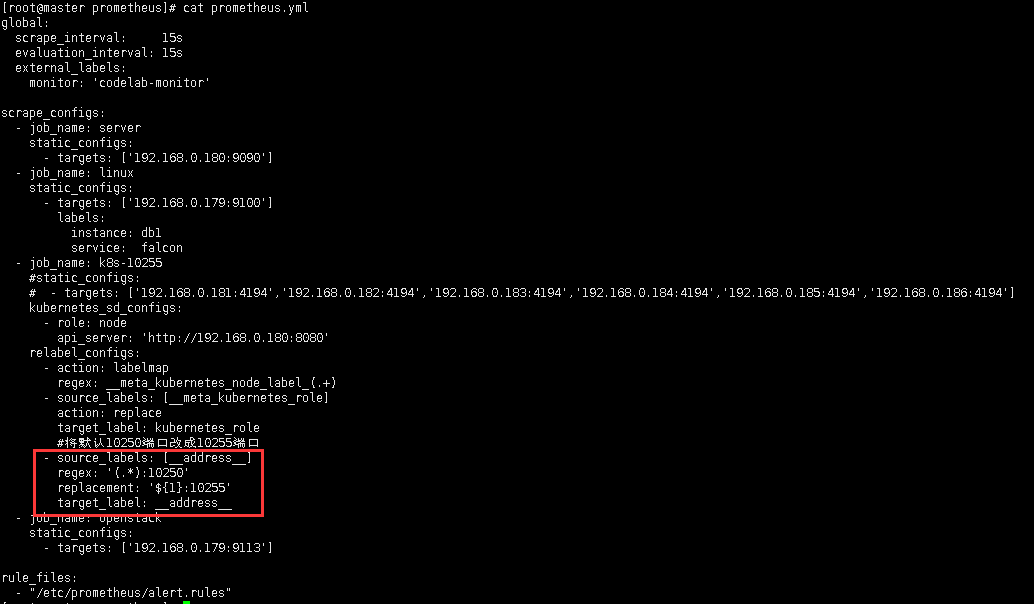
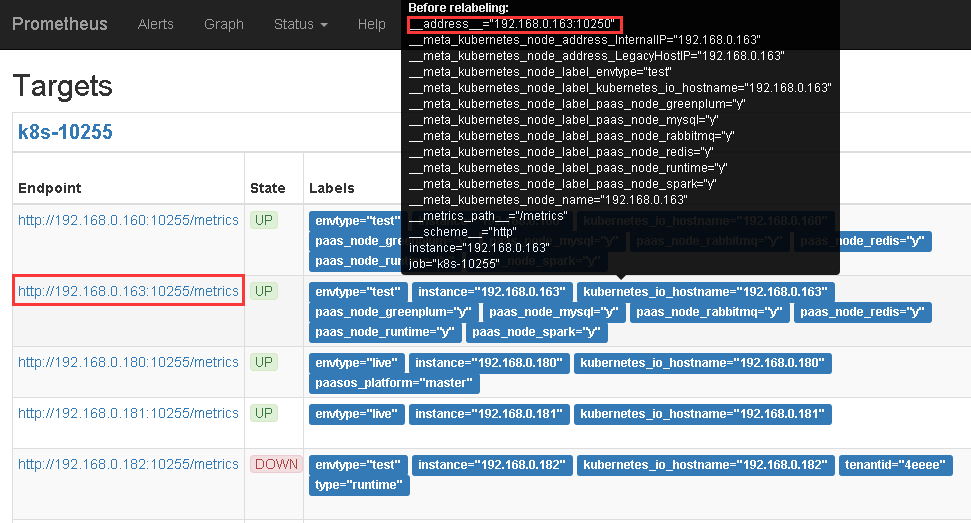
# 将默认10250端口改为10255





# Life of a Label | Robust Perception

https://www.robustperception.io/life-of-a-label/

[Life of a Label](https://www.robustperception.io/life-of-a-label/)

