# Introduction

## Overview | Prometheus

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

**OVERVIEW**

* [What is Prometheus?](https://prometheus.io/docs/introduction/overview/#what-is-prometheus?)
  + [Features](https://prometheus.io/docs/introduction/overview/#features)
  + [Components](https://prometheus.io/docs/introduction/overview/#components)
  + [Architecture](https://prometheus.io/docs/introduction/overview/#architecture)
* [When does it fit?](https://prometheus.io/docs/introduction/overview/#when-does-it-fit?)
* [When does it not fit?](https://prometheus.io/docs/introduction/overview/#when-does-it-not-fit?)

**What is Prometheus?**

[Prometheus](https://github.com/prometheus) is an open-source systems monitoring and alerting toolkit originally built at [SoundCloud](http://soundcloud.com/). Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user [community](https://prometheus.io/community). It is now a standalone open source project and maintained independently of any company. To emphasize this and clarify the project's governance structure, Prometheus joined the [Cloud Native Computing Foundation](https://cncf.io/) in 2016 as the second hosted project after [Kubernetes](http://kubernetes.io/).

For a more elaborate overview, see the resources linked from the [media](https://prometheus.io/docs/introduction/media/) section.

Features

Prometheus's main features are:

* a multi-dimensional [data model](https://prometheus.io/docs/concepts/data_model/) (time series identified by metric name and key/value pairs)
* a [flexible query language](https://prometheus.io/docs/querying/basics/) 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](https://prometheus.io/docs/instrumenting/pushing/) is supported via an intermediary gateway
* targets are discovered via service discovery or static configuration
* multiple modes of graphing and dashboarding support

Components

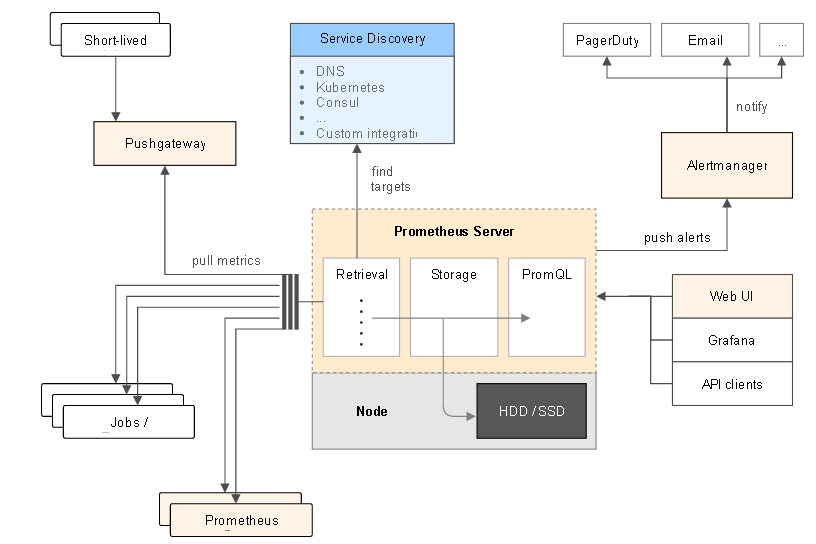
The Prometheus ecosystem consists of multiple components, many of which are optional:

* the main [Prometheus server](https://github.com/prometheus/prometheus) which scrapes and stores time series data
* [client libraries](https://prometheus.io/docs/instrumenting/clientlibs/) for instrumenting application code
* a [push gateway](https://github.com/prometheus/pushgateway) for supporting short-lived jobs
* special-purpose [exporters](https://prometheus.io/docs/instrumenting/exporters/) (for HAProxy, StatsD, Graphite, etc.)
* an [alertmanager](https://github.com/prometheus/alertmanager)
* various support tools

Most Prometheus components are written in [Go](https://golang.org/), making them easy to build and deploy as static binaries.

Architecture

This diagram illustrates the overall architecture of Prometheus and some of its ecosystem components:



Prometheus scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs. It stores all scraped samples locally and runs rules over this data to either record new time series from existing data or generate alerts. Grafana or other API consumers can be used to visualize the collected data.

**When does it fit?**

Prometheus works well for recording any purely numeric time series. It fits both machine-centric monitoring as well as monitoring of highly dynamic service-oriented architectures. In a world of microservices, its support for multi-dimensional data collection and querying is a particular strength.

Prometheus is designed for reliability, to be the system you go to during an outage to allow you to quickly diagnose problems. Each Prometheus server is standalone, not depending on network storage or other remote services. You can rely on it when other parts of your infrastructure are broken, and you do not have to set up complex infrastructure to use it.

**When does it not fit?**

Prometheus values reliability. You can always view what statistics are available about your system, even under failure conditions. If you need 100% accuracy, such as for per-request billing, Prometheus is not a good choice as the collected data will likely not be detailed and complete enough. In such a case you would be best off using some other system to collect and analyse the data for billing, and Prometheus for the rest of your monitoring.

## Installing | Prometheus

https://prometheus.io/docs/introduction/install/

INSTALLING

* [Using pre-compiled binaries](https://prometheus.io/docs/introduction/install/#using-pre-compiled-binaries)
* [From source](https://prometheus.io/docs/introduction/install/#from-source)
* [Using Docker](https://prometheus.io/docs/introduction/install/#using-docker)
  + [Volumes & bind-mount](https://prometheus.io/docs/introduction/install/#volumes-&-bind-mount)
  + [Custom image](https://prometheus.io/docs/introduction/install/#custom-image)
* [Using configuration management systems](https://prometheus.io/docs/introduction/install/#using-configuration-management-systems)

**Using pre-compiled binaries**

We provide precompiled binaries for most official Prometheus components. Check out the [download section](https://prometheus.io/download) for a list of all available versions.

**From source**

For building Prometheus components from source, see theMakefile targets in the respective repository.

**Note:** The documentation on this website refers to the latest stable release (excluding pre-releases). The branch [next-release](https://github.com/prometheus/docs/compare/next-release) refers to unreleased changes that are in master branches of source repos.

**Using Docker**

All Prometheus services are available as Docker images under the [prom](https://hub.docker.com/u/prom/) organization.

prom - Docker Hub <https://hub.docker.com/r/prom/>

https://hub.docker.com/r/prom/prometheus/

docker pull prom/Prometheus

Running Prometheus on Docker is as simple as docker run -p 9090:9090 prom/prometheus. This starts Prometheus with a sample configuration and exposes it on port 9090.

The Prometheus image uses a volume to store the actual metrics. For production deployments it is highly recommended to use the [Data Volume Container](https://docs.docker.com/engine/userguide/containers/dockervolumes/#creating-and-mounting-a-data-volume-container) pattern to ease managing the data on Prometheus upgrades.

To provide your own configuration, there are several options. Here are two examples.

Volumes & bind-mount

Bind-mount your prometheus.yml from the host by running:

docker run -p 9090:9090 -v /tmp/prometheus.yml:/etc/prometheus/prometheus.yml \

prom/prometheus

Or use an additional volume for the config:

docker run -p 9090:9090 -v /prometheus-data \

prom/prometheus -config.file=/prometheus-data/prometheus.yml

Custom image

To avoid managing a file on the host and bind-mount it, the configuration can be baked into the image. This works well if the configuration itself is rather static and the same across all environments.

For this, create a new directory with a Prometheus configuration and a Dockerfile like this:

FROM prom/prometheus

ADD prometheus.yml /etc/prometheus/

Now build and run it:

docker build -t my-prometheus .

docker run -p 9090:9090 my-prometheus

A more advanced option is to render the config dynamically on start with some tooling or even have a daemon update it periodically.

**Using configuration management systems**

If you prefer using configuration management systems you might be interested in the following third-party contributions:

Ansible:

* [griggheo/ansible-prometheus](https://github.com/griggheo/ansible-prometheus)
* [William-Yeh/ansible-prometheus](https://github.com/William-Yeh/ansible-prometheus)

Chef:

* [rayrod2030/chef-prometheus](https://github.com/rayrod2030/chef-prometheus)

Puppet:

* [puppet/prometheus](https://forge.puppet.com/puppet/prometheus)

SaltStack:

* [bechtoldt/saltstack-prometheus-formula](https://github.com/bechtoldt/saltstack-prometheus-formula)

## Getting started | Prometheus

https://prometheus.io/docs/introduction/getting\_started/

**GETTING STARTED**

* [Downloading and running Prometheus](https://prometheus.io/docs/introduction/getting_started/#downloading-and-running-prometheus)
* [Configuring Prometheus to monitor itself](https://prometheus.io/docs/introduction/getting_started/#configuring-prometheus-to-monitor-itself)
* [Starting Prometheus](https://prometheus.io/docs/introduction/getting_started/#starting-prometheus)
* [Using the expression browser](https://prometheus.io/docs/introduction/getting_started/#using-the-expression-browser)
* [Using the graphing interface](https://prometheus.io/docs/introduction/getting_started/#using-the-graphing-interface)
* [Starting up some sample targets](https://prometheus.io/docs/introduction/getting_started/#starting-up-some-sample-targets)
* [Configuring Prometheus to monitor the sample targets](https://prometheus.io/docs/introduction/getting_started/#configuring-prometheus-to-monitor-the-sample-targets)
* [Configure rules for aggregating scraped data into new time series](https://prometheus.io/docs/introduction/getting_started/#configure-rules-for-aggregating-scraped-data-into-new-time-series)

This guide is a "Hello World"-style tutorial which shows how to install, configure, and use Prometheus in a simple example setup. You will download and run Prometheus locally, configure it to scrape itself and an example application, and then work with queries, rules, and graphs to make use of the collected time series data.

**Downloading and running Prometheus**

[Download the latest release](https://prometheus.io/download) of Prometheus for your platform, then extract and run it:

tar xvfz prometheus-\*.tar.gz

cd prometheus-\*

Before starting Prometheus, let's configure it.

**Configuring Prometheus to monitor itself**

Prometheus collects metrics from monitored targets by scraping metrics HTTP endpoints on these targets. Since Prometheus also exposes data in the same manner about itself, it can also scrape and monitor its own health.

While a Prometheus server that collects only data about itself is not very useful in practice, it is a good starting example. Save the following basic Prometheus configuration as a file named prometheus.yml:

global:

scrape\_interval: 15s # By default, scrape targets every 15 seconds.

# Attach these labels to any time series or alerts when communicating with

# external systems (federation, remote storage, Alertmanager).

external\_labels:

monitor: 'codelab-monitor'

# 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'

# Override the global default and scrape targets from this job every 5 seconds.

scrape\_interval: 5s

static\_configs:

- targets: ['localhost:9090']

For a complete specification of configuration options, see the [configuration documentation](https://prometheus.io/docs/operating/configuration).

**Starting Prometheus**

To start Prometheus with your newly created configuration file, change to your Prometheus build directory and run:

# Start Prometheus.

# By default, Prometheus stores its database in ./data (flag -storage.local.path).

./prometheus -config.file=prometheus.yml

Prometheus should start up and it should show a status page about itself at [http://localhost:9090](http://localhost:9090/). Give it a couple of seconds to collect data about itself from its own HTTP metrics endpoint.

You can also verify that Prometheus is serving metrics about itself by navigating to its metrics endpoint:<http://localhost:9090/metrics>

The number of OS threads executed by Prometheus is controlled by the GOMAXPROCS environment variable. As of Go 1.5 the default value is the number of cores available.

Blindly setting GOMAXPROCS to a high value can be counterproductive. See the relevant [Go FAQs](http://golang.org/doc/faq#Why_no_multi_CPU).

Note that Prometheus by default uses around 3GB in memory. If you have a smaller machine, you can tune Prometheus to use less memory. For details, see the [memory usage documentation](https://prometheus.io/docs/operating/storage/#memory-usage).

**Using the expression browser**

Let us try looking at some data that Prometheus has collected about itself. To use Prometheus's built-in expression browser, navigate to <http://localhost:9090/graph> and choose the "Console" view within the "Graph" tab.

As you can gather from <http://localhost:9090/metrics>, one metric that Prometheus exports about itself is calledprometheus\_target\_interval\_length\_seconds (the actual amount of time between target scrapes). Go ahead and enter this into the expression console:

prometheus\_target\_interval\_length\_seconds

This should return a lot of different time series (along with the latest value recorded for each), all with the metric name prometheus\_target\_interval\_length\_seconds, but with different labels. These labels designate different latency percentiles and target group intervals.

If we were only interested in the 99th percentile latencies, we could use this query to retrieve that information:

prometheus\_target\_interval\_length\_seconds{quantile="0.99"}

To count the number of returned time series, you could write:

count(prometheus\_target\_interval\_length\_seconds)

For more about the expression language, see the [expression language documentation](https://prometheus.io/docs/querying/basics/).

**Using the graphing interface**

To graph expressions, navigate to <http://localhost:9090/graph> and use the "Graph" tab.

For example, enter the following expression to graph the per-second rate of all storage chunk operations happening in the self-scraped Prometheus:

rate(prometheus\_local\_storage\_chunk\_ops\_total[1m])

Experiment with the graph range parameters and other settings.

**Starting up some sample targets**

Let us make this more interesting and start some example targets for Prometheus to scrape.

The Go client library includes an example which exports fictional RPC latencies for three services with different latency distributions.

Ensure you have the [Go compiler installed](https://golang.org/doc/install) and have a [working Go build environment](https://golang.org/doc/code.html) (with correct GOPATH) set up.

Download the Go client library for Prometheus and run three of these example processes:

# Fetch the client library code and compile example.

git clone https://github.com/prometheus/client\_golang.git

cd client\_golang/examples/random

go get -d

go build

# Start 3 example targets in separate terminals:

./random -listen-address=:8080

./random -listen-address=:8081

./random -listen-address=:8082

You should now have example targets listening on <http://localhost:8080/metrics>, <http://localhost:8081/metrics>, and <http://localhost:8082/metrics>.

**Configuring Prometheus to monitor the sample targets**

Now we will configure Prometheus to scrape these new targets. Let's group all three endpoints into one job called example-random. However, imagine that the first two endpoints are production targets, while the third one represents a canary instance. To model this in Prometheus, we can add several groups of endpoints to a single job, adding extra labels to each group of targets. In this example, we will add the group="production" label to the first group of targets, while adding group="canary" to the second.

To achieve this, add the following job definition to the scrape\_configs section in your prometheus.yml and restart your Prometheus instance:

scrape\_configs:

- job\_name: 'example-random'

# Override the global default and scrape targets from this job every 5 seconds.

scrape\_interval: 5s

static\_configs:

- targets: ['localhost:8080', 'localhost:8081']

labels:

group: 'production'

- targets: ['localhost:8082']

labels:

group: 'canary'

Go to the expression browser and verify that Prometheus now has information about time series that these example endpoints expose, such as the rpc\_durations\_seconds metric.

**Configure rules for aggregating scraped data into new time series**

Though not a problem in our example, queries that aggregate over thousands of time series can get slow when computed ad-hoc. To make this more efficient, Prometheus allows you to prerecord expressions into completely new persisted time series via configured recording rules. Let's say we are interested in recording the per-second rate of example RPCs (rpc\_durations\_seconds\_count) averaged over all instances (but preserving thejob and service dimensions) as measured over a window of 5 minutes. We could write this as:

avg(rate(rpc\_durations\_seconds\_count[5m])) by (job, service)

Try graphing this expression.

To record the time series resulting from this expression into a new metric calledjob\_service:rpc\_durations\_seconds\_count:avg\_rate5m, create a file with the following recording rule and save it as prometheus.rules:

job\_service:rpc\_durations\_seconds\_count:avg\_rate5m = avg(rate(rpc\_durations\_seconds\_count[5m])) by (job, service)

To make Prometheus pick up this new rule, add a rule\_files statement to the global configuration section in your prometheus.yml. The config should now look like this:

global:

scrape\_interval: 15s # By default, scrape targets every 15 seconds.

evaluation\_interval: 15s # Evaluate rules every 15 seconds.

# Attach these extra labels to all timeseries collected by this Prometheus instance.

external\_labels:

monitor: 'codelab-monitor'

rule\_files:

- 'prometheus.rules'

scrape\_configs:

- job\_name: 'prometheus'

# Override the global default and scrape targets from this job every 5 seconds.

scrape\_interval: 5s

static\_configs:

- targets: ['localhost:9090']

- job\_name: 'example-random'

# Override the global default and scrape targets from this job every 5 seconds.

scrape\_interval: 5s

static\_configs:

- targets: ['localhost:8080', 'localhost:8081']

labels:

group: 'production'

- targets: ['localhost:8082']

labels:

group: 'canary'

Restart Prometheus with the new configuration and verify that a new time series with the metric namejob\_service:rpc\_durations\_seconds\_count:avg\_rate5m is now available by querying it through the expression browser or graphing it.

## Comparison to alternatives | Prometheus

https://prometheus.io/docs/introduction/comparison/

**COMPARISON TO ALTERNATIVES**

* [Prometheus vs. Graphite](https://prometheus.io/docs/introduction/comparison/#prometheus-vs.-graphite)
  + [Scope](https://prometheus.io/docs/introduction/comparison/#scope)
  + [Data model](https://prometheus.io/docs/introduction/comparison/#data-model)
  + [Storage](https://prometheus.io/docs/introduction/comparison/#storage)
  + [Summary](https://prometheus.io/docs/introduction/comparison/#summary)
* [Prometheus vs. InfluxDB](https://prometheus.io/docs/introduction/comparison/#prometheus-vs.-influxdb)
  + [Scope](https://prometheus.io/docs/introduction/comparison/#scope)
  + [Data model / storage](https://prometheus.io/docs/introduction/comparison/#data-model-/-storage)
  + [Architecture](https://prometheus.io/docs/introduction/comparison/#architecture)
  + [Summary](https://prometheus.io/docs/introduction/comparison/#summary)
* [Prometheus vs. OpenTSDB](https://prometheus.io/docs/introduction/comparison/#prometheus-vs.-opentsdb)
  + [Scope](https://prometheus.io/docs/introduction/comparison/#scope)
  + [Data model](https://prometheus.io/docs/introduction/comparison/#data-model)
  + [Storage](https://prometheus.io/docs/introduction/comparison/#storage)
  + [Summary](https://prometheus.io/docs/introduction/comparison/#summary)
* [Prometheus vs. Nagios](https://prometheus.io/docs/introduction/comparison/#prometheus-vs.-nagios)
  + [Scope](https://prometheus.io/docs/introduction/comparison/#scope)
  + [Data model](https://prometheus.io/docs/introduction/comparison/#data-model)
  + [Storage](https://prometheus.io/docs/introduction/comparison/#storage)
  + [Architecture](https://prometheus.io/docs/introduction/comparison/#architecture)
  + [Summary](https://prometheus.io/docs/introduction/comparison/#summary)
* [Prometheus vs. Sensu](https://prometheus.io/docs/introduction/comparison/#prometheus-vs.-sensu)
  + [Scope](https://prometheus.io/docs/introduction/comparison/#scope)
  + [Data model](https://prometheus.io/docs/introduction/comparison/#data-model)
  + [Storage](https://prometheus.io/docs/introduction/comparison/#storage)
  + [Architecture](https://prometheus.io/docs/introduction/comparison/#architecture)
  + [Summary](https://prometheus.io/docs/introduction/comparison/#summary)

### Prometheus vs. Graphite

Scope

[Graphite](http://graphite.readthedocs.org/en/latest/) focuses on being a passive time series database with a query language and graphing features. Any other concerns are addressed by external components.

Prometheus is a full monitoring and trending system that includes built-in and active scraping, storing, querying, graphing, and alerting based on time series data. It has knowledge about what the world should look like (which endpoints should exist, what time series patterns mean trouble, etc.), and actively tries to find faults.

Data model

Graphite stores numeric samples for named time series, much like Prometheus does. However, Prometheus's metadata model is richer: while Graphite metric names consist of dot-separated components which implicitly encode dimensions, Prometheus encodes dimensions explicitly as key-value pairs (labels) attached to a metric name. This allows easy filtering, grouping, and matching by these labels via in the query language.

