## **Installing Alert manager on Prometheus Server**

## 

## **Creating Alert Rules**

[Prometheus Alerting](https://prometheus.io/docs/alerting/overview/) is separated into two parts. The first part is handled by the Prometheus server and includes generating alerts based on alert rules and sending them to [Alertmanager](https://github.com/prometheus/alertmanager). The second part is done by Alertmanager, which manages received alerts and sends them to the appropriate receivers, depending on the configuration.

In this step, you’ll learn the basic syntax of alert rules as you create an alert rule to check if your server is available.

First, create a file to store your alerts. Create an empty file named alert.rules.yml in the /etc/prometheus directory:

sudo touch /etc/prometheus/alert.rules.yml

As this file is part of the Prometheus configuration, make sure the ownership is set to the **prometheus** user you created in the prerequisite Prometheus tutorial:

sudo chown prometheus:prometheus /etc/prometheus/alert.rules.yml

With the alerts file in place, we need to tell Prometheus about it by adding the appropriate directive to the configuration file.

Open the Prometheus configuration file in your editor:

sudo nano /etc/prometheus/prometheus.yml

Add the rule\_files directive after the global directive to make Prometheus load your newly-created alerts file when Prometheus starts.

/etc/prometheus/prometheus.yml

global:

scrape\_interval: 15s

rule\_files:

- alert.rules.yml

scrape\_configs:

...

Save the file and exit your text editor.

Now let’s build a rule that checks if the endpoint is down.

In order to make the alert rule, you’ll use Blackbox Exporter’s probe\_success metric which returns **1** if the endpoint is up and **0** if it isn’t.

The probe\_success metric contains two labels: the instance label with the address of the endpoint, and the job label with the name of the exporter that collected the metric.

Open the alert rules file in your editor:

sudo nano /etc/prometheus/alert.rules.yml

Like the Prometheus configuration file, the alerts rule file uses the YAML format, which strictly forbids tabs and requires two spaces for indentation. Prometheus will fail to start if the file is incorrectly formatted.

First, we’ll create an alert rule called EndpointDown to check if the probe\_sucess metric equals **0** with a duration of **10** seconds. This ensures that Prometheus will not send any alert if the endpoint is not available for less than 10 seconds. You’re free to choose whatever duration you want depending on your application type and needs.

Also, we’ll attach two labels denoting critical severity and a summary of the alert, so we can easily manage and filter alerts.

If you want to include more details in the alert’s labels and annotations, you can use the {{ $labels.metrics\_label }} syntax to get the label’s value. We’ll use this to include the endpoint’s address from the metric’s instance label.

Add the following rule to the alerts file:

/etc/prometheus/alert.rules.yml

groups:

- name: alert.rules

rules:

- alert: HostOutOfMemory

expr: node\_memory\_MemAvailable / node\_memory\_MemTotal \* 100 < 94

for: 5m

labels:

severity: warning

annotations:

summary: "Host out of memory (instance {{ $labels.instance }})"

description: "Node memory is filling up (< 10% left)\n VALUE = {{ $value }}\n LABELS: {{ $labels }}"

- alert: HostMemoryUnderMemoryPressure

expr: rate(node\_vmstat\_pgmajfault[1m]) > -1

for: 5m

labels:

severity: warning

annotations:

summary: "Host memory under memory pressure (instance {{ $labels.instance }})"

description: "The node is under heavy memory pressure. High rate of major page faults\n VALUE = {{ $value }}\n LABELS: {{ $labels }}"

- alert: EndpointDown

expr: probe\_success == 0

for: 10s

labels:

severity: "critical"

annotations:

summary: "Endpoint {{ $labels.instance }} down"

Save the file and exit your text editor.

Before restarting Prometheus, make sure your alerts file is syntactically correct using the following promtool command:

sudo promtool check rules /etc/prometheus/alert.rules.yml

The output contains the number of rules found in the file, along with information about whether or not the rules are syntactically correct:

Output

Checking /etc/prometheus/alert.rules.yml

SUCCESS: 3 rules found

Lastly, restart Prometheus to apply the changes:

sudo systemctl restart prometheus

Verify the service is running with the status command:

sudo systemctl status prometheus

If the service’s status isn’t active, follow the on-screen logs and retrace the preceding steps to resolve the problem before continuing the tutorial.

