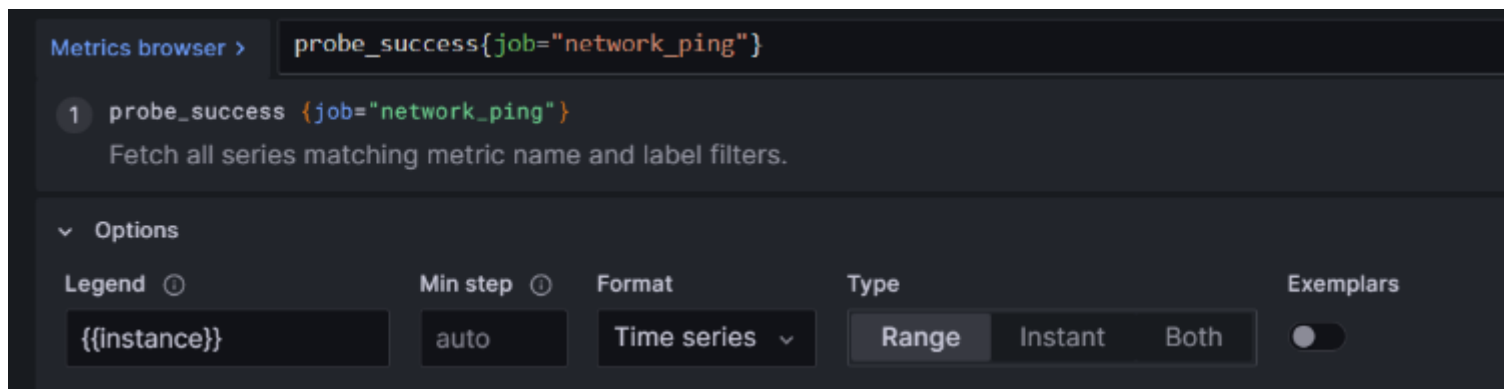
```

- source_labels: [__address__]
  target_label: __param_target
- source_labels: [__param_target]
  target_label: instance
- target_label: __address__
  replacement: 127.0.0.1:9115 # (blackbox exporter uses this
port)

```

We ping one VM from each team, as well as Google's DNS server 8.8.8.8 (to check if we can reach outside our private network).

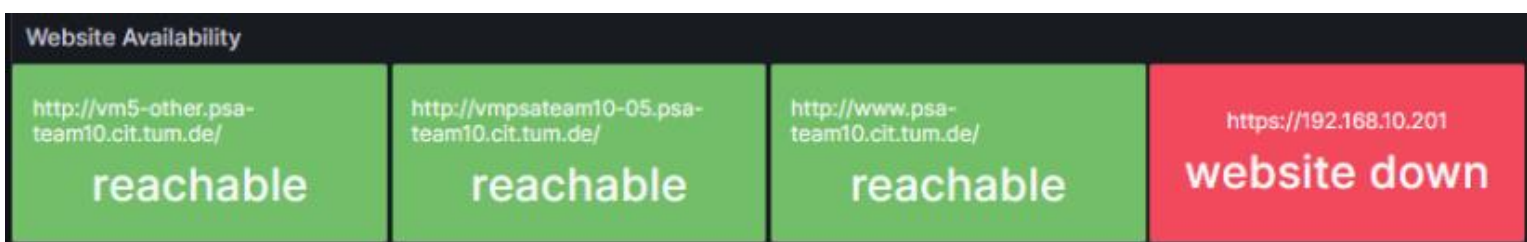
In Grafana, the return value of the probe can be queried using:



"network_ping" being the name of the job that does the blackbox probe.

The panel formatting is the same as the state panels. The only thing of note is that the value mapping uses "reachable" and "ping failed" instead of "UP" and "DOWN" and that there are custom legend names, which are displayed above the values.

Website Availability



This panel also uses blackbox_exporter.