Further, especially when Graphite is used in combination with[StatsD](https://github.com/etsy/statsd/), it is common to store only aggregated data over all monitored instances, rather than preserving the instance as a dimension and being able to drill down into individual problematic ones.

As an example, storing the number of HTTP requests to API servers with the response code 500 and the method POST to the /tracks endpoint would commonly be encoded like this in Graphite/StatsD:

stats.api-server.tracks.post.500 -> 93

In Prometheus the same data could be encoded like this (assuming three api-server instances):

api\_server\_http\_requests\_total{method="POST",handler="/tracks",status="500",instance="<sample1>"} -> 34

api\_server\_http\_requests\_total{method="POST",handler="/tracks",status="500",instance="<sample2>"} -> 28

api\_server\_http\_requests\_total{method="POST",handler="/tracks",status="500",instance="<sample3>"} -> 31

Storage

Graphite stores time series data on local disk in the [Whisper](http://graphite.readthedocs.org/en/latest/whisper.html) format, an RRD-style database that expects samples to arrive at regular intervals. Every time series is stored in a separate file, and new samples overwrite old ones after a certain amount of time.

Prometheus also creates one local file per time series, but allows storing samples at arbitrary intervals as scrapes or rule evaluations occur. Since new samples are simply appended, old data may be kept arbitrarily long. Prometheus also works well for many short-lived, frequently changing sets of time series.

Summary

Prometheus offers a richer data model and query language, in addition to being easier to run and integrate into your environment. If you want a clustered solution that can hold historical data long term, Graphite may be a better choice.

### Prometheus vs. InfluxDB

[InfluxDB](https://influxdata.com/) is an open-source time series database, with a commercial option for scaling and clustering. The InfluxDB project was released almost a year after Prometheus development began, so we were unable to consider it as an alternative at the time. Still, there are significant differences between Prometheus and InfluxDB, and both systems are geared towards slightly different use cases.

Scope

For a fair comparison, we must also consider [Kapacitor](https://github.com/influxdata/kapacitor) together with InfluxDB, as in combination they address the same problem space as Prometheus and the Alertmanager.

The same scope differences as in the case of [Graphite](https://prometheus.io/docs/introduction/comparison/#prometheus-vs-graphite) apply here for InfluxDB itself. In addition InfluxDB offers continuous queries, which are equivalent to Prometheus recording rules.

Kapacitor’s scope is a combination of Prometheus recording rules, alerting rules, and the Alertmanager's notification functionality. Prometheus offers [a more powerful query language for graphing and alerting](https://www.robustperception.io/translating-between-monitoring-languages/). The Prometheus Alertmanager additionally offers grouping, deduplication and silencing functionality.

Data model / storage

Like Prometheus, the InfluxDB data model has key-value pairs as labels, which are called tags. In addition InfluxDB has a second level of labels called fields, which are more limited in use. InfluxDB supports timestamps with up to nanosecond resolution, and float64, int64, bool, and string data types. Prometheus by contrast supports the float64 data type with limited support for strings, and millisecond resolution timestamps.

InfluxDB uses a variant of a [log-structured merge tree for storage with a write ahead log](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/), sharded by time. This is much more suitable to event logging than Prometheus's append-only file per time series approach.

[Logs and Metrics and Graphs, Oh My!](https://blog.raintank.io/logs-and-metrics-and-graphs-oh-my/) describes the difference between event logging and metrics recording.

Architecture

Prometheus servers run independently of each other and only rely on their local storage for their core functionality: scraping, rule processing, and alerting. The open source version of InfluxDB is similar.

The commercial InfluxDB offering is by design a distributed storage cluster with storage and queries being handled by many nodes at once.

This means that the commercial InfluxDB will be easier to scale horizontally, but it also means that you have to manage the complexity of a distributed storage system from the beginning. Prometheus will be simpler to run, but at some point you will need to shard servers explicitly along scalability boundaries like products, services, datacenters, or similar aspects. Independent servers (which can be run redundantly in parallel) may also give you better reliability and failure isolation.

Kapacitor currently has no [built-in distributed/redundant options](https://github.com/influxdata/kapacitor/issues/277) for rules, alerting or notifications. Prometheus and the Alertmanager by contrast offer a redundant option via running redundant replicas of Prometheus and using the Alertmanager's [High Availability](https://github.com/prometheus/alertmanager#high-availability) mode. In addition, Kapacitor can be scaled via manual sharding by the user, similar to Prometheus itself.

Summary

There are many similarities between the systems. Both have labels (called tags in InfluxDB) to efficiently support multi-dimensional metrics. Both use basically the same data compression algorithms. Both have extensive integrations, including with each other. Both have hooks allowing you to extend them further, such as analysing data in statistical tools or performing automated actions.

Where InfluxDB is better:

* If you're doing event logging.
* Commercial option offers clustering for InfluxDB, which is also better for long term data storage.
* Eventually consistent view of data between replicas.

Where Prometheus is better:

* If you're primarily doing metrics.
* More powerful query language, alerting, and notification functionality.
* Higher availability and uptime for graphing and alerting.

InfluxDB is maintained by a single commercial company following the open-core model, offering premium features like closed-source clustering, hosting and support. Prometheus is a [fully open source and independent project](https://prometheus.io/community/), maintained by a number of companies and individuals, some of whom also offer commercial services and support.

### Prometheus vs. OpenTSDB

[OpenTSDB](http://opentsdb.net/) is a distributed time series database based on [Hadoop](http://hadoop.apache.org/) and [HBase](http://hbase.apache.org/).

Scope

The same scope differences as in the case of [Graphite](https://prometheus.io/docs/introduction/comparison/#prometheus-vs-graphite) apply here.

Data model

OpenTSDB's data model is almost identical to Prometheus's: time series are identified by a set of arbitrary key-value pairs (OpenTSDB "tags" are Prometheus "labels"). All data for a metric is [stored together](http://opentsdb.net/docs/build/html/user_guide/writing/index.html#time-series-cardinality), limiting the cardinality of metrics. There are minor differences though, such as that Prometheus allows arbitrary characters in label values, while OpenTSDB is more restrictive. OpenTSDB is also lacking a full query language, only allowing simple aggregation and math via its API.

Storage

[OpenTSDB](http://opentsdb.net/)'s storage is implemented on top of [Hadoop](http://hadoop.apache.org/) and [HBase](http://hbase.apache.org/). This means that it is easy to scale OpenTSDB horizontally, but you have to accept the overall complexity of running a Hadoop/HBase cluster from the beginning.

Prometheus will be simpler to run initially, but will require explicit sharding once the capacity of a single node is exceeded.

Summary

Prometheus offers a much richer query language, can handle higher cardinality metrics and forms part of a complete monitoring system. If you're already running Hadoop and value long term storage over these benefits, OpenTSDB is a good choice.

### Prometheus vs. Nagios

[Nagios](https://www.nagios.org/) is a monitoring system that originated in the 90s as NetSaint.

Scope

Nagios is primarily about alerting based on the exit codes of scripts. These are called “checks”. There is silencing of individual alerts, however no grouping, routing or deduplication.

There are a variety of plugins. For example, piping the few kilobytes of perfData plugins are allowed to return [to a time series database such as Graphite](https://github.com/shawn-sterling/graphios) or using NRPE to [run checks on remote machines](https://exchange.nagios.org/directory/Addons/Monitoring-Agents/NRPE--2D-Nagios-Remote-Plugin-Executor/details).

Data model

Nagios is host-based. Each host can have one or more services, which has one check.

There is no notion of labels or a query language.

Storage

Nagios has no storage per-se, beyond the current check state. There are plugins which can store data such as[for visualisation](https://docs.pnp4nagios.org/).

Architecture

Nagios servers are standalone. All configuration of checks is via file.

Summary

Nagios is suitable for basic monitoring of small and/or static systems where blackbox probing is sufficient.

If you want to do whitebox monitoring, or have a dynamic or cloud based environment then Prometheus is a good choice.

### Prometheus vs. Sensu

[Sensu](https://sensuapp.org/) is broadly speaking a more modern Nagios.

Scope

The same general scope differences as in the case of [Nagios](https://prometheus.io/docs/introduction/comparison/#prometheus-vs-nagios) apply here.

The primary difference is that Sensu clients [register themselves](https://sensuapp.org/docs/0.27/reference/clients.html#what-is-a-sensu-client), and can determine the checks to run either from central or local configuration. Sensu does not have a limit on the amount of perfData.

There is also a [client socket](https://sensuapp.org/docs/0.27/reference/clients.html#what-is-the-sensu-client-socket) permitting arbitrary check results to be pushed into Sensu.

Data model

Sensu has the same rough data model as [Nagios](https://prometheus.io/docs/introduction/comparison/#prometheus-vs-nagios).

Storage

Sensu has storage in Redis called stashes. These are used primarily for storing silences. It also stores all the clients that have registered with it.

Architecture

Sensu has a [number of components](https://sensuapp.org/docs/0.27/overview/architecture.html). It uses RabbitMQ as a transport, Redis for current state, and a separate Server for processing.

Both RabbitMQ and Redis can be clustered. Multiple copies of the server can be run for scaling and redundancy.

Summary

If you have an existing Nagios setup that you wish to scale as-is or taking advantage of the registration feature of Sensu, then Sensu is a good choice.

If you want to do whitebox monitoring, or have a very dynamic or cloud based environment, then Prometheus is a good choice.

## FAQ | Prometheus

https://prometheus.io/docs/introduction/faq/

* [General](https://prometheus.io/docs/introduction/faq/#general)
  + [What is Prometheus?](https://prometheus.io/docs/introduction/faq/#what-is-prometheus?)
  + [How does Prometheus compare against other monitoring systems?](https://prometheus.io/docs/introduction/faq/#how-does-prometheus-compare-against-other-monitoring-systems?)
  + [What dependencies does Prometheus have?](https://prometheus.io/docs/introduction/faq/#what-dependencies-does-prometheus-have?)
  + [Can Prometheus be made highly available?](https://prometheus.io/docs/introduction/faq/#can-prometheus-be-made-highly-available?)
  + [I was told Prometheus “doesn't scale”.](https://prometheus.io/docs/introduction/faq/#i-was-told-prometheus-%E2%80%9Cdoesn't-scale%E2%80%9D.)
  + [What language is Prometheus written in?](https://prometheus.io/docs/introduction/faq/#what-language-is-prometheus-written-in?)
  + [How stable are Prometheus features, storage formats, and APIs?](https://prometheus.io/docs/introduction/faq/#how-stable-are-prometheus-features,-storage-formats,-and-apis?)
  + [Why do you pull rather than push?](https://prometheus.io/docs/introduction/faq/#why-do-you-pull-rather-than-push?)
  + [How to feed logs into Prometheus?](https://prometheus.io/docs/introduction/faq/#how-to-feed-logs-into-prometheus?)
  + [Who wrote Prometheus?](https://prometheus.io/docs/introduction/faq/#who-wrote-prometheus?)
  + [What license is Prometheus released under?](https://prometheus.io/docs/introduction/faq/#what-license-is-prometheus-released-under?)
  + [What is the plural of Prometheus?](https://prometheus.io/docs/introduction/faq/#what-is-the-plural-of-prometheus?)
  + [Can I reload Prometheus's configuration?](https://prometheus.io/docs/introduction/faq/#can-i-reload-prometheus's-configuration?)
  + [Can I send alerts?](https://prometheus.io/docs/introduction/faq/#can-i-send-alerts?)
  + [Can I create dashboards?](https://prometheus.io/docs/introduction/faq/#can-i-create-dashboards?)
  + [Can I change the timezone? Why is everything in UTC?](https://prometheus.io/docs/introduction/faq/#can-i-change-the-timezone?-why-is-everything-in-utc?)
* [Instrumentation](https://prometheus.io/docs/introduction/faq/#instrumentation)
  + [Which languages have instrumentation libraries?](https://prometheus.io/docs/introduction/faq/#which-languages-have-instrumentation-libraries?)
  + [Can I monitor machines?](https://prometheus.io/docs/introduction/faq/#can-i-monitor-machines?)
  + [Can I monitor network devices?](https://prometheus.io/docs/introduction/faq/#can-i-monitor-network-devices?)
  + [Can I monitor batch jobs?](https://prometheus.io/docs/introduction/faq/#can-i-monitor-batch-jobs?)
  + [What applications can Prometheus monitor out of the box?](https://prometheus.io/docs/introduction/faq/#what-applications-can-prometheus-monitor-out-of-the-box?)
  + [Can I monitor JVM applications via JMX?](https://prometheus.io/docs/introduction/faq/#can-i-monitor-jvm-applications-via-jmx?)
  + [What is the performance impact of instrumentation?](https://prometheus.io/docs/introduction/faq/#what-is-the-performance-impact-of-instrumentation?)
* [Troubleshooting](https://prometheus.io/docs/introduction/faq/#troubleshooting)
  + [My Prometheus server takes a long time to start up and spams the log with copious information about crash recovery.](https://prometheus.io/docs/introduction/faq/#my-prometheus-server-takes-a-long-time-to-start-up-and-spams-the-log-with-copious-information-about-crash-recovery.)
  + [My Prometheus server runs out of memory.](https://prometheus.io/docs/introduction/faq/#my-prometheus-server-runs-out-of-memory.)
  + [My Prometheus server reports to be in “rushed mode” or that “storage needs throttling”.](https://prometheus.io/docs/introduction/faq/#my-prometheus-server-reports-to-be-in-%E2%80%9Crushed-mode%E2%80%9D-or-that-%E2%80%9Cstorage-needs-throttling%E2%80%9D.)
* [Implementation](https://prometheus.io/docs/introduction/faq/#implementation)
  + [Why are all sample values 64-bit floats? I want integers.](https://prometheus.io/docs/introduction/faq/#why-are-all-sample-values-64-bit-floats?-i-want-integers.)
  + [Why does Prometheus use a custom storage backend rather than [some other storage method]? Isn't the "one file per time series" approach killing performance?](https://prometheus.io/docs/introduction/faq/#why-does-prometheus-use-a-custom-storage-backend-rather-than-%5Bsome-other-storage-method%5D?-isn't-the-)
  + [Why don't the Prometheus server components support TLS or authentication? Can I add those?](https://prometheus.io/docs/introduction/faq/#why-don't-the-prometheus-server-components-support-tls-or-authentication?-can-i-add-those?)

**General**

What is Prometheus?

Prometheus is an open-source systems monitoring and alerting toolkit with an active ecosystem. See the[overview](https://prometheus.io/docs/introduction/overview/).

How does Prometheus compare against other monitoring systems?

See the [comparison](https://prometheus.io/docs/introduction/comparison/) page.

What dependencies does Prometheus have?

The main Prometheus server runs standalone and has no external dependencies.

Can Prometheus be made highly available?

Yes, run identical Prometheus servers on two or more separate machines. Identical alerts will be deduplicated by the [Alertmanager](https://github.com/prometheus/alertmanager).

For [high availability of the Alertmanager](https://github.com/prometheus/alertmanager#high-availability), you can run multiple instances in a [Mesh cluster](https://github.com/weaveworks/mesh) and configure the Prometheus servers to send notifications to each of them.

I was told Prometheus “doesn't scale”.

There are in fact various ways to scale and federate Prometheus. Read [Scaling and Federating Prometheus](https://www.robustperception.io/scaling-and-federating-prometheus/) on the Robust Perception blog to get started.

What language is Prometheus written in?

Most Prometheus components are written in Go. Some are also written in Java, Python, and Ruby.

How stable are Prometheus features, storage formats, and APIs?

All repositories in the Prometheus GitHub organization that have reached version 1.0.0 broadly follow [semantic versioning](http://semver.org/). Breaking changes are indicated by increments of the major version. Exceptions are possible for experimental components, which are clearly marked as such in announcements.

Even repositories that have not yet reached version 1.0.0 are in general quite stable. We aim for a proper release process and an eventual 1.0.0 release for each repository. In any case, breaking changes will be pointed out in release notes (marked by [CHANGE]) or communicated clearly for components that do not have formal releases yet.

Why do you pull rather than push?

Pulling over HTTP offers a number of advantages:

* You can run your monitoring on your laptop when developing changes.
* You can more easily tell if a target is down.
* You can manually go to a target and inspect its health with a web browser.

Overall we believe that pulling is slightly better than pushing, but it should not be considered a major point when considering a monitoring system.

The [Push vs Pull for Monitoring](http://www.boxever.com/push-vs-pull-for-monitoring) blog post by Brian Brazil goes into more detail.

For cases where you must push, we offer the [Pushgateway](https://prometheus.io/docs/instrumenting/pushing/).

How to feed logs into Prometheus?

Short answer: Don't! Use something like the ELK stack instead.

Longer answer: Prometheus is a system to collect and process metrics, not an event logging system. The Raintank blog post [Logs and Metrics and Graphs, Oh My!](https://blog.raintank.io/logs-and-metrics-and-graphs-oh-my/) provides more details about the differences between logs and metrics.

If you want to extract Prometheus metrics from application logs, Google's [mtail](https://github.com/google/mtail) might be helpful.

Who wrote Prometheus?

Prometheus was initially started privately by [Matt T. Proud](http://www.matttproud.com/) and [Julius Volz](http://juliusv.com/). The majority of its initial development was sponsored by [SoundCloud](https://soundcloud.com/).

It's now maintained and extended by a wide range of companies and individuals.

What license is Prometheus released under?

Prometheus is released under the [Apache 2.0](https://github.com/prometheus/prometheus/blob/master/LICENSE) license.

What is the plural of Prometheus?

After [extensive research](https://youtu.be/B_CDeYrqxjQ), it has been determined that the correct plural of 'Prometheus' is 'Prometheis'.

Can I reload Prometheus's configuration?

Yes, sending SIGHUP to the Prometheus process or an HTTP POST request to the /-/reload endpoint will reload and apply the configuration file. The various components attempt to handle failing changes gracefully.

Can I send alerts?

Yes, with the [Alertmanager](https://github.com/prometheus/alertmanager).

Currently, the following external systems are supported:

* Email
* Generic Webhooks
* [HipChat](https://www.hipchat.com/)
* [OpsGenie](https://www.opsgenie.com/)
* [PagerDuty](http://www.pagerduty.com/)
* [Pushover](https://pushover.net/)
* [Slack](https://slack.com/)

Can I create dashboards?

Yes, we recommend [Grafana](https://prometheus.io/docs/visualization/grafana/) for production usage. There are also [Console templates](https://prometheus.io/docs/visualization/consoles/).

Can I change the timezone? Why is everything in UTC?

To avoid any kind of timezone confusion, especially when the so-called daylight saving time is involved, we decided to exclusively use Unix time internally and UTC for display purposes in all components of Prometheus. A carefully done timezone selection could be introduced into the UI. Contributions are welcome. See [issue #500](https://github.com/prometheus/prometheus/issues/500)for the current state of this effort.

**Instrumentation**

Which languages have instrumentation libraries?

There are a number of client libraries for instrumenting your services with Prometheus metrics. See the [client libraries](https://prometheus.io/docs/instrumenting/clientlibs/) documentation for details.

If you are interested in contributing a client library for a new language, see the [exposition formats](https://prometheus.io/docs/instrumenting/exposition_formats/).

Can I monitor machines?

Yes, the [Node Exporter](https://github.com/prometheus/node_exporter) exposes an extensive set of machine-level metrics on Linux and other Unix systems such as CPU usage, memory, disk utilization, filesystem fullness and network bandwidth.

Can I monitor network devices?

Yes, the [SNMP Exporter](https://github.com/prometheus/snmp_exporter) allows monitoring of devices that support SNMP.

Can I monitor batch jobs?

Yes, using the [Pushgateway](https://prometheus.io/docs/instrumenting/pushing/). See also the [best practices](https://prometheus.io/docs/practices/instrumentation/#batch-jobs) for monitoring batch jobs.

What applications can Prometheus monitor out of the box?