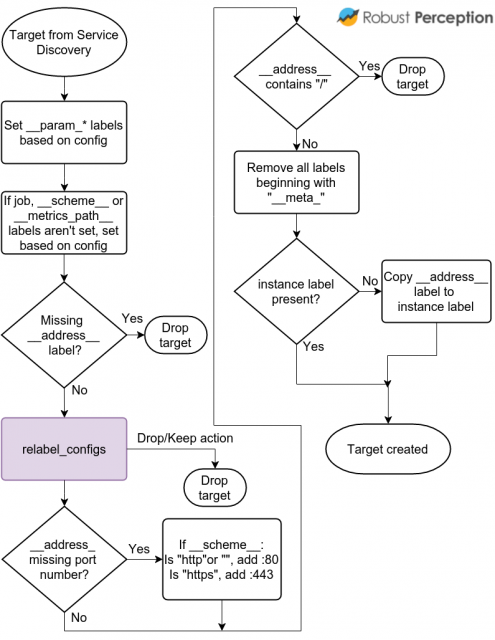
[***Brian Brazil***](https://www.robustperception.io/author/brian/) ***March 9, 2016***

**Prometheus labels allow you to model your application deployment in the manner best suited to your organisation. As directly supporting every potential configurations would be impossible, we offer relabelling to give you the flexibility to configure things how you’d like.**

**How labels propagate can be a bit tricky to get your head around initially. The basic principle is that your service discovery provides you with metadata such as machine type, tags, region in \_\_meta\_\* labels, and which you then relabel into the labels you’d like for your targets to have with relabel\_configs. You can also filter targets with the drop and keep actions.**

**Similarly when actually scraping the targets, metric\_relabel\_configs allow you to tweak the timeseries coming back from the scrape. Filtering can also be used as a temporary measure** [**to deal with expensive metrics**](http://www.robustperception.io/dropping-metrics-at-scrape-time-with-prometheus/)**.**

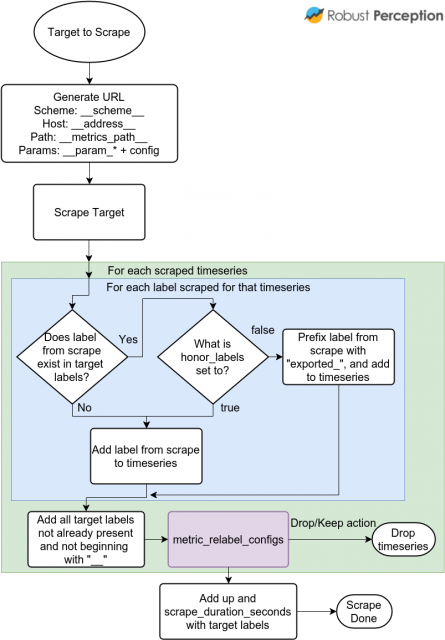
**To help you understanding how this all fits together, I’ve put together flowcharts of the semantics in Prometheus 0.17.0. These cover from how targets are created, scraped and what manipulations are performed before timeseries are inserted into the database:**

**[](http://www.robustperception.io/wp-content/uploads/2016/03/Life-of-a-Label-Target-Labels.png)**

**Targets and Targets Labels come from Service Discovery**

**As this point Prometheus knows the targets that’ll be scraped, and these are what you see on the Status page. The core here is the relabelling in purple. Everything else is defaults and sanity checks to make your life easier.**

**When a target is actually scraped, this is what happens:**

**[](http://www.robustperception.io/wp-content/uploads/2016/03/Life-of-a-Label-Scraping.png)**

**Labels from scrapes are a mix of Scraped Labels and Target Labels**

**The \_\_param\_\* labels contain the first value of each URL parameter, allowing you to relabel the first value. At scrape time, these are combined back with the second and subsequent parameter values.**

**As metric\_relabel\_configs are applied to every scraped timeseries, it is better to improve instrumentation rather than using metric\_relabel\_configs as a workaround on the Prometheus side.**