Add the following job to prometheus.yml:

```

- job_name: "websites"
  scrape_interval: 20s
  metrics_path: /probe
  params:

```

```

    module: [http_2xx]
    static_configs:
      - targets:
        - "https://vmptsateam10-05.psa-team10.cit.tum.de"
        - "https://www.psa-team10.cit.tum.de/"
        - "https://vm5-other.psa-team10.cit.tum.de/"
        - "https://vmptsateam10-05.psa-team10.cit.tum.de/~kastl/"
        - "https://vmptsateam10-05.psa-team10.cit.tum.de/~kastl/cgi-
bin/print_time.py"
        - "http://vmptsateam10-05.psa-team10.cit.tum.de/"
        - "http://www.psa-team10.cit.tum.de/"
        - "http://vm5-other.psa-team10.cit.tum.de/"
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: localhost:9115

```

The websites whose state should be monitored are the targets. Also of note, the module, http_2xx, is customized for our setup. It can be found in blackbox.yml (if not, create it). You need to change it to this:

```

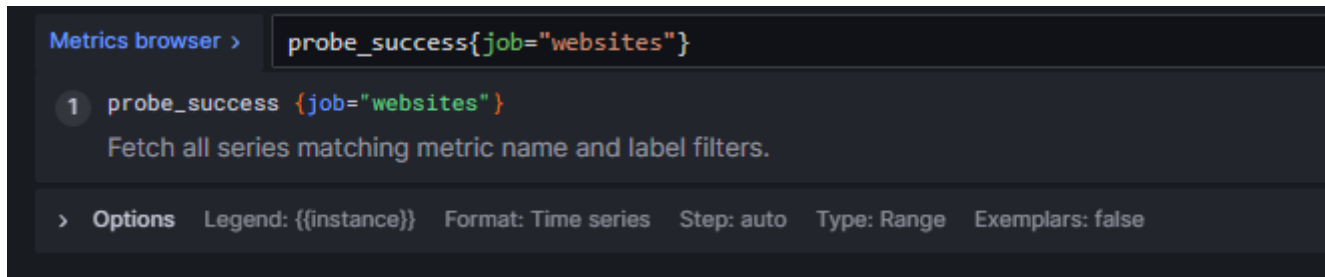
http_2xx:
  prober: http
  http:
    proxy_url: "localhost:9090" # irrelevant, we don't want to use
the proxy for any of our tests. but proxy_url is required when using
no_proxy
    no_proxy: "vmptsateam10-05.psa-team10.cit.tum.de,www.psa-
team10.cit.tum.de,vm5-other.psa-
team10.cit.tum.de,192.168.10.5,192.168.10.201,127.0.0.1,127.0.1.1"
    valid_http_versions: ["HTTP/1.1", "HTTP/2.0"]
    preferred_ip_protocol: "ip4"
    follow_redirects: true
    fail_if_not_ssl: false
    fail_if_ssl: false
    valid_status_codes: []
    tls_config:
      insecure_skip_verify: true

```

This module is configured to not use the TUM proxy, as our websites can't be accessed through it, for any of the domains/IP addresses in the value

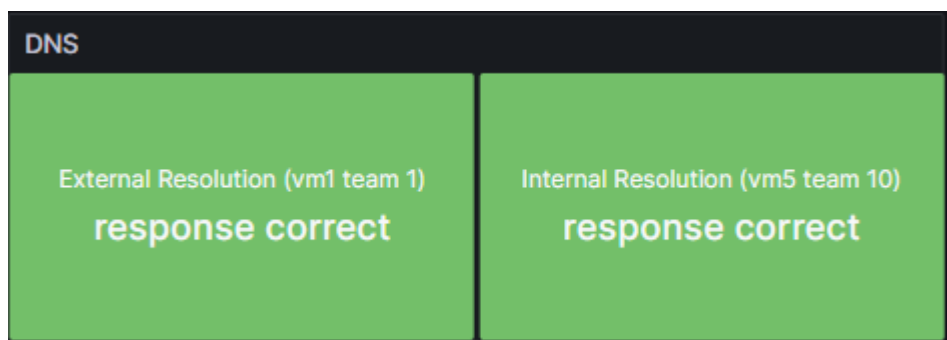
of `no_proxy`. It also bypasses SSL certificate verification through `insecure_skip_verify: true`.