See [the list of exporters and integrations](https://prometheus.io/docs/instrumenting/exporters/).

Can I monitor JVM applications via JMX?

Yes, for applications that you cannot instrument directly with the Java client you can use the [JMX Exporter](https://github.com/prometheus/jmx_exporter) either standalone or as a Java Agent.

What is the performance impact of instrumentation?

Performance across client libraries and languages may vary. For Java, [benchmarks](https://github.com/prometheus/client_java/blob/master/benchmark/README.md) indicate that incrementing a counter/gauge with the Java client will take 12-17ns, depending on contention. This is negligible for all but the most latency-critical code.

**Troubleshooting**

My Prometheus server takes a long time to start up and spams the log with copious information about crash recovery.

You are suffering from an unclean shutdown. Prometheus has to shut down cleanly after a SIGTERM, which might take a while for heavily used servers. If the server crashes or is killed hard (e.g. OOM kill by the kernel or your runlevel system got impatient while waiting for Prometheus to shutdown), a crash recovery has to be performed, which should take less than a minute under normal circumstances, but can take quite long under certain circumstances. See [crash recovery](https://prometheus.io/docs/operating/storage/#crash-recovery) for details.

My Prometheus server runs out of memory.

See [the section about memory usage](https://prometheus.io/docs/operating/storage/#memory-usage) to configure Prometheus for the amount of memory you have available.

My Prometheus server reports to be in “rushed mode” or that “storage needs throttling”.

Your storage is under heavy load. Read [the section about configuring the local storage](https://prometheus.io/docs/operating/storage/) to find out how you can tweak settings for better performance.

**Implementation**

Why are all sample values 64-bit floats? I want integers.

We restrained ourselves to 64-bit floats to simplify the design. The [IEEE 754 double-precision binary floating-point format](http://en.wikipedia.org/wiki/Double-precision_floating-point_format) supports integer precision for values up to 253. Supporting native 64 bit integers would (only) help if you need integer precision above 253 but below 263. In principle, support for different sample value types (including some kind of big integer, supporting even more than 64 bit) could be implemented, but it is not a priority right now. Note that a counter, even if incremented one million times per second, will only run into precision issues after over 285 years.

Why does Prometheus use a custom storage backend rather than [some other storage method]? Isn't the "one file per time series" approach killing performance?

Initially, Prometheus ran completely on LevelDB, but to achieve better performance, we had to change the storage for bulk sample data. We evaluated many storage backends that were available at the time, without getting satisfactory results. So we implemented exactly the parts we needed, while keeping LevelDB for indexes and making heavy use of file system capabilities. Obviously, we could not evaluate every single storage backend out there, and storage backends have evolved meanwhile. However, the performance of the solution implemented now is satisfactory for most use-cases. Our most important requirements are an acceptable query speed for common queries and a sustainable ingestion rate of hundreds of thousands of samples per second. The latter depends on many parameters, like the compressibility of the sample data, the number of time series the samples belong to, the retention policy, and even more subtle aspects like how full your SSD is. If you want to know all the details, read [this document with detailed benchmark results](https://docs.google.com/document/d/1lRKBaz9oXI5nwFZfvSbPhpwzUbUr3-9qryQGG1C6ULk/edit?usp=sharing). The highlights:

* On a typical bare-metal server with 64GiB RAM, 32 CPU cores, and SSD, Prometheus sustained an ingestion rate of 900k samples per second, belonging to 1M time series, scraped from 720 targets.
* On a server with HDD and 128GiB RAM, Prometheus sustained an ingestion rate of 250k samples per second, belonging to 1M time series, scraped from 720 targets.

Running out of inodes is unlikely in a usual set-up. However, if you have a lot of short-lived time series, or you have configured your file system with an unusual low amount of inodes, you might run into inode depletion. Also, if you want to delete Prometheus's storage directory, you will notice that some file systems are very slow when deleting a large number of files.

Why don't the Prometheus server components support TLS or authentication? Can I add those?

While TLS and authentication are frequently requested features, we have intentionally not implemented them in any of Prometheus's server-side components. There are so many different options and parameters for both (10+ options for TLS alone) that we have decided to focus on building the best monitoring system possible rather than supporting fully generic TLS and authentication solutions in every server component.

If you need TLS or authentication, we recommend putting a reverse proxy in front of Prometheus. See for example [Adding Basic Auth to Prometheus with Nginx](https://www.robustperception.io/adding-basic-auth-to-prometheus-with-nginx/).

Note that this applies only to inbound connections. Prometheus does support [scraping TLS- and auth-enabled targets](https://prometheus.io/docs/operating/configuration/#%3Cscrape_config%3E), and other Prometheus components that create outbound connections have similar support.

## Glossary | Prometheus

https://prometheus.io/docs/introduction/glossary/

**GLOSSARY**

* [Alert](https://prometheus.io/docs/introduction/glossary/#alert)
* [Alertmanager](https://prometheus.io/docs/introduction/glossary/#alertmanager)
* [Bridge](https://prometheus.io/docs/introduction/glossary/#bridge)
* [Client library](https://prometheus.io/docs/introduction/glossary/#client-library)
* [Collector](https://prometheus.io/docs/introduction/glossary/#collector)
* [Direct instrumentation](https://prometheus.io/docs/introduction/glossary/#direct-instrumentation)
* [Exporter](https://prometheus.io/docs/introduction/glossary/#exporter)
* [Notification](https://prometheus.io/docs/introduction/glossary/#notification)
* [Promdash](https://prometheus.io/docs/introduction/glossary/#promdash)
* [Prometheus](https://prometheus.io/docs/introduction/glossary/#prometheus)
* [PromQL](https://prometheus.io/docs/introduction/glossary/#promql)
* [Pushgateway](https://prometheus.io/docs/introduction/glossary/#pushgateway)
* [Remote Read](https://prometheus.io/docs/introduction/glossary/#remote-read)
* [Remote Read Adapter](https://prometheus.io/docs/introduction/glossary/#remote-read-adapter)
* [Remote Read Endpoint](https://prometheus.io/docs/introduction/glossary/#remote-read-endpoint)
* [Remote Write](https://prometheus.io/docs/introduction/glossary/#remote-write)
* [Remote Write Adapter](https://prometheus.io/docs/introduction/glossary/#remote-write-adapter)
* [Remote Write Endpoint](https://prometheus.io/docs/introduction/glossary/#remote-write-endpoint)
* [Silence](https://prometheus.io/docs/introduction/glossary/#silence)
* [Target](https://prometheus.io/docs/introduction/glossary/#target)

Alert

An alert is the outcome of an alerting rule in Prometheus that is actively firing. Alerts are sent from Prometheus to the Alertmanager.

Alertmanager

The [Alertmanager](https://prometheus.io/docs/alerting/overview/) takes in alerts, aggregates them into groups, de-duplicates, applies silences, throttles, and then sends out notifications to email, Pagerduty, Slack etc.

Bridge

A bridge is a component that takes samples from a client library and exposes them to a non-Prometheus monitoring system. For example the Python client can export metrics to Graphite.

Client library

A client library is a library in some language (e.g. Go, Java, Python, Ruby) that makes it easy to directly instrument your code, write custom collectors to pull metrics from other systems and expose the metrics to Prometheus.

Collector

A collector is a part of an exporter that represents a set of metrics. It may be a single metric as part of direct instrumentation, or many metrics if it is pulling metrics from another system.

Direct instrumentation

Direct instrumentation is when instrumentation is added inline as part the source code of a program.

Exporter

An exporter is a binary that exposes Prometheus metrics, commonly by converting metrics that are exposed in a non-Prometheus format into a format Prometheus supports.

Notification

A notification represents a group or one of more alerts, and is sent by the Alertmanager to email, Pagerduty, Slack etc.

Promdash

Promdash was a native dashboard builder for Prometheus. It has been replaced by [Grafana](https://prometheus.io/docs/visualization/grafana/).

Prometheus

Prometheus usually refers to the core binary of the Prometheus system. It may also refer to the Prometheus monitoring system as a whole.

PromQL

[PromQL](https://prometheus.io/docs/querying/basics/) is the Prometheus Query Language. It allows for a wide range of operations including aggregation, slicing and dicing, prediction and joins.

Pushgateway

The [Pushgateway](https://prometheus.io/docs/instrumenting/pushing/) persists the most recent push of metrics from batch jobs. This allows Prometheus to scrape their metrics after they have terminated.

Remote Read

Remote read is a Prometheus feature that allows transparent reading of time series from other systems (such as long term storage) as part of queries.

Remote Read Adapter

Not all systems directly support remote read. A remote read adapter sits between Prometheus and another system, converting time series requests and responses between them.

Remote Read Endpoint

A remote read endpoint is what Prometheus talks to when doing a remote read.

Remote Write

Remote write is a Prometheus feature that allows sending ingested samples on the fly to other systems, such as long term storage.

Remote Write Adapter

Not all systems directly support remote write. A remote write adapter sits between Prometheus and another system, converting the samples in the remote write into a format the other system can understand.

Remote Write Endpoint

A remote write endpoint is what Prometheus talks to when doing a remote write.

Silence

A silence in the Alertmanager prevents alerts with labels matching the silence from being included in notifications.

Target

One application, server or endpoint that Prometheus is scraping.

# Concepts

## Data model | Prometheus

https://prometheus.io/docs/concepts/data\_model/

Prometheus fundamentally stores all data as [time series](http://en.wikipedia.org/wiki/Time_series): streams of timestamped values belonging to the same metric and the same set of labeled dimensions. Besides stored time series, Prometheus may generate temporary derived time series as the result of queries.

**Metric names and labels**

Every time series is uniquely identified by its metric name and a set of key-value pairs, also known as labels.

The metric name specifies the general feature of a system that is measured (e.g. http\_requests\_total - the total number of HTTP requests received). It may contain ASCII letters and digits, as well as underscores and colons. It must match the regex [a-zA-Z\_:][a-zA-Z0-9\_:]\*.

Labels enable Prometheus's dimensional data model: any given combination of labels for the same metric name identifies a particular dimensional instantiation of that metric (for example: all HTTP requests that used the method POST to the /api/tracks handler). The query language allows filtering and aggregation based on these dimensions. Changing any label value, including adding or removing a label, will create a new time series.

Label names may contain ASCII letters, numbers, as well as underscores. They must match the regex [a-zA-Z\_][a-zA-Z0-9\_]\*. Label names beginning with \_\_ are reserved for internal use.

Label values may contain any Unicode characters.

See also the [best practices for naming metrics and labels](https://prometheus.io/docs/practices/naming/).

**Samples**

Samples form the actual time series data. Each sample consists of:

* a float64 value
* a millisecond-precision timestamp

**Notation**

Given a metric name and a set of labels, time series are frequently identified using this notation:

<metric name>{<label name>=<label value>, ...}

For example, a time series with the metric name api\_http\_requests\_total and the labels method="POST" and handler="/messages" could be written like this:

api\_http\_requests\_total{method="POST", handler="/messages"}

This is the same notation that [OpenTSDB](http://opentsdb.net/) uses.

## Metric types | Prometheus

https://prometheus.io/docs/concepts/metric\_types/

**METRIC TYPES**

* [Counter](https://prometheus.io/docs/concepts/metric_types/#counter)
* [Gauge](https://prometheus.io/docs/concepts/metric_types/#gauge)
* [Histogram](https://prometheus.io/docs/concepts/metric_types/#histogram)
* [Summary](https://prometheus.io/docs/concepts/metric_types/#summary)

The Prometheus client libraries offer four core metric types. These are currently only differentiated in the client libraries (to enable APIs tailored to the usage of the specific types) and in the wire protocol. The Prometheus server does not yet make use of the type information and flattens all data into untyped time series. This may change in the future.

**Counter**

A counter is a cumulative metric that represents a single numerical value that only ever goes up. A counter is typically used to count requests served, tasks completed, errors occurred, etc. Counters should not be used to expose current counts of items whose number can also go down, e.g. the number of currently running goroutines. Use gauges for this use case.

Client library usage documentation for counters:

* [Go](http://godoc.org/github.com/prometheus/client_golang/prometheus#Counter)
* [Java](https://github.com/prometheus/client_java/blob/master/simpleclient/src/main/java/io/prometheus/client/Counter.java)
* [Python](https://github.com/prometheus/client_python#counter)
* [Ruby](https://github.com/prometheus/client_ruby#counter)

**Gauge**

A gauge is a metric that represents a single numerical value that can arbitrarily go up and down.

Gauges are typically used for measured values like temperatures or current memory usage, but also "counts" that can go up and down, like the number of running goroutines.

Client library usage documentation for gauges:

* [Go](http://godoc.org/github.com/prometheus/client_golang/prometheus#Gauge)
* [Java](https://github.com/prometheus/client_java/blob/master/simpleclient/src/main/java/io/prometheus/client/Gauge.java)
* [Python](https://github.com/prometheus/client_python#gauge)
* [Ruby](https://github.com/prometheus/client_ruby#gauge)

**Histogram**

A histogram samples observations (usually things like request durations or response sizes) and counts them in configurable buckets. It also provides a sum of all observed values.

A histogram with a base metric name of <basename> exposes multiple time series during a scrape:

* cumulative counters for the observation buckets, exposed as <basename>\_bucket{le="<upper inclusive bound>"}
* the **total sum** of all observed values, exposed as <basename>\_sum
* the **count** of events that have been observed, exposed as <basename>\_count (identical to<basename>\_bucket{le="+Inf"} above)

Use the [histogram\_quantile() function](https://prometheus.io/docs/querying/functions/#histogram_quantile) to calculate quantiles from histograms or even aggregations of histograms. A histogram is also suitable to calculate an [Apdex score](http://en.wikipedia.org/wiki/Apdex). When operating on buckets, remember that the histogram is [cumulative](https://en.wikipedia.org/wiki/Histogram#Cumulative_histogram). See [histograms and summaries](https://prometheus.io/docs/practices/histograms) for details of histogram usage and differences to [summaries](https://prometheus.io/docs/concepts/metric_types/#summary).

Client library usage documentation for histograms:

* [Go](http://godoc.org/github.com/prometheus/client_golang/prometheus#Histogram)
* [Java](https://github.com/prometheus/client_java/blob/master/simpleclient/src/main/java/io/prometheus/client/Histogram.java)
* [Python](https://github.com/prometheus/client_python#histogram)
* [Ruby](https://github.com/prometheus/client_ruby#histogram)

**Summary**

Similar to a histogram, a summary samples observations (usually things like request durations and response sizes). While it also provides a total count of observations and a sum of all observed values, it calculates configurable quantiles over a sliding time window.

A summary with a base metric name of <basename> exposes multiple time series during a scrape:

* streaming **φ-quantiles** (0 ≤ φ ≤ 1) of observed events, exposed as <basename>{quantile="<φ>"}
* the **total sum** of all observed values, exposed as <basename>\_sum
* the **count** of events that have been observed, exposed as <basename>\_count

See [histograms and summaries](https://prometheus.io/docs/practices/histograms) for detailed explanations of φ-quantiles, summary usage, and differences to[histograms](https://prometheus.io/docs/concepts/metric_types/#histogram).

Client library usage documentation for summaries:

* [Go](http://godoc.org/github.com/prometheus/client_golang/prometheus#Summary)
* [Java](https://github.com/prometheus/client_java/blob/master/simpleclient/src/main/java/io/prometheus/client/Summary.java)
* [Python](https://github.com/prometheus/client_python#summary)
* [Ruby](https://github.com/prometheus/client_ruby#summary)

## Jobs and instances | Prometheus

https://prometheus.io/docs/concepts/jobs\_instances/

**JOBS AND INSTANCES**

In Prometheus terms, any individually scraped target is called an instance, usually corresponding to a single process. A collection of instances of the same type (replicated for scalability or reliability) is called a job.

For example, an API server job with four replicated instances:

* job: api-server
  + instance 1: 1.2.3.4:5670
  + instance 2: 1.2.3.4:5671
  + instance 3: 5.6.7.8:5670
  + instance 4: 5.6.7.8:5671

**Automatically generated labels and time series**

When Prometheus scrapes a target, it attaches some labels automatically to the scraped time series which serve to identify the scraped target:

* job: The configured job name that the target belongs to.
* instance: The <host>:<port> part of the target's URL that was scraped.

If either of these labels are already present in the scraped data, the behavior depends on the honor\_labelsconfiguration option. See the [scrape configuration documentation](https://prometheus.io/docs/operating/configuration/#%3Cscrape_config%3E) for more information.

For each instance scrape, Prometheus stores a sample in the following time series:

* up{job="<job-name>", instance="<instance-id>"}: 1 if the instance is healthy, i.e. reachable, or 0 if the scrape failed.
* scrape\_duration\_seconds{job="<job-name>", instance="<instance-id>"}: duration of the scrape.
* scrape\_samples\_post\_metric\_relabeling{job="<job-name>", instance="<instance-id>"}: the number of samples remaining after metric relabeling was applied.
* scrape\_samples\_scraped{job="<job-name>", instance="<instance-id>"}: the number of samples the target exposed.

The up time series is useful for instance availability monitoring.

# Querying

## Querying basics | Prometheus

https://prometheus.io/docs/querying/basics/

**QUERYING PROMETHEUS**

* [Examples](https://prometheus.io/docs/querying/basics/#examples)
* [Expression language data types](https://prometheus.io/docs/querying/basics/#expression-language-data-types)
* [Literals](https://prometheus.io/docs/querying/basics/#literals)
  + [String literals](https://prometheus.io/docs/querying/basics/#string-literals)
  + [Float literals](https://prometheus.io/docs/querying/basics/#float-literals)
* [Time series Selectors](https://prometheus.io/docs/querying/basics/#time-series-selectors)
  + [Instant vector selectors](https://prometheus.io/docs/querying/basics/#instant-vector-selectors)
  + [Range Vector Selectors](https://prometheus.io/docs/querying/basics/#range-vector-selectors)
  + [Offset modifier](https://prometheus.io/docs/querying/basics/#offset-modifier)
* [Operators](https://prometheus.io/docs/querying/basics/#operators)
* [Functions](https://prometheus.io/docs/querying/basics/#functions)
* [Gotchas](https://prometheus.io/docs/querying/basics/#gotchas)
  + [Interpolation and staleness](https://prometheus.io/docs/querying/basics/#interpolation-and-staleness)
  + [Avoiding slow queries and overloads](https://prometheus.io/docs/querying/basics/#avoiding-slow-queries-and-overloads)

Prometheus provides a functional expression language that lets the user select and aggregate time series data in real time. The result of an expression can either be shown as a graph, viewed as tabular data in Prometheus's expression browser, or consumed by external systems via the [HTTP API](https://prometheus.io/docs/querying/api/).

**Examples**

This document is meant as a reference. For learning, it might be easier to start with a couple of [examples](https://prometheus.io/docs/querying/examples/).

### Expression language data types

In Prometheus's expression language, an expression or sub-expression can evaluate to one of four types:

* **Instant vector** - a set of time series containing a single sample for each time series, all sharing the same timestamp
* **Range vector** - a set of time series containing a range of data points over time for each time series
* **Scalar** - a simple numeric floating point value
* **String** - a simple string value; currently unused

Depending on the use-case (e.g. when graphing vs. displaying the output of an expression), only some of these types are legal as the result from a user-specified expression. For example, an expression that returns an instant vector is the only type that can be directly graphed.

**Literals**

String literals

Strings may be specified as literals in single quotes, double quotes or backticks.

PromQL follows the same [escaping rules as Go](https://golang.org/ref/spec#String_literals). In single or double quotes a backslash begins an escape sequence, which may be followed by a, b, f, n, r, t, v or \. Specific characters can be provided using octal (\nnn) or hexadecimal (\xnn, \unnnn and \Unnnnnnnn).

No escaping is processed inside backticks. Unlike Go, Prometheus does not discard newlines inside backticks.

Example:

"this is a string"

'these are unescaped: \n \\ \t'

`these are not unescaped: \n ' " \t`

Float literals

Scalar float values can be literally written as numbers of the form [-](digits)[.(digits)].