With the alert rules in place, we can download and install Alertmanager.

## **Step 2 — Downloading Alertmanager**

Blackbox Exporter is configured and our alert rules are in place. Let’s download and install Alertmanager to process the alerts received by Prometheus.

You can find the latest binaries along with their checksums on the [Prometheus download page](https://prometheus.io/download/). Download and unpack the current stable version of Alertmanager into your home directory:

cd ~

curl -LO https://github.com/prometheus/alertmanager/releases/download/v0.14.0/alertmanager-0.14.0.linux-amd64.tar.gz

Before unpacking the archive, verify the file’s checksums using the following sha256sum command:

sha256sum alertmanager-0.14.0.linux-amd64.tar.gz

Compare the output from this command with the checksum on the Prometheus download page to ensure that your file is both genuine and not corrupted.

Output

caddbbbe3ef8545c6cefb32f9a11207ae18dcc788e8d0fb19659d88c58d14b37 alertmanager-0.14.0.linux-amd64.tar.gz

If the checksums don’t match, remove the downloaded file and repeat the preceding steps to re-download the file.

Once you’ve verified the download, unpack the archive:

tar xvf alertmanager-0.14.0.linux-amd64.tar.gz

This creates a directory called alertmanager-0.14.0.linux-amd64 containing two binary files (alertmanager and amtool), a license and an example configuration file.

Move the two binary files to the /usr/local/bin directory:

sudo mv alertmanager-0.14.0.linux-amd64/alertmanager /usr/local/bin

sudo mv alertmanager-0.14.0.linux-amd64/amtool /usr/local/bin

Set the user and group ownership on the binary files to the **alertmanager** user you created in Step 1:

sudo chown alertmanager:alertmanager /usr/local/bin/alertmanager

sudo chown alertmanager:alertmanager /usr/local/bin/amtool

Remove the leftover files from your home directory as they are no longer needed:

rm -rf alertmanager-0.14.0.linux-amd64 alertmanager-0.14.0.linux-amd64.tar.gz

Now that the required files are in the appropriate location, we can configure Alertmanager to send notifications for alerts over email.

## **Step 3 — Configuring Alertmanager To Send Alerts Over Email**

In this step, you’ll create the directory and files to store Alertmanager’s data and configuration settings, and then configure Alertmanager to send alerts via email.

Following the standard Linux conventions, we’ll create a directory in /etc to store Alertmanager’s configuration file.

sudo mkdir /etc/alertmanager

Set the user and group ownership for the newly-created directory to the **alertmanager** user:

sudo chown alertmanager:alertmanager /etc/alertmanager

We’ll store the configuration file in the alertmanager.yml file, so create this file and open it in your editor:

sudo nano /etc/alertmanager/alertmanager.yml

Like other Prometheus-related files, this one uses YAML format as well, so make sure to use two spaces instead of tabs for indentation.

We’ll configure Alertmanager to send emails using Postfix, which you installed following the prerequisite tutorial. We need to provide the SMTP server’s address, using the smtp\_smarthost directive, as well as the address we want to send emails from, using the smtp\_from directive. As Postfix is running on the same server as Alertmanager, the server’s address is localhost:25. We’ll use the **alertmanager** user for sending emails.

By default, Postfix doesn’t have TLS configured, so we need to tell Alertmanager to allow non-TLS SMTP servers using the smtp\_require\_tls directive.

Put the SMTP configuration under the global directive, as it’s used to specify parameters valid in all other configuration contexts. This includes SMTP configuration in our case, and can also include API tokens for various integrations:

Alertmanager config file part 1 - /etc/alertmanager/alertmanager.yml

global:

smtp\_smarthost: 'localhost:25'

smtp\_from: 'alertmanager@your\_domain'

smtp\_require\_tls: false

**Note:** Make sure to replace your\_domin in the smtp\_from directive with your domain name.

At this point, Alertmanager knows how to send emails, but we need to define how it will handle incoming alerts using the route directive. The route directive is applied to every incoming alert and defines properties such as how Alertmanager will group alerts, who is the default recipient, or how long Alertmanager will wait before sending an initial alert.

To group alerts, use the group\_by sub-directive, which takes an inline array of labels (such as ['label-1', 'label-2']). Grouping ensures that alerts containing the same labels will be grouped and sent in the same batch.