[**prometheus**](https://www.robustperception.io/tag/prometheus/)**,** [**relabelling**](https://www.robustperception.io/tag/relabelling/)**,** [**service discovery**](https://www.robustperception.io/tag/service-discovery/)

# relabel\_configs vs metric\_relabel\_configs | Robust Perception

https://www.robustperception.io/relabel\_configs-vs-metric\_relabel\_configs/

**We’ve looked at the full** [**Life of a Label**](https://www.robustperception.io/life-of-a-label/)**. Let’s focus on one of the most common confusions around relabelling.**

**It’s not uncommon for a user to share a Prometheus config with a valid relabel\_configs and wonder why it isn’t taking effect. This is often resolved by using metric\_relabel\_configs instead (the reverse has also happened, but it’s far less common). So let’s shine some light on these two configuration options.**

**Prometheus needs to know what to scrape, and that’s where service discovery and relabel\_configs come in. Relabel configs allow you to select** [**which targets you want scraped**](https://www.robustperception.io/automatically-monitoring-ec2-instances/)**, and** [**what the target labels will be**](https://www.robustperception.io/finding-consul-services-to-monitor-with-prometheus/)**. So if you want to say scrape this type of machine but not that one, use relabel\_configs.**

**metrics\_relabel\_configs by contrast are applied after the scrape has happened, but before the data is ingested by the storage system. So if there are some** [**expensive metrics you want to drop**](https://www.robustperception.io/dropping-metrics-at-scrape-time-with-prometheus/)**, or labels coming from the scrape itself (e.g. from the /metrics page) that you want to manipulate that’s where metrics\_relabel\_configs applies.**

**So as a simple rule of thumb: relabel\_config happens before the scrape, metrics\_relabel\_configs happens after the scrape. And if one doesn’t work you can always try the other!**

**Need help with relabelling?** [***Contact us***](mailto:prometheus@robustperception.io)**.**

[**prometheus**](https://www.robustperception.io/tag/prometheus/)**,** [**relabelling**](https://www.robustperception.io/tag/relabelling/)

# Dropping metrics at scrape time with Prometheus | Robust Perception

https://www.robustperception.io/dropping-metrics-at-scrape-time-with-prometheus/

[Dropping metrics at scrape time with Prometheus](https://www.robustperception.io/dropping-metrics-at-scrape-time-with-prometheus/)

[***Brian Brazil***](https://www.robustperception.io/author/brian/)***September 16, 2015***

**It’s easy to get carried away by the power of labels with Prometheus. In the extreme this can overload your**[**Prometheus**](https://prometheus.io/)**server, such as if you create a time series for each of hundreds of thousands of users. Thankfully there’s a way to deal with this without having to turn off monitoring or deploy a new version of your code.**

**Firstly you need to find which metric is the problem. Go to the expression browser on Prometheus (that’s the /graph endpoint) and evaluate  topk(20, count by (\_\_name\_\_, job)({\_\_name\_\_=~".+"})). This will return the 20 biggest time series by metric name and job, which one is the problem should be obvious.**

**Now that you know the name of the metric and the job it’s part of, you can modify the job’s scrape config to drop it. Let’s say it’s a metric called my\_too\_large\_metric. Add a metric\_relabel\_configs section to drop it:**

**scrape\_configs:**

**- job\_name: 'my\_job'**

**static\_configs:**

**- targets:**

**- my\_target:1234**

**metric\_relabel\_configs:**

**- source\_labels: [ \_\_name\_\_ ]**

**regex: 'my\_too\_large\_metric'**

**action: drop**

**All the samples are still pulled from the job being scraped, so this should only be a temporary solution until you can push a fixed version of your code.**

[**prometheus**](https://www.robustperception.io/tag/prometheus/)**,**[**promql**](https://www.robustperception.io/tag/promql/)**,**[**relabelling**](https://www.robustperception.io/tag/relabelling/)**,**[**reliability**](https://www.robustperception.io/tag/reliability/)