-2.43

### Time series Selectors

Instant vector selectors

Instant vector selectors allow the selection of a set of time series and a single sample value for each at a given timestamp (instant): in the simplest form, only a metric name is specified. This results in an instant vector containing elements for all time series that have this metric name.

This example selects all time series that have the http\_requests\_total metric name:

http\_requests\_total

It is possible to filter these time series further by appending a set of labels to match in curly braces ({}).

This example selects only those time series with the http\_requests\_total metric name that also have the job label set to prometheus and their group label set to canary:

http\_requests\_total{job="prometheus",group="canary"}

It is also possible to negatively match a label value, or to match label values against regular expressions. The following label matching operators exist:

* =: Select labels that are exactly equal to the provided string.
* !=: Select labels that are not equal to the provided string.
* =~: Select labels that regex-match the provided string (or substring).
* !~: Select labels that do not regex-match the provided string (or substring).

For example, this selects all http\_requests\_total time series for staging, testing, and developmentenvironments and HTTP methods other than GET.

http\_requests\_total{environment=~"staging|testing|development",method!="GET"}

Label matchers that match empty label values also select all time series that do not have the specific label set at all. Regex-matches are fully anchored.

Vector selectors must either specify a name or at least one label matcher that does not match the empty string. The following expression is illegal:

{job=~".\*"} # Bad!

In contrast, these expressions are valid as they both have a selector that does not match empty label values.

{job=~".+"} # Good!

{job=~".\*",method="get"} # Good!

Label matchers can also be applied to metric names by matching against the internal \_\_name\_\_ label. For example, the expression http\_requests\_total is equivalent to {\_\_name\_\_="http\_requests\_total"}. Matchers other than = (!=, =~, !~) may also be used. The following expression selects all metrics that have a name starting with job::

{\_\_name\_\_=~"^job:.\*"}

{\_\_name\_\_=~"^http\_requests\_total.\*"}

### Range Vector Selectors

Range vector literals work like instant vector literals, except that they select a range of samples back from the current instant. Syntactically, a range duration is appended in square brackets ([]) at the end of a vector selector to specify how far back in time values should be fetched for each resulting range vector element.

Time durations are specified as a number, followed immediately by one of the following units:

* s - seconds
* m - minutes
* h - hours
* d - days
* w - weeks
* y - years

In this example, we select all the values we have recorded within the last 5 minutes for all time series that have the metric name http\_requests\_total and a job label set to prometheus:

http\_requests\_total{job="prometheus"}[5m]

### Offset modifier

The offset modifier allows changing the time offset for individual instant and range vectors in a query.

For example, the following expression returns the value of http\_requests\_total 5 minutes in the past relative to the current query evaluation time:

http\_requests\_total offset 5m

Note that the offset modifier always needs to follow the selector immediately, i.e. the following would be correct:

sum(http\_requests\_total{method="GET"} offset 5m) // GOOD.

While the following would be incorrect:

sum(http\_requests\_total{method="GET"}) offset 5m // INVALID.

The same works for range vectors. This returns the 5-minutes rate that http\_requests\_total had a week ago:

rate(http\_requests\_total[5m] offset 1w)

**Operators**

Prometheus supports many binary and aggregation operators. These are described in detail in the [expression language operators](https://prometheus.io/docs/querying/operators/) page.

**Functions**

Prometheus supports several functions to operate on data. These are described in detail in the [expression language functions](https://prometheus.io/docs/querying/functions/) page.

**Gotchas**

Interpolation and staleness

When queries are run, timestamps at which to sample data are selected independently of the actual present time series data. This is mainly to support cases like aggregation (sum, avg, and so on), where multiple aggregated time series do not exactly align in time. Because of their independence, Prometheus needs to assign a value at those timestamps for each relevant time series. It does so by simply taking the newest sample before this timestamp.

If no stored sample is found (by default) 5 minutes before a sampling timestamp, no value is assigned for this time series at this point in time. This effectively means that time series "disappear" from graphs at times where their latest collected sample is older than 5 minutes.

**NOTE:** Staleness and interpolation handling might change. See<https://github.com/prometheus/prometheus/issues/398> and<https://github.com/prometheus/prometheus/issues/581>.

Avoiding slow queries and overloads

If a query needs to operate on a very large amount of data, graphing it might time out or overload the server or browser. Thus, when constructing queries over unknown data, always start building the query in the tabular view of Prometheus's expression browser until the result set seems reasonable (hundreds, not thousands, of time series at most). Only when you have filtered or aggregated your data sufficiently, switch to graph mode. If the expression still takes too long to graph ad-hoc, pre-record it via a [recording rule](https://prometheus.io/docs/querying/rules/#recording-rules).

This is especially relevant for Prometheus's query language, where a bare metric name selector likeapi\_http\_requests\_total could expand to thousands of time series with different labels. Also keep in mind that expressions which aggregate over many time series will generate load on the server even if the output is only a small number of time series. This is similar to how it would be slow to sum all values of a column in a relational database, even if the output value is only a single number.

## Operators | Prometheus

https://prometheus.io/docs/querying/operators/

**OPERATORS**

* [Binary operators](https://prometheus.io/docs/querying/operators/#binary-operators)
  + [Arithmetic binary operators](https://prometheus.io/docs/querying/operators/#arithmetic-binary-operators)
  + [Comparison binary operators](https://prometheus.io/docs/querying/operators/#comparison-binary-operators)
  + [Logical/set binary operators](https://prometheus.io/docs/querying/operators/#logical/set-binary-operators)
* [Vector matching](https://prometheus.io/docs/querying/operators/#vector-matching)
* [Aggregation operators](https://prometheus.io/docs/querying/operators/#aggregation-operators)
* [Binary operator precedence](https://prometheus.io/docs/querying/operators/#binary-operator-precedence)

**Binary operators**

Prometheus's query language supports basic logical and arithmetic operators. For operations between two instant vectors, the [matching behavior](https://prometheus.io/docs/querying/operators/#vector-matching) can be modified.

Arithmetic binary operators

The following binary arithmetic operators exist in Prometheus:

* + (addition)
* - (subtraction)
* \* (multiplication)
* / (division)
* % (modulo)
* ^ (power/exponentiation)

Binary arithmetic operators are defined between scalar/scalar, vector/scalar, and vector/vector value pairs.

**Between two scalars**, the behavior is obvious: they evaluate to another scalar that is the result of the operator applied to both scalar operands.

**Between an instant vector and a scalar**, the operator is applied to the value of every data sample in the vector. E.g. if a time series instant vector is multiplied by 2, the result is another vector in which every sample value of the original vector is multiplied by 2.

**Between two instant vectors**, a binary arithmetic operator is applied to each entry in the left-hand-side vector and its [matching element](https://prometheus.io/docs/querying/operators/#vector-matching) in the right hand vector. The result is propagated into the result vector and the metric name is dropped. Entries for which no matching entry in the right-hand vector can be found are not part of the result.

### Comparison binary operators

The following binary comparison operators exist in Prometheus:

* == (equal)
* != (not-equal)
* > (greater-than)
* < (less-than)
* >= (greater-or-equal)
* <= (less-or-equal)

Comparison operators are defined between scalar/scalar, vector/scalar, and vector/vector value pairs. By default they filter. Their behaviour can be modified by providing bool after the operator, which will return 0 or1 for the value rather than filtering.

**Between two scalars**, the bool modifier must be provided and these operators result in another scalar that is either 0 (false) or 1 (true), depending on the comparison result.

**Between an instant vector and a scalar**, these operators are applied to the value of every data sample in the vector, and vector elements between which the comparison result is false get dropped from the result vector. If the bool modifier is provided, vector elements that would be dropped instead have the value 0 and vector elements that would be kept have the value 1.

**Between two instant vectors**, these operators behave as a filter by default, applied to matching entries. Vector elements for which the expression is not true or which do not find a match on the other side of the expression get dropped from the result, while the others are propagated into a result vector with their original (left-hand-side) metric names and label values. If the bool modifier is provided, vector elements that would have been dropped instead have the value 0 and vector elements that would be kept have the value 1 with the left-hand-side metric names and label values.

Logical/set binary operators

These logical/set binary operators are only defined between instant vectors:

* and (intersection)
* or (union)
* unless (complement)

vector1 and vector2 results in a vector consisting of the elements of vector1 for which there are elements invector2 with exactly matching label sets. Other elements are dropped. The metric name and values are carried over from the left-hand-side vector.

vector1 or vector2 results in a vector that contains all original elements (label sets + values) of vector1 and additionally all elements of vector2 which do not have matching label sets in vector1.

vector1 unless vector2 results in a vector consisting of the elements of vector1 for which there are no elements in vector2 with exactly matching label sets. All matching elements in both vectors are dropped.

### Vector matching

Operations between vectors attempt to find a matching element in the right-hand-side vector for each entry in the left-hand side. There are two basic types of matching behavior:

**One-to-one** finds a unique pair of entries from each side of the operation. In the default case, that is an operation following the format vector1 <operator> vector2. Two entries match if they have the exact same set of labels and corresponding values. The ignoring keyword allows ignoring certain labels when matching, while the on keyword allows reducing the set of considered labels to a provided list:

<vector expr> <bin-op> ignoring(<label list>) <vector expr>

<vector expr> <bin-op> on(<label list>) <vector expr>

Example input:

method\_code:http\_errors:rate5m{method="get", code="500"} 24

method\_code:http\_errors:rate5m{method="get", code="404"} 30

method\_code:http\_errors:rate5m{method="put", code="501"} 3

method\_code:http\_errors:rate5m{method="post", code="500"} 6

method\_code:http\_errors:rate5m{method="post", code="404"} 21

method:http\_requests:rate5m{method="get"} 600

method:http\_requests:rate5m{method="del"} 34

method:http\_requests:rate5m{method="post"} 120

Example query:

method\_code:http\_errors:rate5m{code="500"} / ignoring(code) method:http\_requests:rate5m

This returns a result vector containing the fraction of HTTP requests with status code of 500 for each method, as measured over the last 5 minutes. Without ignoring(code) there would have been no match as the metrics do not share the same set of labels. The entries with methods put and del have no match and will not show up in the result:

{method="get"} 0.04 // 24 / 600

{method="post"} 0.05 // 6 / 120

**Many-to-one** and **one-to-many** matchings refer to the case where each vector element on the "one"-side can match with multiple elements on the "many"-side. This has to be explicitly requested using the group\_left orgroup\_right modifier, where left/right determines which vector has the higher cardinality.

<vector expr> <bin-op> ignoring(<label list>) group\_left(<label list>) <vector expr>

<vector expr> <bin-op> ignoring(<label list>) group\_right(<label list>) <vector expr>

<vector expr> <bin-op> on(<label list>) group\_left(<label list>) <vector expr>

<vector expr> <bin-op> on(<label list>) group\_right(<label list>) <vector expr>

The label list provided with the group modifier contains additional labels from the "one"-side to be included in the result metrics. For on a label can only appear in one of the lists. Every time series of the result vector must be uniquely identifiable.

Grouping modifiers can only be used for[*comparison*](https://prometheus.io/docs/querying/operators/#comparison-binary-operators)and[*arithmetic*](https://prometheus.io/docs/querying/operators/#arithmetic-binary-operators). Operations as *and*, *unless* and *or*operations match with all possible entries in the right vector by default.

Example query:

method\_code:http\_errors:rate5m / ignoring(code) group\_left method:http\_requests:rate5m

In this case the left vector contains more than one entry per method label value. Thus, we indicate this usinggroup\_left. The elements from the right side are now matched with multiple elements with the same methodlabel on the left:

{method="get", code="500"} 0.04 // 24 / 600

{method="get", code="404"} 0.05 // 30 / 600

{method="post", code="500"} 0.05 // 6 / 120

{method="post", code="404"} 0.175 // 21 / 120

Many-to-one and one-to-many matching are advanced use cases that should be carefully considered. Often a proper use of *ignoring(<labels>)* provides the desired outcome.

### Aggregation operators

Prometheus supports the following built-in aggregation operators that can be used to aggregate the elements of a single instant vector, resulting in a new vector of fewer elements with aggregated values:

* sum (calculate sum over dimensions)
* min (select minimum over dimensions)
* max (select maximum over dimensions)
* avg (calculate the average over dimensions)
* stddev (calculate population standard deviation over dimensions)
* stdvar (calculate population standard variance over dimensions)
* count (count number of elements in the vector)
* count\_values (count number of elements with the same value)
* bottomk (smallest k elements by sample value)
* topk (largest k elements by sample value)
* quantile (calculate φ-quantile (0 ≤ φ ≤ 1) over dimensions)

These operators can either be used to aggregate over **all** label dimensions or preserve distinct dimensions by including a without or by clause.

<aggr-op>([parameter,] <vector expression>) [without|by (<label list>)] [keep\_common]

parameter is only required for count\_values, quantile, topk and bottomk. without removes the listed labels from the result vector, while all other labels are preserved the output. by does the opposite and drops labels that are not listed in the by clause, even if their label values are identical between all elements of the vector. The keep\_common clause allows keeping those extra labels (labels that are identical between elements, but not in the by clause).

sum(http\_requests\_total) by (code)

count\_values outputs one time series per unique sample value. Each series has an additional label. The name of that label is given by the aggregation parameter, and the label value is the unique sample value. The value of each time series is the number of times that sample value was present.

topk and bottomk are different from other aggregators in that a subset of the input samples, including the original labels, are returned in the result vector. by and without are only used to bucket the input vector.

Example:

If the metric http\_requests\_total had time series that fan out by application, instance, and group labels, we could calculate the total number of seen HTTP requests per application and group over all instances via:

sum(http\_requests\_total) without (instance)

sum(http\_requests\_total) without (instance,method,handler,job,envtype,kubernetes\_io\_hostname,paasos\_platform,hostname)

If we are just interested in the total of HTTP requests we have seen in **all** applications, we could simply write:

sum(http\_requests\_total)

To count the number of binaries running each build version we could write:

count\_values("version", build\_version)

To get the 5 largest HTTP requests counts across all instances we could write:

topk(5, http\_requests\_total)

**Binary operator precedence**

The following list shows the precedence of binary operators in Prometheus, from highest to lowest.

1. ^
2. \*, /, %
3. +, -
4. ==, !=, <=, <, >=, >
5. and, unless
6. or

Operators on the same precedence level are left-associative. For example, 2 \* 3 % 2 is equivalent to (2 \* 3) % 2. However ^ is right associative, so 2 \* 3 ^ 2 is equivalent to 2 \* (3 ^ 2).

## Query functions | Prometheus

https://prometheus.io/docs/querying/functions/

**FUNCTIONS**

* [abs()](https://prometheus.io/docs/querying/functions/#abs())
* [absent()](https://prometheus.io/docs/querying/functions/#absent())
* [ceil()](https://prometheus.io/docs/querying/functions/#ceil())
* [changes()](https://prometheus.io/docs/querying/functions/#changes())
* [clamp\_max()](https://prometheus.io/docs/querying/functions/#clamp_max())
* [clamp\_min()](https://prometheus.io/docs/querying/functions/#clamp_min())
* [count\_scalar()](https://prometheus.io/docs/querying/functions/#count_scalar())
* [day\_of\_month()](https://prometheus.io/docs/querying/functions/#day_of_month())
* [day\_of\_week()](https://prometheus.io/docs/querying/functions/#day_of_week())
* [days\_in\_month()](https://prometheus.io/docs/querying/functions/#days_in_month())
* [delta()](https://prometheus.io/docs/querying/functions/#delta())
* [deriv()](https://prometheus.io/docs/querying/functions/#deriv())
* [drop\_common\_labels()](https://prometheus.io/docs/querying/functions/#drop_common_labels())
* [exp()](https://prometheus.io/docs/querying/functions/#exp())
* [floor()](https://prometheus.io/docs/querying/functions/#floor())
* [histogram\_quantile()](https://prometheus.io/docs/querying/functions/#histogram_quantile())
* [holt\_winters()](https://prometheus.io/docs/querying/functions/#holt_winters())
* [hour()](https://prometheus.io/docs/querying/functions/#hour())
* [idelta()](https://prometheus.io/docs/querying/functions/#idelta())
* [increase()](https://prometheus.io/docs/querying/functions/#increase())
* [irate()](https://prometheus.io/docs/querying/functions/#irate())
* [label\_replace()](https://prometheus.io/docs/querying/functions/#label_replace())
* [ln()](https://prometheus.io/docs/querying/functions/#ln())
* [log2()](https://prometheus.io/docs/querying/functions/#log2())
* [log10()](https://prometheus.io/docs/querying/functions/#log10())
* [minute()](https://prometheus.io/docs/querying/functions/#minute())
* [month()](https://prometheus.io/docs/querying/functions/#month())
* [predict\_linear()](https://prometheus.io/docs/querying/functions/#predict_linear())
* [rate()](https://prometheus.io/docs/querying/functions/#rate())
* [resets()](https://prometheus.io/docs/querying/functions/#resets())
* [round()](https://prometheus.io/docs/querying/functions/#round())
* [scalar()](https://prometheus.io/docs/querying/functions/#scalar())
* [sort()](https://prometheus.io/docs/querying/functions/#sort())
* [sort\_desc()](https://prometheus.io/docs/querying/functions/#sort_desc())
* [sqrt()](https://prometheus.io/docs/querying/functions/#sqrt())
* [time()](https://prometheus.io/docs/querying/functions/#time())
* [vector()](https://prometheus.io/docs/querying/functions/#vector())
* [year()](https://prometheus.io/docs/querying/functions/#year())
* [<aggregation>\_over\_time()](https://prometheus.io/docs/querying/functions/#<aggregation>_over_time())

Some functions have default arguments, e.g.year(v=vector(time()) instant-vector). This means that there is one argument v which is an instant vector, which if not provided it will default to the value of the expression vector(time()).

**abs()**

abs(v instant-vector) returns the input vector with all sample values converted to their absolute value.

**absent()**

absent(v instant-vector) returns an empty vector if the vector passed to it has any elements and a 1-element vector with the value 1 if the vector passed to it has no elements.

This is useful for alerting on when no time series exist for a given metric name and label combination.

absent(nonexistent{job="myjob"})

# => {job="myjob"}

absent(nonexistent{job="myjob",instance=~".\*"})

# => {job="myjob"}

absent(sum(nonexistent{job="myjob"}))

# => {}

In the second example, absent() tries to be smart about deriving labels of the 1-element output vector from the input vector.

**ceil()**

ceil(v instant-vector) rounds the sample values of all elements in v up to the nearest integer.

**changes()**

For each input time series, changes(v range-vector) returns the number of times its value has changed within the provided time range as an instant vector.

**clamp\_max()**

clamp\_max(v instant-vector, max scalar) clamps the sample values of all elements in v to have an upper limit of max.

**clamp\_min()**

clamp\_min(v instant-vector, min scalar) clamps the sample values of all elements in v to have a lower limit ofmin.

**count\_scalar()**

count\_scalar(v instant-vector) returns the number of elements in a time series vector as a scalar. This is in contrast to the count() [aggregation operator](https://prometheus.io/docs/querying/operators/#aggregation-operators), which always returns a vector (an empty one if the input vector is empty) and allows grouping by labels via a by clause.

**day\_of\_month()**

day\_of\_month(v=vector(time()) instant-vector) returns the day of the month for each of the given times in UTC. Returned values are from 1 to 31.

**day\_of\_week()**

day\_of\_week(v=vector(time()) instant-vector) returns the day of the week for each of the given times in UTC. Returned values are from 0 to 6, where 0 means Sunday etc.

**days\_in\_month()**

days\_in\_month(v=vector(time()) instant-vector) returns number of days in the month for each of the given times in UTC. Returned values are from 28 to 31.

**delta()**

delta(v range-vector) calculates the difference between the first and last value of each time series element in a range vector v, returning an instant vector with the given deltas and equivalent labels. The delta is interpolated to cover the full time range.