Every route directive has a single receiver defined using the receiver sub-directive. If you want to add multiple receivers, you’ll need to either define multiple receivers under the same directive or nest multiple route directives using the routes sub-directive. In this tutorial, we’ll cover the first approach to configure Slack alerts.

In this case, we’ll only group by Blackbox’s instance label and the severity label we attached to the alert in step 6, ensuring we’ll get multiple alerts for our endpoint with critical severity in one mail.

Add the following group\_by directive:

Alertmanager config file part 2 - /etc/alertmanager/alertmanager.yml

...

route:

group\_by: ['instance', 'alert']

Next, we’ll define intervals, such as how long Alertmanager will wait before sending initial and new alerts.

Using the group\_wait sub-directive, we’ll define how long Alertmanager will wait before sending the initial alert. During this period, Alertmanager will wait for Prometheus to send other alerts if they exist so they can be sent in the same batch. As we only have one alert, we’ll select an arbitrary value of 30 seconds.

Next, using the group\_interval interval, we’ll define how long Alertmanager will wait before sending the next batch of alerts if there are new alerts in the same group. You’re free to choose any value depending on your needs, but we’ll set this to every 5 minutes.

The last interval we’ll configure is the repeat\_interval, which defines how long Alertmanager will wait before it sends notification if alerts are not resolved yet. You can choose whatever value suits your needs, but we’ll use the arbitrary value of 3 hours.

Lastly, using the receiver sub-directive, define who will receive notifications for the alerts. We’ll use a receiver called team-1, which we will define later.

Modify the route directive so it looks like this:

Alertmanager config file part 2 - /etc/alertmanager/alertmanager.yml

route:

group\_by: ['instance', 'severity']

group\_wait: 30s

group\_interval: 5m

repeat\_interval: 3h

receiver: team-1

If you want to match and send notifications only about specific alerts, you can use the match and match\_re sub-directives to filter out alerts by their label’s value. The match sub-directive represents equality match, where the match\_re sub-directive represents matching via regular expressions.

Now we’ll configure the team-1 receiver so you can receive notifications for alerts. Under the receivers directive you can define receivers containing the name and appropriate configuration sub-directive. The list of available receivers and instructions on how to configure them is available as the part of [Alertmanager’s documentation](https://prometheus.io/docs/alerting/configuration/#%3Creceiver).

In order to configure the team-1 email receiver, we’ll use the email\_configs sub-directive under the receivers directive:

Alertmanager config file part 3 - /etc/alertmanager/alertmanager.yml

receivers:

- name: 'team-1'

email\_configs:

- to: 'your-email-address'

At this point, you have configured Alertmanager to send notifications for alerts to your e-mail address. Your configuration file should look like:

Alertmanager config file - /etc/alertmanager/alertmanager.yml

global:

resolve\_timeout: 5m

route:

group\_by: ['alertname']

group\_wait: 10s

group\_interval: 10s

repeat\_interval: 1h

receiver: email-me

receivers:

# receiver configuration https://prometheus.io/docs/alerting/configuration/#receiver

- name: 'email-me'

email\_configs:

- to: shreeram.gholap@gslab.com

from: prometheus\_admin@gslab.com

smarthost: smtp.gmail.com:587

auth\_username: "cloudadmin@gslab.com"

auth\_identity: "cloudadmin@gslab.com"

auth\_password: "vputrgdtmpzklozw"

inhibit\_rules:

- source\_match:

severity: 'critical'

In the next step, we’ll configure Alertmanager to send alerts to your Slack channel. If you don’t want to configure Slack, you can skip straight to step 10 where we’ll create the service file and configure Prometheus to work with Alertmanager.

## **Step 9 — Running Alertmanager**

Let’s get Alertmanager up and running. We’ll first create a systemd unit file for Alertmanager to manage its service using systemd. Then we’ll update Prometheus to use Alertmanager.