In Grafana, the results of the blackbox probe can be queried using:



The Grafana panel has the same format as the ping panels. The only difference is that the value mappings use slightly different words.

DNS Availability and Correctness



This panel uses the `blackbox_exporter` endpoint on the Prometheus Server. We wanted to test one query of a domain name outside of our team (external) and one inside our team (internal). Since `blackbox_exporter` can only probe one query per job, we had to add two jobs to `prometheus.yml`:

```
- job_name: "dns_team1"
  metrics_path: /probe
  params:
    module: [dns_external]
  static_configs:
    - targets:
        - "192.168.10.4"
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
```

```

        replacement: "127.0.0.1:9115"
- job_name: "dns_team10"
  metrics_path: /probe
  params:
    module: [dns_internal]
  static_configs:
    - targets:
        - "192.168.10.4"
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: "127.0.0.1:9115"

```

The DNS queries are specified within the custom modules for these jobs. These modules need to be placed in `/etc/blackbox_exporter/blackbox.yml`:

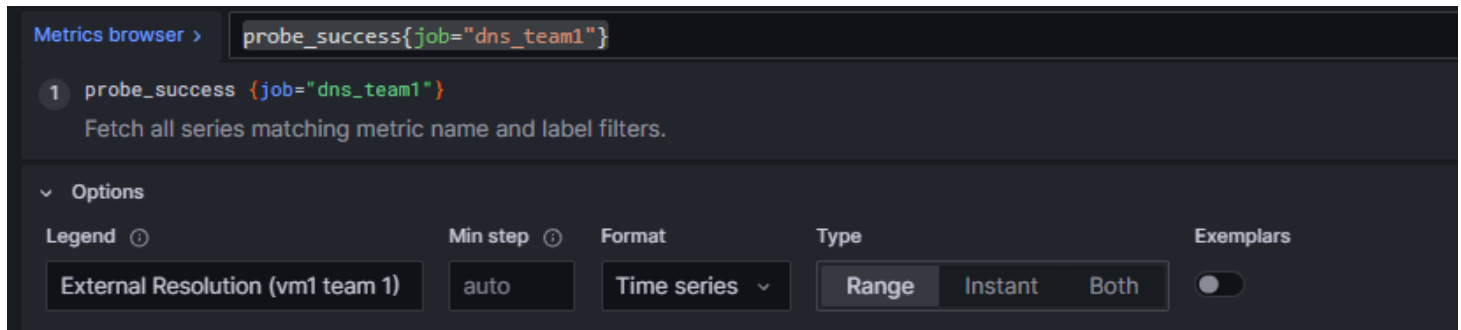
```

dns_external:
  prober: dns
  timeout: 5s
  dns:
    preferred_ip_protocol: "ip4"
    # what name to send a DNS query for
    query_name: "vm1.psa-team1.cit.tum.de"
    query_type: "A"
    valid_rcodes: ["NOERROR", "NOTAUTH"]
    validate_answer_rrs:
      fail_if_not_matches_regexp:
        - "(?s).*192.168.1.1(?s).*"
dns_internal:
  prober: dns
  timeout: 5s
  dns:
    preferred_ip_protocol: "ip4"
    # what name to send a DNS query for
    query_name: "vm1.psa-team1.cit.tum.de"
    query_type: "A"
    valid_rcodes: ["NOERROR"]
    validate_answer_rrs:
      fail_if_not_matches_regexp:
        - "(?s).*192.168.10.5(?s).*"

```

The first model queries a domain name from team 1, the second one from our own team. `fail_if_not_matches_regex` checks to see if the response contains the correct IP address.

In Grafana, the metrics provided by these jobs are queried using:



The panel formatting is the same as the ping panels, just with different words.