The following example expression returns the difference in CPU temperature between now and 2 hours ago:

delta(cpu\_temp\_celsius{host="zeus"}[2h])

delta should only be used with gauges.

**deriv()**

deriv(v range-vector) calculates the per-second derivative of the time series in a range vector v, using [simple linear regression](http://en.wikipedia.org/wiki/Simple_linear_regression).

deriv should only be used with gauges.

**drop\_common\_labels()**

drop\_common\_labels(instant-vector) drops all labels that have the same name and value across all series in the input vector.

**exp()**

exp(v instant-vector) calculates the exponential function for all elements in v. Special cases are:

* Exp(+Inf) = +Inf
* Exp(NaN) = NaN

**floor()**

floor(v instant-vector) rounds the sample values of all elements in v down to the nearest integer.

**histogram\_quantile()**

histogram\_quantile(φ float, b instant-vector) calculates the φ-quantile (0 ≤ φ ≤ 1) from the buckets b of a[histogram](https://prometheus.io/docs/concepts/metric_types/#histogram). (See [histograms and summaries](https://prometheus.io/docs/practices/histograms) for a detailed explanation of φ-quantiles and the usage of the histogram metric type in general.) The samples in b are the counts of observations in each bucket. Each sample must have a label le where the label value denotes the inclusive upper bound of the bucket. (Samples without such a label are silently ignored.) The [histogram metric type](https://prometheus.io/docs/concepts/metric_types/#histogram) automatically provides time series with the \_bucketsuffix and the appropriate labels.

Use the rate() function to specify the time window for the quantile calculation.

Example: A histogram metric is called http\_request\_duration\_seconds. To calculate the 90th percentile of request durations over the last 10m, use the following expression:

histogram\_quantile(0.9, rate(http\_request\_duration\_seconds\_bucket[10m]))

The quantile is calculated for each label combination in http\_request\_duration\_seconds. To aggregate, use thesum() aggregator around the rate() function. Since the le label is required by histogram\_quantile(), it has to be included in the by clause. The following expression aggregates the 90th percentile by job:

histogram\_quantile(0.9, sum(rate(http\_request\_duration\_seconds\_bucket[10m])) by (job, le))

To aggregate everything, specify only the le label:

histogram\_quantile(0.9, sum(rate(http\_request\_duration\_seconds\_bucket[10m])) by (le))

The histogram\_quantile() function interpolates quantile values by assuming a linear distribution within a bucket. The highest bucket must have an upper bound of +Inf. (Otherwise, NaN is returned.) If a quantile is located in the highest bucket, the upper bound of the second highest bucket is returned. A lower limit of the lowest bucket is assumed to be 0 if the upper bound of that bucket is greater than 0. In that case, the usual linear interpolation is applied within that bucket. Otherwise, the upper bound of the lowest bucket is returned for quantiles located in the lowest bucket.

If b contains fewer than two buckets, NaN is returned. For φ < 0, -Inf is returned. For φ > 1, +Inf is returned.

**holt\_winters()**

holt\_winters(v range-vector, sf scalar, tf scalar) produces a smoothed value for time series based on the range in v. The lower the smoothing factor sf, the more importance is given to old data. The higher the trend factor tf, the more trends in the data is considered. Both sf and tf must be between 0 and 1.

holt\_winters should only be used with gauges.

**hour()**

hour(v=vector(time()) instant-vector) returns the hour of the day for each of the given times in UTC. Returned values are from 0 to 23.

**idelta()**

idelta(v range-vector)

idelta(v range-vector) calculates the difference between the last two samples in the range vector v, returning an instant vector with the given deltas and equivalent labels.

idelta should only be used with gauges.

**increase()**

increase(v range-vector) calculates the increase in the time series in the range vector. Breaks in monotonicity (such as counter resets due to target restarts) are automatically adjusted for.

The following example expression returns the number of HTTP requests as measured over the last 5 minutes, per time series in the range vector:

increase(http\_requests\_total{job="api-server"}[5m])

increase should only be used with counters. It should be used primarily for human readability. Use rate in recording rules so that increases are tracked consistently on a per-second basis.

### irate()

irate(v range-vector) calculates the per-second instant rate of increase of the time series in the range vector. This is based on the last two data points. Breaks in monotonicity (such as counter resets due to target restarts) are automatically adjusted for.

The following example expression returns the per-second rate of HTTP requests looking up to 5 minutes back for the two most recent data points, per time series in the range vector:

irate(http\_requests\_total{job="api-server"}[5m])

irate should only be used when graphing volatile, fast-moving counters. Use rate for alerts and slow-moving counters, as brief changes in the rate can reset the FOR clause and graphs consisting entirely of rare spikes are hard to read.

Note that when combining irate() with an [aggregation operator](https://prometheus.io/docs/querying/operators/#aggregation-operators) (e.g. sum()) or a function aggregating over time (any function ending in \_over\_time), always take a irate() first, then aggregate. Otherwise irate()cannot detect counter resets when your target restarts.

**label\_replace()**

For each timeseries in v, label\_replace(v instant-vector, dst\_label string, replacement string, src\_label string, regex string) matches the regular expression regex against the label src\_label. If it matches, then the timeseries is returned with the label dst\_label replaced by the expansion of replacement. $1 is replaced with the first matching subgroup, $2 with the second etc. If the regular expression doesn't match then the timeseries is returned unchanged.

This example will return a vector with each time series having a foo label with the value a added to it:label\_replace(up{job="api-server",service="a:c"}, "foo", "$1", "service", "(.\*):.\*")

**ln()**

ln(v instant-vector) calculates the natural logarithm for all elements in v. Special cases are:

* ln(+Inf) = +Inf
* ln(0) = -Inf
* ln(x < 0) = NaN
* ln(NaN) = NaN

**log2()**

log2(v instant-vector) calculates the binary logarithm for all elements in v. The special cases are equivalent to those in ln.

**log10()**

log10(v instant-vector) calculates the decimal logarithm for all elements in v. The special cases are equivalent to those in ln.

**minute()**

minute(v=vector(time()) instant-vector) returns the minute of the hour for each of the given times in UTC. Returned values are from 0 to 59.

**month()**

month(v=vector(time()) instant-vector) returns the month of the year for each of the given times in UTC. Returned values are from 1 to 12, where 1 means January etc.

**predict\_linear()**

predict\_linear(v range-vector, t scalar) predicts the value of time series t seconds from now, based on the range vector v, using [simple linear regression](http://en.wikipedia.org/wiki/Simple_linear_regression).

predict\_linear should only be used with gauges.

### rate()

rate(v range-vector) calculates the per-second average rate of increase of the time series in the range vector. Breaks in monotonicity (such as counter resets due to target restarts) are automatically adjusted for.

The following example expression returns the per-second rate of HTTP requests as measured over the last 5 minutes, per time series in the range vector:

rate(http\_requests\_total{job="api-server"}[5m])

rate should only be used with counters. It is best suited for alerting, and for graphing of slow-moving counters.

Note that when combining rate() with an aggregation operator (e.g. sum()) or a function aggregating over time (any function ending in \_over\_time), always take a rate() first, then aggregate. Otherwise rate() cannot detect counter resets when your target restarts.

**resets()**

For each input time series, resets(v range-vector) returns the number of counter resets within the provided time range as an instant vector. Any decrease in the value between two consecutive samples is interpreted as a counter reset.

resets should only be used with counters.

**round()**

round(v instant-vector, to\_nearest=1 scalar) rounds the sample values of all elements in v to the nearest integer. Ties are resolved by rounding up. The optional to\_nearest argument allows specifying the nearest multiple to which the sample values should be rounded. This multiple may also be a fraction.

**scalar()**

Given a single-element input vector, scalar(v instant-vector) returns the sample value of that single element as a scalar. If the input vector does not have exactly one element, scalar will return NaN.

**sort()**

sort(v instant-vector) returns vector elements sorted by their sample values, in ascending order.

**sort\_desc()**

Same as sort, but sorts in descending order.

**sqrt()**

sqrt(v instant-vector) calculates the square root of all elements in v.

**time()**

time() returns the number of seconds since January 1, 1970 UTC. Note that this does not actually return the current time, but the time at which the expression is to be evaluated.

**vector()**

vector(s scalar) returns the scalar s as a vector with no labels.

**year()**

year(v=vector(time()) instant-vector) returns the year for each of the given times in UTC.

**<aggregation>\_over\_time()**

The following functions allow aggregating each series of a given range vector over time and return an instant vector with per-series aggregation results:

* avg\_over\_time(range-vector): the average value of all points in the specified interval.
* min\_over\_time(range-vector): the minimum value of all points in the specified interval.
* max\_over\_time(range-vector): the maximum value of all points in the specified interval.
* sum\_over\_time(range-vector): the sum of all values in the specified interval.
* count\_over\_time(range-vector): the count of all values in the specified interval.
* quantile\_over\_time(scalar, range-vector): the φ-quantile (0 ≤ φ ≤ 1) of the values in the specified interval.
* stddev\_over\_time(range-vector): the population standard deviation of the values in the specified interval.
* stdvar\_over\_time(range-vector): the population standard variance of the values in the specified interval.

Note that all values in the specified interval have the same weight in the aggregation even if the values are not equally spaced throughout the interval.

## Querying examples | Prometheus

https://prometheus.io/docs/querying/examples/

**QUERY EXAMPLES**

* [Simple time series selection](https://prometheus.io/docs/querying/examples/#simple-time-series-selection)
* [Using functions, operators, etc.](https://prometheus.io/docs/querying/examples/#using-functions,-operators,-etc.)

### Simple time series selection

Return all time series with the metric http\_requests\_total:

http\_requests\_total

Return all time series with the metric http\_requests\_total and the given job and handler labels:

http\_requests\_total{job="apiserver", handler="/api/comments"}

Return a whole range of time (in this case 5 minutes) for the same vector, making it a range vector:

http\_requests\_total{job="apiserver", handler="/api/comments"}[5m]

Note that an expression resulting in a range vector cannot be graphed directly, but viewed in the tabular ("Console") view of the expression browser.

Using regular expressions, you could select time series only for jobs whose name match a certain pattern, in this case, all jobs that end with server. Note that this does a substring match, not a full string match:

http\_requests\_total{job=~"server$"}

To select all HTTP status codes except 4xx ones, you could run:

http\_requests\_total{status!~"^4..$"}

### Using functions, operators, etc.

Return the per-second rate for all time series with the http\_requests\_total metric name, as measured over the last 5 minutes:

rate(http\_requests\_total[5m])

Assuming that the http\_requests\_total time series all have the labels job (fanout by job name) and instance(fanout by instance of the job), we might want to sum over the rate of all instances, so we get fewer output time series, but still preserve the job dimension:

sum(rate(http\_requests\_total[5m])) by (job)

If we have two different metrics with the same dimensional labels, we can apply binary operators to them and elements on both sides with the same label set will get matched and propagated to the output. For example, this expression returns the unused memory in MiB for every instance (on a fictional cluster scheduler exposing these metrics about the instances it runs):

(instance\_memory\_limit\_bytes - instance\_memory\_usage\_bytes) / 1024 / 1024

The same expression, but summed by application, could be written like this:

sum(

instance\_memory\_limit\_bytes - instance\_memory\_usage\_bytes

) by (app, proc) / 1024 / 1024

If the same fictional cluster scheduler exposed CPU usage metrics like the following for every instance:

instance\_cpu\_time\_ns{app="lion", proc="web", rev="34d0f99", env="prod", job="cluster-manager"}

instance\_cpu\_time\_ns{app="elephant", proc="worker", rev="34d0f99", env="prod", job="cluster-manager"}

instance\_cpu\_time\_ns{app="turtle", proc="api", rev="4d3a513", env="prod", job="cluster-manager"}

instance\_cpu\_time\_ns{app="fox", proc="widget", rev="4d3a513", env="prod", job="cluster-manager"}

...

...we could get the top 3 CPU users grouped by application (app) and process type (proc) like this:

topk(3, sum(rate(instance\_cpu\_time\_ns[5m])) by (app, proc))

Assuming this metric contains one time series per running instance, you could count the number of running instances per application like this:

count(instance\_cpu\_time\_ns) by (app)

## Recording rules | Prometheus

https://prometheus.io/docs/querying/rules/

**DEFINING RECORDING RULES**

* [Configuring rules](https://prometheus.io/docs/querying/rules/#configuring-rules)
* [Syntax-checking rules](https://prometheus.io/docs/querying/rules/#syntax-checking-rules)
* [Recording rules](https://prometheus.io/docs/querying/rules/#recording-rules)

**Configuring rules**

Prometheus supports two types of rules which may be configured and then evaluated at regular intervals: recording rules and[alerting rules](https://prometheus.io/docs/alerting/rules). To include rules in Prometheus, create a file containing the necessary rule statements and have Prometheus load the file via the rule\_files field in the[Prometheus configuration](https://prometheus.io/docs/operating/configuration).

The rule files can be reloaded at runtime by sending SIGHUP to the Prometheus process. The changes are only applied if all rule files are well-formatted.

**Syntax-checking rules**

To quickly check whether a rule file is syntactically correct without starting a Prometheus server, install and run Prometheus's promtool command-line utility tool:

go get github.com/prometheus/prometheus/cmd/promtool

promtool check-rules /path/to/example.rules

When the file is syntactically valid, the checker prints a textual representation of the parsed rules to standard output and then exits with a 0 return status.

If there are any syntax errors, it prints an error message to standard error and exits with a 1 return status. On invalid input arguments the exit status is 2.

**Recording rules**

Recording rules allow you to precompute frequently needed or computationally expensive expressions and save their result as a new set of time series. Querying the precomputed result will then often be much faster than executing the original expression every time it is needed. This is especially useful for dashboards, which need to query the same expression repeatedly every time they refresh.

To add a new recording rule, add a line of the following syntax to your rule file:

<new time series name>[{<label overrides>}] = <expression to record>

Some examples:

# Saving the per-job HTTP in-progress request count as a new set of time series:

job:http\_inprogress\_requests:sum = sum(http\_inprogress\_requests) by (job)

# Drop or rewrite labels in the result time series:

new\_time\_series{label\_to\_change="new\_value",label\_to\_drop=""} = old\_time\_series

Recording rules are evaluated at the interval specified by the evaluation\_interval field in the Prometheus configuration. During each evaluation cycle, the right-hand-side expression of the rule statement is evaluated at the current instant in time and the resulting sample vector is stored as a new set of time series with the current timestamp and a new metric name (and perhaps an overridden set of labels).

## HTTP API | Prometheus

https://prometheus.io/docs/querying/api/

**HTTP API**

* [Format overview](https://prometheus.io/docs/querying/api/#format-overview)
* [Expression queries](https://prometheus.io/docs/querying/api/#expression-queries)
  + [Instant queries](https://prometheus.io/docs/querying/api/#instant-queries)
  + [Range queries](https://prometheus.io/docs/querying/api/#range-queries)
* [Querying metadata](https://prometheus.io/docs/querying/api/#querying-metadata)
  + [Finding series by label matchers](https://prometheus.io/docs/querying/api/#finding-series-by-label-matchers)
  + [Querying label values](https://prometheus.io/docs/querying/api/#querying-label-values)
* [Deleting series](https://prometheus.io/docs/querying/api/#deleting-series)
* [Expression query result formats](https://prometheus.io/docs/querying/api/#expression-query-result-formats)
  + [Range vectors](https://prometheus.io/docs/querying/api/#range-vectors)
  + [Instant vectors](https://prometheus.io/docs/querying/api/#instant-vectors)
  + [Scalars](https://prometheus.io/docs/querying/api/#scalars)
  + [Strings](https://prometheus.io/docs/querying/api/#strings)
* [Targets](https://prometheus.io/docs/querying/api/#targets)
* [Alertmanagers](https://prometheus.io/docs/querying/api/#alertmanagers)

The current stable HTTP API is reachable under /api/v1 on a Prometheus server. Any non-breaking additions will be added under that endpoint.

**Format overview**

The API response format is JSON. Every successful API request returns a 2xx status code.

Invalid requests that reach the API handlers return a JSON error object and one of the following HTTP response codes:

* 400 Bad Request when parameters are missing or incorrect.
* 422 Unprocessable Entity when an expression can't be executed ([RFC4918](http://tools.ietf.org/html/rfc4918#page-78)).
* 503 Service Unavailable when queries time out or abort.

Other non-2xx codes may be returned for errors occurring before the API endpoint is reached.

The JSON response envelope format is as follows:

{

"status": "success" | "error",

"data": <data>,

// Only set if status is "error". The data field may still hold

// additional data.

"errorType": "<string>",

"error": "<string>"

}

Input timestamps may be provided either in [RFC3339](https://www.ietf.org/rfc/rfc3339.txt) format or as a Unix timestamp in seconds, with optional decimal places for sub-second precision. Output timestamps are always represented as Unix timestamps in seconds.

Names of query parameters that may be repeated end with [].

<series\_selector> placeholders refer to Prometheus [time series selectors](https://prometheus.io/docs/querying/basics/#time-series-selectors) like http\_requests\_total orhttp\_requests\_total{method=~"^GET|POST$"} and need to be URL-encoded.

<duration> placeholders refer to Prometheus duration strings of the form [0-9]+[smhdwy]. For example, 5mrefers to a duration of 5 minutes.

**Expression queries**

Query language expressions may be evaluated at a single instant or over a range of time. The sections below describe the API endpoints for each type of expression query.

Instant queries

The following endpoint evaluates an instant query at a single point in time:

GET /api/v1/query

URL query parameters:

* query=<string>: Prometheus expression query string.
* time=<rfc3339 | unix\_timestamp>: Evaluation timestamp. Optional.
* timeout=<duration>: Evaluation timeout. Optional. Defaults to and is capped by the value of the -query.timeout flag.

The current server time is used if the time parameter is omitted.

The data section of the query result has the following format:

{

"resultType": "matrix" | "vector" | "scalar" | "string",

"result": <value>

}

<value> refers to the query result data, which has varying formats depending on the resultType. See the[expression query result formats](https://prometheus.io/docs/querying/api/#expression-query-result-formats).

The following example evaluates the expression up at the time 2015-07-01T20:10:51.781Z:

$ curl 'http://localhost:9090/api/v1/query?query=up&time=2015-07-01T20:10:51.781Z'

{

"status" : "success",

"data" : {

"resultType" : "vector",

"result" : [

{

"metric" : {

"\_\_name\_\_" : "up",

"job" : "prometheus",

"instance" : "localhost:9090"

},

"value": [ 1435781451.781, "1" ]

},

{

"metric" : {

"\_\_name\_\_" : "up",

"job" : "node",

"instance" : "localhost:9100"

},

"value" : [ 1435781451.781, "0" ]

}

]

}

}

Range queries

The following endpoint evaluates an expression query over a range of time:

GET /api/v1/query\_range

URL query parameters:

* query=<string>: Prometheus expression query string.
* start=<rfc3339 | unix\_timestamp>: Start timestamp.
* end=<rfc3339 | unix\_timestamp>: End timestamp.
* step=<duration>: Query resolution step width.
* timeout=<duration>: Evaluation timeout. Optional. Defaults to and is capped by the value of the -query.timeout flag.

The data section of the query result has the following format:

{

"resultType": "matrix",

"result": <value>

}

For the format of the <value> placeholder, see the [range-vector result format](https://prometheus.io/docs/querying/api/#range-vectors).

The following example evaluates the expression up over a 30-second range with a query resolution of 15 seconds.

$ curl 'http://localhost:9090/api/v1/query\_range?query=up&start=2015-07-01T20:10:30.781Z&end=2015-07-01T20:11:00.781Z&step=15s'

{

"status" : "success",

"data" : {

"resultType" : "matrix",

"result" : [

{

"metric" : {

"\_\_name\_\_" : "up",

"job" : "prometheus",

"instance" : "localhost:9090"

},

"values" : [

[ 1435781430.781, "1" ],

[ 1435781445.781, "1" ],

[ 1435781460.781, "1" ]

]

},

{

"metric" : {

"\_\_name\_\_" : "up",

"job" : "node",

"instance" : "localhost:9091"

},

"values" : [

[ 1435781430.781, "0" ],

[ 1435781445.781, "0" ],

[ 1435781460.781, "1" ]

]

}

]

}

}

**Querying metadata**

Finding series by label matchers

The following endpoint returns the list of time series that match a certain label set.