Create a new systemd unit file and open it in your text editor:

sudo nano /etc/systemd/system/alertmanager.service

Add the following to the file to configure systemd to run Alertmanager as the **alertmanager** user, using the configuration file located at /etc/alertmanager/alertmanager.yml and Alertmanager’s URL, configured to use your server’s IP address:

/etc/systemd/system/alertmanager.service

[Unit]

Description=Alertmanager

Wants=network-online.target

After=network-online.target

[Service]

User=alertmanager

Group=alertmanager

Type=simple

WorkingDirectory=/etc/alertmanager/

ExecStart=/usr/local/bin/alertmanager --config.file=/etc/alertmanager/alertmanager.yml --web.external-url http://your\_server\_ip:9093

[Install]

WantedBy=multi-user.target

This will run Alertmanager as the **alertmanager** user. It also tells Alertmanager to use the URL http://your\_server\_ip:9093 for its Web UI, where 9093 is Alertmanager’s default port. Be sure to include the protocol (http://) or things won’t work.

Save the file and close your text editor.

Next, we need to tell Prometheus about Alertmanager by adding the appropriate Alertmanager service discovery directory to the Prometheus configuration file. By default, Alertmanager is running on port 9093, and since it’s on the same server as Prometheus, we’ll use the address localhost:9093.

Open the Prometheus configuration file:

sudo nano /etc/prometheus/prometheus.yml

After the rule\_files directive, add the following alerting directive:

Prometheus configuration file - /etc/prometheus/prometheus.yml

...

rule\_files:

- alert.rules.yml

alerting:

alertmanagers:

- static\_configs:

- targets:

- localhost:9093

...

Once you’re done, save the file and close your text editor.

In order to be able to follow URLs from the alerts you receive, you need to tell Prometheus the IP address or domain name of your server using the -web.external-url flag when you start Prometheus.

Open the systemd unit file for Prometheus:

sudo nano /etc/systemd/system/prometheus.service

Replace the existing ExecStart line with the following one:

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.external-url http://your\_server\_ip

Your new Prometheus unit file will look like this:

Prometheus service file - /etc/systemd/system/prometheus.service

[Unit]

Description=Prometheus

Wants=network-online.target

After=network-online.target

[Service]

User=prometheus

Group=prometheus

Type=simple

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.external-url http://your\_server\_ip

[Install]

WantedBy=multi-user.target

Save the file and close your text editor.

Reload systemd and restart Prometheus to apply the changes:

sudo systemctl daemon-reload

sudo systemctl restart prometheus

Make sure Prometheus is working as intended by checking the service’s status:

sudo systemctl status prometheus

If the service’s status isn’t active (running), follow the on-screen logs and retrace the preceding steps to resolve the problem before continuing the tutorial.

Finally, start Alertmanager for the first time:

sudo systemctl start alertmanager

Check the service’s status to make sure Alertmanager is working as intended:

sudo systemctl status alertmanager

If the service’s status isn’t active (running), follow the on-screen messages and retrace the preceding steps to resolve the problem before continuing the tutorial.

Lastly, enable the service to make sure Alertmanager will start when the system boots:

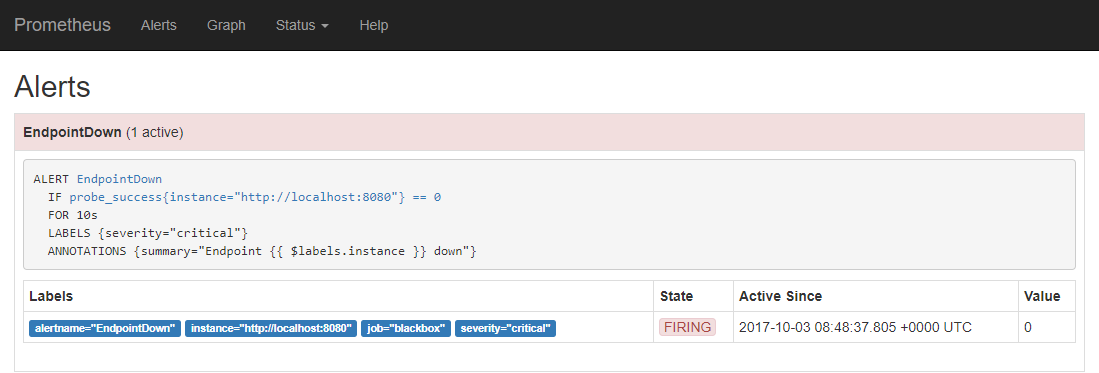
sudo systemctl enable alertmanager

To access Alertmanager’s Web UI, allow traffic to port 9093 through your firewall:

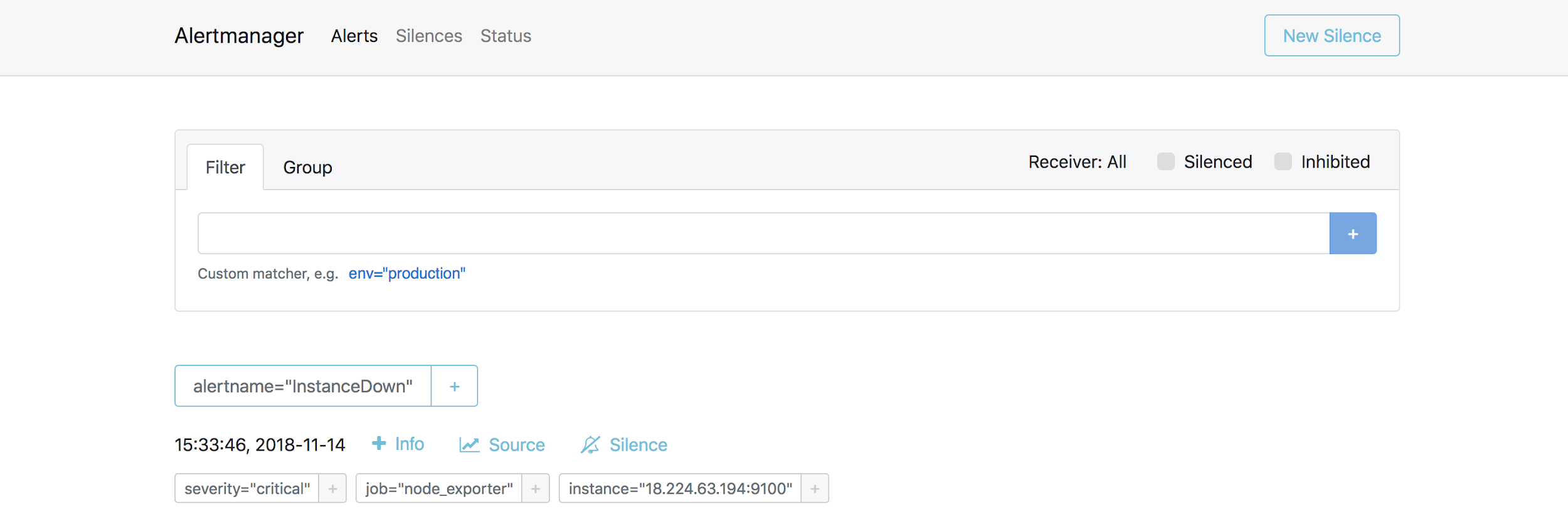
sudo ufw allow 9093/tcp

Alertmanager is now configured to send notifications for alerts via email . Let’s ensure it works.

You can also check the alert’s status from the Prometheus Web UI, by pointing your web browser to the http://your\_server\_ip/alerts. You’ll be asked to enter the username and password you chose by following the Prometheus tutorial. By clicking on the alert name, you’ll see the status, the alert rule, and associated labels:



Once you’ve verified Alertmanager is work

AlertManager received an alert from Prometheus server

## **[FIRING:1] HostOutOfMemory (localhost:9100 node\_exporter warning)**

|  |  |
| --- | --- |
| Inbox | x |



|  |  |  |
| --- | --- | --- |
| |  | | --- | | **cloudadmin@gslab.com** | | 12:32 PM (23 minutes ago) |

|  |  |
| --- | --- |
|  |  |



|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | to me | | | |

|  |
| --- |
| 1 alert for alertname=HostOutOfMemory |
| |  | | --- | | [**View in AlertManager**](http://10.43.12.69:9093/#/alerts?receiver=email-me) | | **[1] Firing** | | **Labels**  alertname = HostOutOfMemory  instance = localhost:9100  job = node\_exporter  severity = warning  **Annotations**  description = Node memory is filling up (< 10% left) VALUE = 93.57099464074119 LABELS: map[instance:localhost:9100 job:node\_exporter]  summary = Host out of memory (instance localhost:9100)  [Source](http://test:9090/graph?g0.expr=node_memory_MemAvailable+%2F+node_memory_MemTotal+%2A+100+%3C+94&g0.tab=1) | |

|  |
| --- |
| [Sent by AlertManage](http://10.43.12.69:9093) |