NTP

We kept having issues with our Prometheus server being out of synch with the system clock, which led to Grafana not getting updates from Prometheus, so we enabled synchronization using ntp.

```
sudo apt update
sudo apt install ntp
sudo systemctl start ntp
sudo systemctl enable ntp
```

Run `ntpq -p` to verify that it's active, it should print something like this:

remote	refid	st	t	when	poll	reach	delay	offset	jitter
0.ubuntu.pool.ntp.org	.POOL.	16	p	-	64	0	0.000	+0.000	0.000
1.ubuntu.pool.ntp.org	.POOL.	16	p	-	64	0	0.000	+0.000	0.000
2.ubuntu.pool.ntp.org	.POOL.	16	p	-	64	0	0.000	+0.000	0.000
3.ubuntu.pool.ntp.org	.POOL.	16	p	-	64	0	0.000	+0.000	0.000
ntp.ubuntu.com	.POOL.	16	p	-	64	0	0.000	+0.000	0.000

And add rules allowing traffic on udp port 123 to the firewall.

Tests

The tests are on VM9.

```
#!/bin/bash

failed_tests=0

fail() {
    ((failed_tests++))
    echo "FAIL $@"
}

ok() {
    echo "OK $@"
}

dir_exists() {
    if [ -d "$1" ]; then
        ok "$1 exists."
    else
        fail "$1 does not exist."
    fi
}

file_exists() {
    if [ -f "$1" ]; then
        ok "$1 exists."
    else
        fail "$1 does not exist."
    fi
}

active() {
    systemctl is-active --quiet $1
    if [ $? -eq 0 ]; then
        ok "$1 is running."
    else
        fail "$1 is inactive."
    fi
}

promtool check config "/etc/prometheus/prometheus.yml" > /dev/null
2>&1

if [ $? -eq 0 ]; then
    ok "prometheus.yml is valid."
else
```

```

    fail "prometheus.yml is invalid."
fi

dir_exists "/etc/prometheus"
dir_exists "/etc/blackbox_exporter"
dir_exists "/etc/process_exporter"
#data directories
dir_exists "/var/lib/prometheus"
dir_exists "/var/lib/grafana"

#executables
ex_path="/usr/local/bin"
file_exists "${ex_path}/prometheus"
file_exists "${ex_path}/node_exporter"
file_exists "${ex_path}/blackbox_exporter"
file_exists "${ex_path}/process-exporter"
file_exists "${ex_path}/alertmanager"

#services
active prometheus
active node_exporter
active blackbox_exporter
active process_exporter
active prometheus_alertmanager
active grafana-server
active nftables

# grafana website
url="https://131.159.74.56:61056/login"
http_status=$(wget --no-proxy --no-check-certificate --spider --
server-response $url 2>&1 | grep "HTTP/" | awk '{print $2}')

if [ "$http_status" == "200" ]; then
    ok "Grafana website is available (${url} 200 OK)"
else
    fail "Grafana website error (${url} STATUS: ${http_status})"
    if [ "$https_status" == "" ]; then
        echo -e "\tno http response/status code"
    fi
fi

# prometheus website
# this is only accessible on localhost and will complain that
javascript is not installed (status code 405) even when working
correctly
# so status code 200 isn't expected
url="http://localhost:9090"

```



```

http_status=$(curl --noproxy "*" -k "$url" -L -I 2>&1 | grep "HTTP/"
| awk '{print $2}')

if [ "$http_status" == "405" ] || [ "$http_status" == "200" ]; then
    ok "Prometheus is available (${url} status
code ${http_status} (405 is fine too))"
else
    fail "Prometheus error (${url} STATUS: ${http_status})"
    if [ "$https_status" == "" ]; then
        echo -e "\tno http response/status code"
    fi
fi

echo "Tests failed: $failed_tests"

```

(Screenshot of the entire set up in case anything were to break between deadline and grading. Note: If certain things don't show up you might need to log out and log back in again.)