GET /api/v1/series

URL query parameters:

* match[]=<series\_selector>: Repeated series selector argument that selects the series to return. At least one match[] argument must be provided.
* start=<rfc3339 | unix\_timestamp>: Start timestamp.
* end=<rfc3339 | unix\_timestamp>: End timestamp.

The data section of the query result consists of a list of objects that contain the label name/value pairs which identify each series.

The following example returns all series that match either of the selectors up orprocess\_start\_time\_seconds{job="prometheus"}:

$ curl -g 'http://localhost:9090/api/v1/series?match[]=up&match[]=process\_start\_time\_seconds{job="prometheus"}'

{

"status" : "success",

"data" : [

{

"\_\_name\_\_" : "up",

"job" : "prometheus",

"instance" : "localhost:9090"

},

{

"\_\_name\_\_" : "up",

"job" : "node",

"instance" : "localhost:9091"

},

{

"\_\_name\_\_" : "process\_start\_time\_seconds",

"job" : "prometheus",

"instance" : "localhost:9090"

}

]

}

Querying label values

The following endpoint returns a list of label values for a provided label name:

GET /api/v1/label/<label\_name>/values

The data section of the JSON response is a list of string label names.

This example queries for all label values for the job label:

$ curl http://localhost:9090/api/v1/label/job/values

{

"status" : "success",

"data" : [

"node",

"prometheus"

]

}

**Deleting series**

The following endpoint deletes matched series entirely from a Prometheus server:

DELETE /api/v1/series

URL query parameters:

* match[]=<series\_selector>: Repeated label matcher argument that selects the series to delete. At least one match[] argument must be provided.

The data section of the JSON response has the following format:

{

"numDeleted": <number of deleted series>

}

The following example deletes all series that match either of the selectors up orprocess\_start\_time\_seconds{job="prometheus"}:

$ curl -XDELETE -g 'http://localhost:9090/api/v1/series?match[]=up&match[]=process\_start\_time\_seconds{job="prometheus"}'

{

"status" : "success",

"data" : {

"numDeleted" : 3

}

}

**Expression query result formats**

Expression queries may return the following response values in the result property of the data section.<sample\_value> placeholders are numeric sample values. JSON does not support special float values such asNaN, Inf, and -Inf, so sample values are transferred as quoted JSON strings rather than raw numbers.

Range vectors

Range vectors are returned as result type matrix. The corresponding result property has the following format:

[

{

"metric": { "<label\_name>": "<label\_value>", ... },

"values": [ [ <unix\_time>, "<sample\_value>" ], ... ]

},

...

]

Instant vectors

Instant vectors are returned as result type vector. The corresponding result property has the following format:

[

{

"metric": { "<label\_name>": "<label\_value>", ... },

"value": [ <unix\_time>, "<sample\_value>" ]

},

...

]

Scalars

Scalar results are returned as result type scalar. The corresponding result property has the following format:

[ <unix\_time>, "<scalar\_value>" ]

Strings

String results are returned as result type string. The corresponding result property has the following format:

[ <unix\_time>, "<string\_value>" ]

**Targets**

This API is experimental as it is intended to be extended with targets dropped due to relabelling in the future.

The following endpoint returns an overview of the current state of the Prometheus target discovery:

GET /api/v1/targets

Currently only the active targets are part of the response.

$ curl http://localhost:9090/api/v1/targets

{

"status": "success", [3/11]

"data": {

"activeTargets": [

{

"discoveredLabels": {

"\_\_address\_\_": "127.0.0.1:9090",

"\_\_metrics\_path\_\_": "/metrics",

"\_\_scheme\_\_": "http",

"job": "prometheus"

},

"labels": {

"instance": "127.0.0.1:9090",

"job": "prometheus"

},

"scrapeUrl": "http://127.0.0.1:9090/metrics",

"lastError": "",

"lastScrape": "2017-01-17T15:07:44.723715405+01:00",

"health": "up"

}

]

}

}

**Alertmanagers**

This API is experimental as it is intended to be extended with Alertmanagers dropped due to relabelling in the future.

The following endpoint returns an overview of the current state of the Prometheus alertmanager discovery:

GET /api/v1/alertmanagers

Currently only the active Alertmanagers are part of the response.

$ curl http://localhost:9090/api/v1/alertmanagers

{

"status": "success",

"data": {

"activeAlertmanagers": [

{

"url": "http://127.0.0.1:9090/api/v1/alerts"

}

]

}

}

# Visualization

## Console templates | Prometheus

https://prometheus.io/docs/visualization/consoles/

CONSOLE TEMPLATES

* [Getting started](https://prometheus.io/docs/visualization/consoles/#getting-started)
* [Example Console](https://prometheus.io/docs/visualization/consoles/#example-console)
* [Graph Library](https://prometheus.io/docs/visualization/consoles/#graph-library)

Console templates allow for creation of arbitrary consoles using the[Go templating language](http://golang.org/pkg/text/template/). These are served from the Prometheus server.

Console templates are the most powerful way to create templates that can be easily managed in source control. There is a learning curve though, so users new to this style of monitoring should try out [Grafana](https://prometheus.io/docs/visualization/grafana/) first.

**Getting started**

Prometheus comes with an example set of consoles to get you going. These can be found at/consoles/index.html.example on a running Prometheus and will display Node Exporter consoles if Prometheus is scraping Node Exporters with a job="node" label.

The example consoles have 5 parts:

1. A navigation bar on top
2. A menu on the left
3. Time controls on the bottom
4. The main content in the center, usually graphs
5. A table on the right

The navigation bar is for links to other systems, such as other Prometheis [1](https://prometheus.io/docs/introduction/faq/#what-is-the-plural-of-prometheus), documentation, and whatever else makes sense to you. The menu is for navigation inside the same Prometheus server, which is very useful to be able to quickly open a console in another tab to correlate information. Both are configured in console\_libraries/menu.lib.

The time controls allow changing of the duration and range of the graphs. Console URLs can be shared and will show the same graphs for others.

The main content is usually graphs. There is a configurable JavaScript graphing library provided that will handle requesting data from Prometheus, and rendering it via [Rickshaw](http://code.shutterstock.com/rickshaw/).

Finally, the table on the right can be used to display statistics in a more compact form than graphs.

**Example Console**

This is a basic console. It shows the number of tasks, how many of them are up, the average CPU usage, and the average memory usage in the right-hand-side table. The main content has a queries-per-second graph.

{{template "head" .}}

{{template "prom\_right\_table\_head"}}

<tr>

<th>MyJob</th>

<th>{{ template "prom\_query\_drilldown" (args "sum(up{job='myjob'})") }}

/ {{ template "prom\_query\_drilldown" (args "count(up{job='myjob'})") }}

</th>

</tr>

<tr>

<td>CPU</td>

<td>{{ template "prom\_query\_drilldown" (args

"avg by(job)(rate(process\_cpu\_seconds\_total{job='myjob'}[5m]))"

"s/s" "humanizeNoSmallPrefix") }}

</td>

</tr>

<tr>

<td>Memory</td>

<td>{{ template "prom\_query\_drilldown" (args

"avg by(job)(process\_resident\_memory\_bytes{job='myjob'})"

"B" "humanize1024") }}

</td>

</tr>

{{template "prom\_right\_table\_tail"}}

{{template "prom\_content\_head" .}}

<h1>MyJob</h1>

<h3>Queries</h3>

<div id="queryGraph"></div>

<script>

new PromConsole.Graph({

node: document.querySelector("#queryGraph"),

expr: "sum(rate(http\_query\_count{job='myjob'}[5m]))",

name: "Queries",

yAxisFormatter: PromConsole.NumberFormatter.humanizeNoSmallPrefix,

yHoverFormatter: PromConsole.NumberFormatter.humanizeNoSmallPrefix,

yUnits: "/s",

yTitle: "Queries"

})

</script>

{{template "prom\_content\_tail" .}}

{{template "tail"}}

The prom\_right\_table\_head and prom\_right\_table\_tail templates contain the right-hand-side table. This is optional.

prom\_query\_drilldown is a template that will evaluate the expression passed to it, format it, and link to the expression in the [expression browser](https://prometheus.io/docs/visualization/browser/). The first argument is the expression. The second argument is the unit to use. The third argument is how to format the output. Only the first argument is required.

Valid output formats for the third argument to prom\_query\_drilldown:

* Not specified: Default Go display output.
* humanize: Display the result using [metric prefixes](http://en.wikipedia.org/wiki/Metric_prefix).
* humanizeNoSmallPrefix: For absolute values greater than 1, display the result using [metric prefixes](http://en.wikipedia.org/wiki/Metric_prefix). For absolute values less than 1, display 3 significant digits. This is useful to avoid units such as milliqueries per second that can be produced by humanize.
* humanize1024: Display the humanized result using a base of 1024 rather than 1000. This is usually used withB as the second argument to produce units such as KiB and MiB.
* printf.3g: Display 3 significant digits.

Custom formats can be defined. See [prom.lib](https://github.com/prometheus/prometheus/blob/master/console_libraries/prom.lib) for examples.

**Graph Library**

The graph library is invoked as:

<div id="queryGraph"></div>

<script>

new PromConsole.Graph({

node: document.querySelector("#queryGraph"),

expr: "sum(rate(http\_query\_count{job='myjob'}[5m]))"

})

</script>

The head template loads the required Javascript and CSS.

Parameters to the graph library:

| **Name** | **Description** |
| --- | --- |
| expr | Required. Expression to graph. Can be a list. |
| node | Required. DOM node to render into. |
| duration | Optional. Duration of the graph. Defaults to 1 hour. |
| endTime | Optional. Unixtime the graph ends at. Defaults to now. |
| width | Optional. Width of the graph, excluding titles. Defaults to auto-detection. |
| height | Optional. Height of the graph, excluding titles and legends. Defaults to 200 pixels. |
| min | Optional. Minimum x-axis value. Defaults to lowest data value. |
| max | Optional. Maximum y-axis value. Defaults to highest data value. |
| renderer | Optional. Type of graph. Options are line and area (stacked graph). Defaults to line. |
| name | Optional. Title of plots in legend and hover detail. If passed a string, [[ label ]] will be substituted with the label value. If passed a function, it will be passed a map of labels and should return the name as a string. Can be a list. |
| xTitle | Optional. Title of the x-axis. Defaults to Time. |
| yUnits | Optional. Units of the y-axis. Defaults to empty. |
| yTitle | Optional. Title of the y-axis. Defaults to empty. |
| yAxisFormatter | Optional. Number formatter for the y-axis. Defaults toPromConsole.NumberFormatter.humanize. |
| yHoverFormatter | Optional. Number formatter for the hover detail. Defaults toPromConsole.NumberFormatter.humanizeExact. |

If both expr and name are lists, they must be of the same length. The name will be applied to the plots for the corresponding expression.

Valid options for the yAxisFormatter and yHoverFormatter:

* PromConsole.NumberFormatter.humanize: Format using [metric prefixes](http://en.wikipedia.org/wiki/Metric_prefix).
* PromConsole.NumberFormatter.humanizeNoSmallPrefix: For absolute values greater than 1, format using using [metric prefixes](http://en.wikipedia.org/wiki/Metric_prefix). For absolute values less than 1, format with 3 significant digits. This is useful to avoid units such as milliqueries per second that can be produced by PromConsole.NumberFormatter.humanize.
* PromConsole.NumberFormatter.humanize1024: Format the humanized result using a base of 1024 rather than 1000.

# Intrumenting

## Exporters and integrations | Prometheus

https://prometheus.io/docs/instrumenting/exporters/

EXPORTERS AND INTEGRATIONS

* [Third-party exporters](https://prometheus.io/docs/instrumenting/exporters/#third-party-exporters)
  + [Databases](https://prometheus.io/docs/instrumenting/exporters/#databases)
  + [Hardware related](https://prometheus.io/docs/instrumenting/exporters/#hardware-related)
  + [Messaging systems](https://prometheus.io/docs/instrumenting/exporters/#messaging-systems)
  + [Storage](https://prometheus.io/docs/instrumenting/exporters/#storage)
  + [HTTP](https://prometheus.io/docs/instrumenting/exporters/#http)
  + [APIs](https://prometheus.io/docs/instrumenting/exporters/#apis)
  + [Logging](https://prometheus.io/docs/instrumenting/exporters/#logging)
  + [Other monitoring systems](https://prometheus.io/docs/instrumenting/exporters/#other-monitoring-systems)
  + [Miscellaneous](https://prometheus.io/docs/instrumenting/exporters/#miscellaneous)
* [Directly instrumented software](https://prometheus.io/docs/instrumenting/exporters/#directly-instrumented-software)
* [Other third-party utilities](https://prometheus.io/docs/instrumenting/exporters/#other-third-party-utilities)

There are a number of libraries and servers which help in exporting existing metrics from third-party systems as Prometheus metrics. This is useful for cases where it is not feasible to instrument a given system with Prometheus metrics directly (for example, HAProxy or Linux system stats).

**Third-party exporters**

Some of these exporters are maintained as part of the official[Prometheus GitHub organization](https://github.com/prometheus), those are marked as official, others are externally contributed and maintained.

We encourage the creation of more exporters but, cannot vet all of them for [best practices](https://prometheus.io/docs/instrumenting/writing_exporters/). Commonly, those exporters are hosted outside of the Prometheus GitHub organization.

The [exporter default port](https://github.com/prometheus/prometheus/wiki/Default-port-allocations) wiki page has become another catalog of exporters, and may include exporters not listed here due to overlapping functionality or still being in development.

The [JMX exporter](https://github.com/prometheus/jmx_exporter) can export from a wide variety of JVM-based applications, for example [Kafka](http://kafka.apache.org/) and [Cassandra](http://cassandra.apache.org/).

Databases

* [Aerospike exporter](https://github.com/alicebob/asprom)
* [ClickHouse exporter](https://github.com/f1yegor/clickhouse_exporter)
* [Consul exporter](https://github.com/prometheus/consul_exporter) (**official**)
* [CouchDB exporter](https://github.com/gesellix/couchdb-exporter)
* [ElasticSearch exporter](https://github.com/justwatchcom/elasticsearch_exporter)
* [Memcached exporter](https://github.com/prometheus/memcached_exporter) (**official**)
* [MongoDB exporter](https://github.com/dcu/mongodb_exporter)
* [MySQL server exporter](https://github.com/prometheus/mysqld_exporter) (**official**)
* [OpenTSDB Exporter](https://github.com/cloudflare/opentsdb_exporter)
* [PgBouncer exporter](http://git.cbaines.net/prometheus-pgbouncer-exporter/about)
* [PostgreSQL exporter](https://github.com/wrouesnel/postgres_exporter)
* [ProxySQL exporter](https://github.com/percona/proxysql_exporter)
* [Redis exporter](https://github.com/oliver006/redis_exporter)
* [RethinkDB exporter](https://github.com/oliver006/rethinkdb_exporter)
* [SQL query result set metrics exporter](https://github.com/chop-dbhi/prometheus-sql)
* [Tarantool metric library](https://github.com/tarantool/prometheus)

Hardware related

* [apcupsd exporter](https://github.com/mdlayher/apcupsd_exporter)
* [IoT Edison exporter](https://github.com/roman-vynar/edison_exporter)
* [IPMI exporter](https://github.com/lovoo/ipmi_exporter)
* [knxd exporter](https://github.com/RichiH/knxd_exporter)
* [Node/system metrics exporter](https://github.com/prometheus/node_exporter) (**official**)
* [Ubiquiti UniFi exporter](https://github.com/mdlayher/unifi_exporter)

Messaging systems

* [NATS exporter](https://github.com/lovoo/nats_exporter)
* [NSQ exporter](https://github.com/lovoo/nsq_exporter)
* [RabbitMQ exporter](https://github.com/kbudde/rabbitmq_exporter)
* [RabbitMQ Management Plugin exporter](https://github.com/deadtrickster/prometheus_rabbitmq_exporter)
* [Mirth Connect exporter](https://github.com/vynca/mirth_exporter)
* [MQTT blackbox exporter](https://github.com/inovex/mqtt_blackbox_exporter)

Storage

* [Ceph exporter](https://github.com/digitalocean/ceph_exporter)
* [ScaleIO exporter](https://github.com/syepes/sio2prom)
* [Gluster exporter](https://github.com/ofesseler/gluster_exporter)
* [Lustre exporter](https://github.com/HewlettPackard/lustre_exporter)

HTTP

* [Apache exporter](https://github.com/neezgee/apache_exporter)
* [HAProxy exporter](https://github.com/prometheus/haproxy_exporter) (**official**)
* [Nginx metric library](https://github.com/knyar/nginx-lua-prometheus)
* [Nginx VTS exporter](https://github.com/hnlq715/nginx-vts-exporter)
* [Passenger exporter](https://github.com/stuartnelson3/passenger_exporter)
* [Tinyproxy exporter](https://github.com/igzivkov/tinyproxy_exporter)
* [Varnish exporter](https://github.com/jonnenauha/prometheus_varnish_exporter)
* [WebDriver exporter](https://github.com/mattbostock/webdriver_exporter)

APIs

* [AWS ECS exporter](https://github.com/slok/ecs-exporter)
* [Cloudflare exporter](https://github.com/wehkamp/docker-prometheus-cloudflare-exporter)
* [DigitalOcean exporter](https://github.com/metalmatze/digitalocean_exporter)
* [Docker Cloud exporter](https://github.com/infinityworksltd/docker-cloud-exporter)
* [Docker Hub exporter](https://github.com/infinityworksltd/docker-hub-exporter)
* [GitHub exporter](https://github.com/infinityworksltd/github-exporter)
* [Mozilla Observatory exporter](https://github.com/Jimdo/observatory-exporter)
* [OpenWeatherMap exporter](https://github.com/RichiH/openweathermap_exporter)
* [Rancher exporter](https://github.com/infinityworksltd/prometheus-rancher-exporter)
* [Speedtest.net exporter](https://github.com/RichiH/speedtest_exporter)

Logging

* [Google's mtail log data extractor](https://github.com/google/mtail)
* [Grok exporter](https://github.com/fstab/grok_exporter)

Other monitoring systems

* [Akamai Cloudmonitor exporter](https://github.com/ExpressenAB/cloudmonitor_exporter)
* [AWS CloudWatch exporter](https://github.com/prometheus/cloudwatch_exporter) (**official**)
* [Cloud Foundry Firehose exporter](https://github.com/cloudfoundry-community/firehose_exporter)
* [Collectd exporter](https://github.com/prometheus/collectd_exporter) (**official**)
* [Graphite exporter](https://github.com/prometheus/graphite_exporter) (**official**)
* [Heka dashboard exporter](https://github.com/docker-infra/heka_exporter)
* [Heka exporter](https://github.com/imgix/heka_exporter)
* [InfluxDB exporter](https://github.com/prometheus/influxdb_exporter) (**official**)
* [JavaMelody exporter](https://github.com/fschlag/javamelody-prometheus-exporter)
* [JMX exporter](https://github.com/prometheus/jmx_exporter) (**official**)
* [Munin exporter](https://github.com/pvdh/munin_exporter)
* [New Relic exporter](https://github.com/jfindley/newrelic_exporter)
* [Pingdom exporter](https://github.com/giantswarm/prometheus-pingdom-exporter)
* [scollector exporter](https://github.com/tgulacsi/prometheus_scollector)
* [SNMP exporter](https://github.com/prometheus/snmp_exporter) (**official**)
* [StatsD exporter](https://github.com/prometheus/statsd_exporter) (**official**)

Miscellaneous

* [BIG-IP exporter](https://github.com/ExpressenAB/bigip_exporter)
* [BIND exporter](https://github.com/digitalocean/bind_exporter)
* [Blackbox exporter](https://github.com/prometheus/blackbox_exporter) (**official**)
* [BOSH exporter](https://github.com/cloudfoundry-community/bosh_exporter)
* [Dovecot exporter](https://github.com/kumina/dovecot_exporter)
* [Jenkins exporter](https://github.com/lovoo/jenkins_exporter)
* [Kemp LoadBalancer exporter](https://github.com/giantswarm/prometheus-kemp-exporter)
* [Meteor JS web framework exporter](https://atmospherejs.com/sevki/prometheus-exporter)
* [Minecraft exporter module](https://github.com/Baughn/PrometheusIntegration)
* [PowerDNS exporter](https://github.com/ledgr/powerdns_exporter)
* [Process exporter](https://github.com/ncabatoff/process-exporter)
* [rTorrent exporter](https://github.com/mdlayher/rtorrent_exporter)
* [Script exporter](https://github.com/adhocteam/script_exporter)
* [SMTP/Maildir MDA blackbox prober](https://github.com/cherti/mailexporter)
* [Transmission exporter](https://github.com/metalmatze/transmission-exporter)
* [Unbound exporter](https://github.com/kumina/unbound_exporter)
* [Xen exporter](https://github.com/lovoo/xenstats_exporter)

When implementing a new Prometheus exporter, please follow the [guidelines on writing exporters](https://prometheus.io/docs/instrumenting/writing_exporters) Please also consider consulting the [development mailing list](https://groups.google.com/forum/#!forum/prometheus-developers). We are happy to give advice on how to make your exporter as useful and consistent as possible.

**Directly instrumented software**

Some third-party software already exposes Prometheus metrics natively, so no separate exporters are needed:

* [cAdvisor](https://github.com/google/cadvisor)
* [Doorman](https://github.com/youtube/doorman)
* [Etcd](https://github.com/coreos/etcd)
* [Kubernetes-Mesos](https://github.com/mesosphere/kubernetes-mesos)
* [Kubernetes](https://github.com/kubernetes/kubernetes)
* [RobustIRC](http://robustirc.net/)
* [Quobyte](https://www.quobyte.com/)
* [SkyDNS](https://github.com/skynetservices/skydns)
* [Weave Flux](http://weaveworks.github.io/flux/)

**Other third-party utilities**

This section lists libraries and other utilities that help you instrument code in a certain language. They are not Prometheus client libraries themselves but make use of one of the normal Prometheus client libraries under the hood. As for all independently maintained software, we cannot vet all of them for best practices.

* Clojure: [prometheus-clj](https://github.com/soundcloud/prometheus-clj)
* Go: [go-metrics instrumentation library](https://github.com/armon/go-metrics)
* Go: [gokit](https://github.com/peterbourgon/gokit)
* Go: [prombolt](https://github.com/mdlayher/prombolt)
* Java/JVM: [Hystrix metrics publisher](https://github.com/soundcloud/prometheus-hystrix)
* Python-Django: [django-prometheus](https://github.com/korfuri/django-prometheus)

# Operating

## Configuration | Prometheus

https://prometheus.io/docs/operating/configuration/

**CONFIGURATION**

* [Configuration file](https://prometheus.io/docs/operating/configuration/#configuration-file)
  + [<scrape\_config>](https://prometheus.io/docs/operating/configuration/#<scrape_config>)
  + [<tls\_config>](https://prometheus.io/docs/operating/configuration/#<tls_config>)
  + [<azure\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<azure_sd_config>)
  + [<consul\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<consul_sd_config>)
  + [<dns\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<dns_sd_config>)
  + [<ec2\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<ec2_sd_config>)
  + [<file\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<file_sd_config>)
  + [<gce\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<gce_sd_config>)
  + [<kubernetes\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<kubernetes_sd_config>)
  + [<marathon\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<marathon_sd_config>)
  + [<nerve\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<nerve_sd_config>)
  + [<serverset\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<serverset_sd_config>)
  + [<triton\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<triton_sd_config>)
  + [<static\_config>](https://prometheus.io/docs/operating/configuration/#<static_config>)
  + [<relabel\_config>](https://prometheus.io/docs/operating/configuration/#<relabel_config>)
  + [<metric\_relabel\_configs>](https://prometheus.io/docs/operating/configuration/#<metric_relabel_configs>)
  + [<alert\_relabel\_configs>](https://prometheus.io/docs/operating/configuration/#<alert_relabel_configs>)
  + [<alertmanager\_config>](https://prometheus.io/docs/operating/configuration/#<alertmanager_config>)
  + [<remote\_write>](https://prometheus.io/docs/operating/configuration/#<remote_write>)
  + [<remote\_read>](https://prometheus.io/docs/operating/configuration/#<remote_read>)

Prometheus is configured via command-line flags and a configuration file. While the command-line flags configure immutable system parameters (such as storage locations, amount of data to keep on disk and in memory, etc.), the configuration file defines everything related to scraping [jobs and their instances](https://prometheus.io/docs/concepts/jobs_instances/), as well as which [rule files to load](https://prometheus.io/docs/querying/rules/#configuring-rules).

To view all available command-line flags, run prometheus -h.

Prometheus can reload its configuration at runtime. If the new configuration is not well-formed, the changes will not be applied. A configuration reload is triggered by sending a SIGHUP to the Prometheus process or sending a HTTP POST request to the/-/reload endpoint. This will also reload any configured rule files.

**Configuration file**

To specify which configuration file to load, use the -config.fileflag.

The file is written in [YAML format](http://en.wikipedia.org/wiki/YAML), defined by the scheme described below. Brackets indicate that a parameter is optional. For non-list parameters the value is set to the specified default.

Generic placeholders are defined as follows:

* <boolean>: a boolean that can take the values true or false
* <duration>: a duration matching the regular expression [0-9]+(ms|[smhdwy])
* <labelname>: a string matching the regular expression [a-zA-Z\_][a-zA-Z0-9\_]\*
* <labelvalue>: a string of unicode characters
* <filename>: a valid path in the current working directory
* <host>: a valid string consisting of a hostname or IP followed by an optional port number
* <path>: a valid URL path
* <scheme>: a string that can take the values http or https
* <string>: a regular string

The other placeholders are specified separately.

A valid example file can be found [here](https://github.com/prometheus/prometheus/blob/master/config/testdata/conf.good.yml).

The global configuration specifies parameters that are valid in all other configuration contexts. They also serve as defaults for other configuration sections.

global:

# How frequently to scrape targets by default.

[ scrape\_interval: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 1m ]

# How long until a scrape request times out.

[ scrape\_timeout: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 10s ]

# How frequently to evaluate rules.

[ evaluation\_interval: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 1m ]

# The labels to add to any time series or alerts when communicating with

# external systems (federation, remote storage, Alertmanager).

external\_labels:

[ [<labelname>](https://prometheus.io/docs/operating/configuration/#<labelname>): [<labelvalue>](https://prometheus.io/docs/operating/configuration/#<labelvalue>) ... ]

# Rule files specifies a list of globs. Rules and alerts are read from

# all matching files.

rule\_files:

[ - <filepath\_glob> ... ]

# A list of scrape configurations.

scrape\_configs:

[ - [<scrape\_config>](https://prometheus.io/docs/operating/configuration/#<scrape_config>) ... ]

# Alerting specifies settings related to the Alertmanager.

alerting:

alert\_relabel\_configs:

[ - [<relabel\_config>](https://prometheus.io/docs/operating/configuration/#<relabel_config>) ... ]

alertmanagers:

[ - [<alertmanager\_config>](https://prometheus.io/docs/operating/configuration/#<alertmanager_config>) ... ]

# Settings related to the experimental remote write feature.

remote\_write:

[ - [<remote\_write>](https://prometheus.io/docs/operating/configuration/#<remote_write>) ... ]

# Settings related to the experimental remote read feature.

remote\_read:

[ - [<remote\_read>](https://prometheus.io/docs/operating/configuration/#<remote_read>) ... ]

### <scrape\_config>

A scrape\_config section specifies a set of targets and parameters describing how to scrape them. In the general case, one scrape configuration specifies a single job. In advanced configurations, this may change.

Targets may be statically configured via the static\_configs parameter or dynamically discovered using one of the supported service-discovery mechanisms.

Additionally, relabel\_configs allow advanced modifications to any target and its labels before scraping.

# The job name assigned to scraped metrics by default.

job\_name: [<job\_name>](https://prometheus.io/docs/operating/configuration/#<job_name>)

# How frequently to scrape targets from this job.

[ scrape\_interval: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = <global\_config.scrape\_interval> ]

# Per-scrape timeout when scraping this job.

[ scrape\_timeout: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = <global\_config.scrape\_timeout> ]

# The HTTP resource path on which to fetch metrics from targets.

[ metrics\_path: [<path>](https://prometheus.io/docs/operating/configuration/#<path>) | default = /metrics ]

# honor\_labels controls how Prometheus handles conflicts between labels that are

# already present in scraped data and labels that Prometheus would attach

# server-side ("job" and "instance" labels, manually configured target

# labels, and labels generated by service discovery implementations).

#

# If honor\_labels is set to "true", label conflicts are resolved by keeping label

# values from the scraped data and ignoring the conflicting server-side labels.

#

# If honor\_labels is set to "false", label conflicts are resolved by renaming

# conflicting labels in the scraped data to "exported\_<original-label>" (for

# example "exported\_instance", "exported\_job") and then attaching server-side

# labels. This is useful for use cases such as federation, where all labels

# specified in the target should be preserved.

#

# Note that any globally configured "external\_labels" are unaffected by this

# setting. In communication with external systems, they are always applied only

# when a time series does not have a given label yet and are ignored otherwise.

[ honor\_labels: [<boolean>](https://prometheus.io/docs/operating/configuration/#<boolean>) | default = false ]

# Configures the protocol scheme used for requests.

[ scheme: [<scheme>](https://prometheus.io/docs/operating/configuration/#<scheme>) | default = http ]

# Optional HTTP URL parameters.

params:

[ [<string>](https://prometheus.io/docs/operating/configuration/#<string>): [[<string>](https://prometheus.io/docs/operating/configuration/#<string>), ...] ]

# Sets the `Authorization` header on every scrape request with the

# configured username and password.

basic\_auth:

[ username: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

[ password: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Sets the `Authorization` header on every scrape request with

# the configured bearer token. It is mutually exclusive with `bearer\_token\_file`.

[ bearer\_token: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Sets the `Authorization` header on every scrape request with the bearer token

# read from the configured file. It is mutually exclusive with `bearer\_token`.

[ bearer\_token\_file: /path/to/bearer/token/file ]

# Configures the scrape request's TLS settings.

tls\_config:

[ [<tls\_config>](https://prometheus.io/docs/operating/configuration/#<tls_config>) ]

# Optional proxy URL.

[ proxy\_url: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# List of Azure service discovery configurations.

azure\_sd\_configs:

[ - [<azure\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<azure_sd_config>) ... ]

# List of Consul service discovery configurations.

consul\_sd\_configs:

[ - [<consul\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<consul_sd_config>) ... ]

# List of DNS service discovery configurations.

dns\_sd\_configs:

[ - [<dns\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<dns_sd_config>) ... ]

# List of EC2 service discovery configurations.

ec2\_sd\_configs:

[ - [<ec2\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<ec2_sd_config>) ... ]

# List of file service discovery configurations.

file\_sd\_configs:

[ - [<file\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<file_sd_config>) ... ]

# List of GCE service discovery configurations.

gce\_sd\_configs:

[ - [<gce\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<gce_sd_config>) ... ]

# List of Kubernetes service discovery configurations.

kubernetes\_sd\_configs:

[ - [<kubernetes\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<kubernetes_sd_config>) ... ]

# List of Marathon service discovery configurations.

marathon\_sd\_configs:

[ - [<marathon\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<marathon_sd_config>) ... ]

# List of AirBnB's Nerve service discovery configurations.

nerve\_sd\_configs:

[ - [<nerve\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<nerve_sd_config>) ... ]

# List of Zookeeper Serverset service discovery configurations.

serverset\_sd\_configs:

[ - [<serverset\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<serverset_sd_config>) ... ]

# List of Triton service discovery configurations.

triton\_sd\_configs:

[ - [<triton\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<triton_sd_config>) ... ]

# List of labeled statically configured targets for this job.

static\_configs:

[ - [<static\_config>](https://prometheus.io/docs/operating/configuration/#<static_config>) ... ]

# List of target relabel configurations.

relabel\_configs:

[ - [<relabel\_config>](https://prometheus.io/docs/operating/configuration/#<relabel_config>) ... ]

# List of metric relabel configurations.

metric\_relabel\_configs:

[ - [<relabel\_config>](https://prometheus.io/docs/operating/configuration/#<relabel_config>) ... ]

# Per-scrape limit on number of scraped samples that will be accepted.

# If more than this number of samples are present after metric relabelling

# the entire scrape will be treated as failed. 0 means no limit.

[ sample\_limit: <int> | default = 0 ]

Where <job\_name> must be unique across all scrape configurations.

### <tls\_config>

A tls\_config allows configuring TLS connections.

# CA certificate to validate API server certificate with.

[ ca\_file: [<filename>](https://prometheus.io/docs/operating/configuration/#<filename>) ]

# Certificate and key files for client cert authentication to the server.

[ cert\_file: [<filename>](https://prometheus.io/docs/operating/configuration/#<filename>) ]

[ key\_file: [<filename>](https://prometheus.io/docs/operating/configuration/#<filename>) ]

# ServerName extension to indicate the name of the server.

# http://tools.ietf.org/html/rfc4366#section-3.1

[ server\_name: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Disable validation of the server certificate.

[ insecure\_skip\_verify: [<boolean>](https://prometheus.io/docs/operating/configuration/#<boolean>) ]

### <azure\_sd\_config>

Azure SD is in beta: breaking changes to configuration are still likely in future releases.

Azure SD configurations allow retrieving scrape targets from Azure VMs.

The following meta labels are available on targets during relabeling:

* \_\_meta\_azure\_machine\_id: the machine ID
* \_\_meta\_azure\_machine\_location: the location the machine runs in
* \_\_meta\_azure\_machine\_name: the machine name
* \_\_meta\_azure\_machine\_private\_ip: the machine's private IP
* \_\_meta\_azure\_machine\_resource\_group: the machine's resource group
* \_\_meta\_azure\_tag\_<tagname>: each tag value of the machine

See below for the configuration options for Azure discovery:

# The information to access the Azure API.

# The subscription ID.

subscription\_id: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# The tenant ID.

tenant\_id: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# The client ID.

client\_id: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# The client secret.

client\_secret: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# Refresh interval to re-read the instance list.

[ refresh\_interval: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 300s ]

# The port to scrape metrics from. If using the public IP address, this must

# instead be specified in the relabeling rule.

[ port: <int> | default = 80 ]

<consul\_sd\_config>

Consul SD configurations allow retrieving scrape targets from [Consul's](https://www.consul.io/) Catalog API.

The following meta labels are available on targets during [relabeling](https://prometheus.io/docs/operating/configuration/#relabel_config):

* \_\_meta\_consul\_address: the address of the target
* \_\_meta\_consul\_dc: the datacenter name for the target
* \_\_meta\_consul\_node: the node name defined for the target
* \_\_meta\_consul\_service\_address: the service address of the target
* \_\_meta\_consul\_service\_id: the service ID of the target
* \_\_meta\_consul\_service\_port: the service port of the target
* \_\_meta\_consul\_service: the name of the service the target belongs to
* \_\_meta\_consul\_tags: the list of tags of the target joined by the tag separator

# The information to access the Consul API. It is to be defined

# as the Consul documentation requires.

server: [<host>](https://prometheus.io/docs/operating/configuration/#<host>)

[ token: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

[ datacenter: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

[ scheme: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

[ username: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

[ password: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# A list of services for which targets are retrieved. If omitted, all services

# are scraped.

services:

[ - [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# The string by which Consul tags are joined into the tag label.

[ tag\_separator: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) | default = , ]

Note that the IP number and port used to scrape the targets is assembled as <\_\_meta\_consul\_address>:<\_\_meta\_consul\_service\_port>. However, in some Consul setups, the relevant address is in\_\_meta\_consul\_service\_address. In those cases, you can use the [relabel](https://prometheus.io/docs/operating/configuration/#relabel_config) feature to replace the special\_\_address\_\_ label.

### <dns\_sd\_config>

A DNS-based service discovery configuration allows specifying a set of DNS domain names which are periodically queried to discover a list of targets. The DNS servers to be contacted are read from /etc/resolv.conf.

This service discovery method only supports basic DNS A, AAAA and SRV record queries, but not the advanced DNS-SD approach specified in [RFC6763](https://tools.ietf.org/html/rfc6763).

During the [relabeling phase](https://prometheus.io/docs/operating/configuration/#relabel_config), the meta label \_\_meta\_dns\_name is available on each target and is set to the record name that produced the discovered target.

# A list of DNS domain names to be queried.

names:

[ - [<domain\_name>](https://prometheus.io/docs/operating/configuration/#<domain_name>) ]

# The type of DNS query to perform.

[ type: [<query\_type>](https://prometheus.io/docs/operating/configuration/#<query_type>) | default = 'SRV' ]

# The port number used if the query type is not SRV.

[ port: <number>]

# The time after which the provided names are refreshed.

[ refresh\_interval: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 30s ]

Where <domain\_name> is a valid DNS domain name. Where <query\_type> is SRV, A, or AAAA.

### <ec2\_sd\_config>

EC2 SD configurations allow retrieving scrape targets from AWS EC2 instances. The private IP address is used by default, but may be changed to the public IP address with relabeling.

The following meta labels are available on targets during [relabeling](https://prometheus.io/docs/operating/configuration/#relabel_config):

* \_\_meta\_ec2\_availability\_zone: the availability zone in which the instance is running
* \_\_meta\_ec2\_instance\_id: the EC2 instance ID
* \_\_meta\_ec2\_instance\_state: the state of the EC2 instance
* \_\_meta\_ec2\_instance\_type: the type of the EC2 instance
* \_\_meta\_ec2\_private\_ip: the private IP address of the instance, if present
* \_\_meta\_ec2\_public\_dns\_name: the public DNS name of the instance, if available
* \_\_meta\_ec2\_public\_ip: the public IP address of the instance, if available
* \_\_meta\_ec2\_subnet\_id: comma separated list of subnets IDs in which the instance is running, if available
* \_\_meta\_ec2\_tag\_<tagkey>: each tag value of the instance
* \_\_meta\_ec2\_vpc\_id: the ID of the VPC in which the instance is running, if available

See below for the configuration options for EC2 discovery:

# The information to access the EC2 API.

# The AWS Region.

region: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# The AWS API keys. If blank, the environment variables `AWS\_ACCESS\_KEY\_ID`

# and `AWS\_SECRET\_ACCESS\_KEY` are used.

[ access\_key: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

[ secret\_key: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Named AWS profile used to connect to the API.

[ profile: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Refresh interval to re-read the instance list.

[ refresh\_interval: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 60s ]

# The port to scrape metrics from. If using the public IP address, this must

# instead be specified in the relabeling rule.

[ port: <int> | default = 80 ]

### <file\_sd\_config>

File-based service discovery provides a more generic way to configure static targets and serves as an interface to plug in custom service discovery mechanisms.

It reads a set of files containing a list of zero or more <static\_config>s. Changes to all defined files are detected via disk watches and applied immediately. Files may be provided in YAML or JSON format. Only changes resulting in well-formed target groups are applied.

The JSON file must contain a list of static configs, using this format:

[

{

"targets": [ "[<host>](https://prometheus.io/docs/operating/configuration/#<host>)", ... ],

"labels": {

"[<labelname>](https://prometheus.io/docs/operating/configuration/#<labelname>)": "[<labelvalue>](https://prometheus.io/docs/operating/configuration/#<labelvalue>)", ...

}

},

...

]

As a fallback, the file contents are also re-read periodically at the specified refresh interval.

Each target has a meta label \_\_meta\_filepath during the [relabeling phase](https://prometheus.io/docs/operating/configuration/#relabel_config). Its value is set to the filepath from which the target was extracted.

# Patterns for files from which target groups are extracted.

files:

[ - [<filename\_pattern>](https://prometheus.io/docs/operating/configuration/#<filename_pattern>) ... ]

# Refresh interval to re-read the files.

[ refresh\_interval: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 5m ]

Where <filename\_pattern> may be a path ending in .json, .yml or .yaml. The last path segment may contain a single \* that matches any character sequence, e.g. my/path/tg\_\*.json.

### <gce\_sd\_config>

GCE SD is in beta: breaking changes to configuration are still likely in future releases.

[GCE](https://cloud.google.com/compute/) SD configurations allow retrieving scrape targets from GCP GCE instances. The private IP address is used by default, but may be changed to the public IP address with relabeling.

The following meta labels are available on targets during [relabeling](https://prometheus.io/docs/operating/configuration/#relabel_config):

* \_\_meta\_gce\_instance\_name: the name of the instance
* \_\_meta\_gce\_metadata\_<name>: each metadata item of the instance
* \_\_meta\_gce\_network: the network of the instance
* \_\_meta\_gce\_private\_ip: the private IP address of the instance
* \_\_meta\_gce\_project: the GCP project in which the instance is running
* \_\_meta\_gce\_public\_ip: the public IP address of the instance, if present
* \_\_meta\_gce\_subnetwork: the subnetwork of the instance
* \_\_meta\_gce\_tags: comma separated list of instance tags
* \_\_meta\_gce\_zone: the GCE zone in which the instance is running

See below for the configuration options for GCE discovery:

# The information to access the GCE API.

# The GCP Project

project: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# The zone of the scrape targets. If you need multiple zones use multiple

# gce\_sd\_configs.

zone: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# Filter can be used optionally to filter the instance list by other criteria

[ filter: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Refresh interval to re-read the instance list

[ refresh\_interval: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 60s ]

# The port to scrape metrics from. If using the public IP address, this must

# instead be specified in the relabeling rule.

[ port: <int> | default = 80 ]

# The tag separator is used to separate the tags on concatenation

[ tag\_separator: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) | default = , ]

Credentials are discovered by the Google Cloud SDK default client by looking in the following places, preferring the first location found:

1. a JSON file specified by the GOOGLE\_APPLICATION\_CREDENTIALS environment variable
2. a JSON file in the well-known path $HOME/.config/gcloud/application\_default\_credentials.json
3. fetched from the GCE metadata server

If Prometheus is running within GCE, the service account associated with the instance it is running on should have at least read-only permissions to the compute resources. If running outside of GCE make sure to create an appropriate service account and place the credential file in one of the expected locations.

### <kubernetes\_sd\_config>

Kubernetes SD is in beta: breaking changes to configuration are still likely in future releases.

Kubernetes SD configurations allow retrieving scrape targets from [Kubernetes'](http://kubernetes.io/) REST API and always staying synchronized with the cluster state.

One of the following role types can be configured to discover targets:

**node**

The node role discovers one target per cluster node with the address defaulting to the Kubelet's HTTP port. The target address defaults to the first existing address of the Kubernetes node object in the address type order ofNodeInternalIP, NodeExternalIP, NodeLegacyHostIP, and NodeHostName.

Available meta labels:

* \_\_meta\_kubernetes\_node\_name: The name of the node object.
* \_\_meta\_kubernetes\_node\_label\_<labelname>: Each label from the node object.
* \_\_meta\_kubernetes\_node\_annotation\_<annotationname>: Each annotation from the node object.
* \_\_meta\_kubernetes\_node\_address\_<address\_type>: The first address for each node address type, if it exists.

In addition, the instance label for the node will be set to the node name as retrieved from the API server.

**service**

The service role discovers a target for each service port for each service. This is generally useful for blackbox monitoring of a service. The address will be set to the Kubernetes DNS name of the service and respective service port.

Available meta labels:

* \_\_meta\_kubernetes\_namespace: The namespace of the service object.
* \_\_meta\_kubernetes\_service\_name: The name of the service object.
* \_\_meta\_kubernetes\_service\_label\_<labelname>: The label of the service object.
* \_\_meta\_kubernetes\_service\_annotation\_<annotationname>: The annotation of the service object.
* \_\_meta\_kubernetes\_service\_port\_name: Name of the service port for the target.
* \_\_meta\_kubernetes\_service\_port\_number: Number of the service port for the target.
* \_\_meta\_kubernetes\_service\_port\_portocol: Protocol of the service port for the target.

**pod**

The pod role discovers all pods and exposes their containers as targets. For each declared port of a container, a single target is generated. If a container has no specified ports, a port-free target per container is created for manually adding a port via relabeling.

Available meta labels:

* \_\_meta\_kubernetes\_namespace: The namespace of the pod object.
* \_\_meta\_kubernetes\_pod\_name: The name of the pod object.
* \_\_meta\_kubernetes\_pod\_ip: The pod IP of the pod object.
* \_\_meta\_kubernetes\_pod\_label\_<labelname>: The label of the pod object.
* \_\_meta\_kubernetes\_pod\_annotation\_<annotationname>: The annotation of the pod object.
* \_\_meta\_kubernetes\_pod\_container\_name: Name of the container the target address points to.
* \_\_meta\_kubernetes\_pod\_container\_port\_name: Name of the container port.
* \_\_meta\_kubernetes\_pod\_container\_port\_number: Number of the container port.
* \_\_meta\_kubernetes\_pod\_container\_port\_protocol: Protocol of the container port.
* \_\_meta\_kubernetes\_pod\_ready: Set to true or false for the pod's ready state.
* \_\_meta\_kubernetes\_pod\_node\_name: The name of the node the pod is scheduled onto.
* \_\_meta\_kubernetes\_pod\_host\_ip: The current host IP of the pod object.

**endpoints**

The endpoints role discovers targets from listed endpoints of a service. For each endpoint address one target is discovered per port. If the endpoint is backed by a pod, all additional container ports of the pod, not bound to an endpoint port, are discovered as targets as well.

Available meta labels:

* \_\_meta\_kubernetes\_namespace: The namespace of the endpoints object.
* \_\_meta\_kubernetes\_endpoints\_name: The names of the endpoints object.
* For all targets discovered directly from the endpoints list (those not additionally inferred from underlying pods), the following labels are attached:
  + \_\_meta\_kubernetes\_endpoint\_ready: Set to true or false for the endpoint's ready state.
  + \_\_meta\_kubernetes\_endpoint\_port\_name: Name of the endpoint port.
  + \_\_meta\_kubernetes\_endpoint\_port\_protocol: Protocol of the endpoint port.
* If the endpoints belong to a service, all labels of the role: service discovery are attached.
* For all targets backed by a pod, all labels of the role: pod discovery are attached.

See below for the configuration options for Kubernetes discovery:

# The information to access the Kubernetes API.

# The API server addresses. If left empty, Prometheus is assumed to run inside

# of the cluster and will discover API servers automatically and use the pod's

# CA certificate and bearer token file at /var/run/secrets/kubernetes.io/serviceaccount/.

[ api\_server: [<host>](https://prometheus.io/docs/operating/configuration/#<host>) ]

# The Kubernetes role of entities that should be discovered.

role: [<role>](https://prometheus.io/docs/operating/configuration/#<role>)

# Optional authentication information used to authenticate to the API server.

# Note that `basic\_auth`, `bearer\_token` and `bearer\_token\_file` options are

# mutually exclusive.

# Optional HTTP basic authentication information.

basic\_auth:

[ username: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

[ password: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Optional bearer token authentication information.

[ bearer\_token: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Optional bearer token file authentication information.

[ bearer\_token\_file: [<filename>](https://prometheus.io/docs/operating/configuration/#<filename>) ]

# TLS configuration.

tls\_config:

[ [<tls\_config>](https://prometheus.io/docs/operating/configuration/#<tls_config>) ]

Where <role> must be endpoints, service, pod, or node.

See [this example Prometheus configuration file](https://github.com/prometheus/prometheus/blob/master/documentation/examples/prometheus-kubernetes.yml) for a detailed example of configuring Prometheus for Kubernetes.

You may wish to check out the 3rd party [Prometheus Operator](https://github.com/coreos/prometheus-operator), which automates the Prometheus setup on top of Kubernetes.

### <marathon\_sd\_config>

Marathon SD is in beta: breaking changes to configuration are still likely in future releases.

Marathon SD configurations allow retrieving scrape targets using the [Marathon](https://mesosphere.github.io/marathon/) REST API. Prometheus will periodically check the REST endpoint for currently running tasks and create a target group for every app that has at least one healthy task.

The following meta labels are available on targets during [relabeling](https://prometheus.io/docs/operating/configuration/#relabel_config):

* \_\_meta\_marathon\_app: the name of the app (with slashes replaced by dashes)
* \_\_meta\_marathon\_image: the name of the Docker image used (if available)
* \_\_meta\_marathon\_task: the ID of the Mesos task
* \_\_meta\_marathon\_app\_label\_<labelname>: any Marathon labels attached to the app

See below for the configuration options for Marathon discovery:

# List of URLs to be used to contact Marathon servers.

# You need to provide at least one server URL, but should provide URLs for

# all masters you have running.

servers:

- [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# Optional bearer token authentication information.

# It is mutually exclusive with `bearer\_token\_file`.

[ bearer\_token: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Optional bearer token file authentication information.

# It is mutually exclusive with `bearer\_token`.

[ bearer\_token\_file: [<filename>](https://prometheus.io/docs/operating/configuration/#<filename>) ]

# Polling interval

[ refresh\_interval: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 30s ]

By default every app listed in Marathon will be scraped by Prometheus. If not all of your services provide Prometheus metrics, you can use a Marathon label and Prometheus relabeling to control which instances will actually be scraped. Also by default all apps will show up as a single job in Prometheus (the one specified in the configuration file), which can also be changed using relabeling.

### <nerve\_sd\_config>

Nerve SD configurations allow retrieving scrape targets from [AirBnB's Nerve](https://github.com/airbnb/nerve) which are stored in [Zookeeper](https://zookeeper.apache.org/).

The following meta labels are available on targets during [relabeling](https://prometheus.io/docs/operating/configuration/#relabel_config):

* \_\_meta\_nerve\_path: the full path to the endpoint node in Zookeeper
* \_\_meta\_nerve\_endpoint\_host: the host of the endpoint
* \_\_meta\_nerve\_endpoint\_port: the port of the endpoint
* \_\_meta\_nerve\_endpoint\_name: the name of the endpoint

# The Zookeeper servers.

servers:

- [<host>](https://prometheus.io/docs/operating/configuration/#<host>)

# Paths can point to a single service, or the root of a tree of services.

paths:

- [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

[ timeout: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 10s ]

### <serverset\_sd\_config>

Serverset SD configurations allow retrieving scrape targets from [Serversets](https://github.com/twitter/finagle/tree/master/finagle-serversets) which are stored in [Zookeeper](https://zookeeper.apache.org/). Serversets are commonly used by [Finagle](https://twitter.github.io/finagle/) and [Aurora](http://aurora.apache.org/).

The following meta labels are available on targets during relabeling:

* \_\_meta\_serverset\_path: the full path to the serverset member node in Zookeeper
* \_\_meta\_serverset\_endpoint\_host: the host of the default endpoint
* \_\_meta\_serverset\_endpoint\_port: the port of the default endpoint
* \_\_meta\_serverset\_endpoint\_host\_<endpoint>: the host of the given endpoint
* \_\_meta\_serverset\_endpoint\_port\_<endpoint>: the port of the given endpoint
* \_\_meta\_serverset\_shard: the shard number of the member
* \_\_meta\_serverset\_status: the status of the member

# The Zookeeper servers.

servers:

- [<host>](https://prometheus.io/docs/operating/configuration/#<host>)

# Paths can point to a single serverset, or the root of a tree of serversets.

paths:

- [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

[ timeout: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 10s ]

Serverset data must be in the JSON format, the Thrift format is not currently supported.

### <triton\_sd\_config>

Triton SD is in beta: breaking changes to configuration are still likely in future releases.

[Triton](https://github.com/joyent/triton) SD configurations allow retrieving scrape targets from [Container Monitor](https://github.com/joyent/rfd/blob/master/rfd/0027/README.md) discovery endpoints.

The following meta labels are available on targets during relabeling:

* \_\_meta\_triton\_machine\_id: the UUID of the target container
* \_\_meta\_triton\_machine\_alias: the alias of the target container
* \_\_meta\_triton\_machine\_image: the target containers image type
* \_\_meta\_triton\_machine\_server\_id: the server UUID for the target container

# The information to access the Triton discovery API.

# The account to use for discovering new target containers.

account: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# The DNS suffix which should be applied to target containers.

dns\_suffix: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# The Triton discovery endpoint (e.g. 'cmon.us-east-3b.triton.zone'). This is

# often the same value as dns\_suffix.

endpoint: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# The port to use for discovery and metric scraping.

[ port: <int> | default = 9163 ]

# The interval which should should be used for refreshing target containers.

[ refresh\_interval: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 60s ]

# The Triton discovery API version.

[ version: <int> | default = 1 ]

# TLS configuration.

tls\_config:

[ [<tls\_config>](https://prometheus.io/docs/operating/configuration/#<tls_config>) ]

<static\_config>

A static\_config allows specifying a list of targets and a common label set for them. It is the canonical way to specify static targets in a scrape configuration.

# The targets specified by the static config.

targets:

[ - '[<host>](https://prometheus.io/docs/operating/configuration/#<host>)' ]

# Labels assigned to all metrics scraped from the targets.

labels:

[ [<labelname>](https://prometheus.io/docs/operating/configuration/#<labelname>): [<labelvalue>](https://prometheus.io/docs/operating/configuration/#<labelvalue>) ... ]

### <relabel\_config>

Relabeling is a powerful tool to dynamically rewrite the label set of a target before it gets scraped. Multiple relabeling steps can be configured per scrape configuration. They are applied to the label set of each target in order of their appearance in the configuration file.

Initially, aside from the configured per-target labels, a target's job label is set to the job\_name value of the respective scrape configuration. The \_\_address\_\_ label is set to the <host>:<port> address of the target. After relabeling, the instance label is set to the value of \_\_address\_\_ by default if it was not set during relabeling. The\_\_scheme\_\_ and \_\_metrics\_path\_\_ labels are set to the scheme and metrics path of the target respectively. The\_\_param\_<name> label is set to the value of the first passed URL parameter called <name>.

Additional labels prefixed with \_\_meta\_ may be available during the relabeling phase. They are set by the service discovery mechanism that provided the target and vary between mechanisms.

Labels starting with \_\_ will be removed from the label set after relabeling is completed.

If a relabeling step needs to store a label value only temporarily (as the input to a subsequent relabeling step), use the \_\_tmp label name prefix. This prefix is guaranteed to never be used by Prometheus itself.

# The source labels select values from existing labels. Their content is concatenated

# using the configured separator and matched against the configured regular expression

# for the replace, keep, and drop actions.

[ source\_labels: '[' [<labelname>](https://prometheus.io/docs/operating/configuration/#<labelname>) [, ...] ']' ]

# Separator placed between concatenated source label values.

[ separator: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) | default = ; ]

# Label to which the resulting value is written in a replace action.

# It is mandatory for replace actions. Regex capture groups are available.

[ target\_label: [<labelname>](https://prometheus.io/docs/operating/configuration/#<labelname>) ]

# Regular expression against which the extracted value is matched.

[ regex: [<regex>](https://prometheus.io/docs/operating/configuration/#<regex>) | default = (.\*) ]

# Modulus to take of the hash of the source label values.

[ modulus: <uint64> ]

# Replacement value against which a regex replace is performed if the

# regular expression matches. Regex capture groups are available.

[ replacement: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) | default = $1 ]

# Action to perform based on regex matching.

[ action: [<relabel\_action>](https://prometheus.io/docs/operating/configuration/#<relabel_action>) | default = replace ]

<regex> is any valid [RE2 regular expression](https://github.com/google/re2/wiki/Syntax). It is required for the replace, keep, drop, labelmap,labeldropand labelkeep actions. The regex is anchored on both ends. To un-anchor the regex, use .\*<regex>.\*.

<relabel\_action> determines the relabeling action to take:

* replace: Match regex against the concatenated source\_labels. Then, set target\_label to replacement, with match group references (${1}, ${2}, ...) in replacement substituted by their value. If regex does not match, no replacement takes place.
* keep: Drop targets for which regex does not match the concatenated source\_labels.
* drop: Drop targets for which regex matches the concatenated source\_labels.
* hashmod: Set target\_label to the modulus of a hash of the concatenated source\_labels.
* labelmap: Match regex against all label names. Then copy the values of the matching labels to label names given by replacement with match group references (${1}, ${2}, ...) in replacement substituted by their value.
* labeldrop: Match regex against all label names. Any label that matches will be removed from the set of labels.
* labelkeep: Match regex against all label names. Any label that does not match will be removed from the set of labels.

Care must be taken with labeldrop and labelkeep to ensure that metrics are still uniquely labeled once the labels are removed.

### <metric\_relabel\_configs>

Metric relabeling is applied to samples as the last step before ingestion. It has the same configuration format and actions as target relabeling. Metric relabeling does not apply to automatically generated timeseries such as up.

One use for this is to blacklist time series that are too expensive to ingest.

<alert\_relabel\_configs>

Alert relabeling is applied to alerts before they are sent to the Alertmanager. It has the same configuration format and actions as target relabeling. Alert relabeling is applied after external labels.

One use for this is ensuring a HA pair of Prometheus servers with different external labels send identical alerts.

<alertmanager\_config>

Dynamic discovery of Alertmanager instances is in alpha state. Breaking configuration changes may happen in future releases. Use static configuration via the -alertmanager.url flag as a stable alternative.

An alertmanager\_config section specifies Alertmanager instances the Prometheus server sends alerts to. It also provides parameters to configure how to communicate with these Alertmanagers.

Alertmanagers may be statically configured via the static\_configs parameter or dynamically discovered using one of the supported service-discovery mechanisms.

Additionally, relabel\_configs allow selecting Alertmanagers from discovered entities and provide advanced modifications to the used API path, which is exposed through the \_\_alerts\_path\_\_ label.

# Per-target Alertmanager timeout when pushing alerts.

[ timeout: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 10s ]

# Prefix for the HTTP path alerts are pushed to.

[ path\_prefix: [<path>](https://prometheus.io/docs/operating/configuration/#<path>) | default = / ]

# Configures the protocol scheme used for requests.

[ scheme: [<scheme>](https://prometheus.io/docs/operating/configuration/#<scheme>) | default = http ]

# Sets the `Authorization` header on every request with the

# configured username and password.

basic\_auth:

[ username: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

[ password: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Sets the `Authorization` header on every request with

# the configured bearer token. It is mutually exclusive with `bearer\_token\_file`.

[ bearer\_token: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Sets the `Authorization` header on every request with the bearer token

# read from the configured file. It is mutually exclusive with `bearer\_token`.

[ bearer\_token\_file: /path/to/bearer/token/file ]

# Configures the scrape request's TLS settings.

tls\_config:

[ [<tls\_config>](https://prometheus.io/docs/operating/configuration/#<tls_config>) ]

# Optional proxy URL.

[ proxy\_url: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# List of Azure service discovery configurations.

azure\_sd\_configs:

[ - [<azure\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<azure_sd_config>) ... ]

# List of Consul service discovery configurations.

consul\_sd\_configs:

[ - [<consul\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<consul_sd_config>) ... ]

# List of DNS service discovery configurations.

dns\_sd\_configs:

[ - [<dns\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<dns_sd_config>) ... ]

# List of EC2 service discovery configurations.

ec2\_sd\_configs:

[ - [<ec2\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<ec2_sd_config>) ... ]

# List of file service discovery configurations.

file\_sd\_configs:

[ - [<file\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<file_sd_config>) ... ]

# List of GCE service discovery configurations.

gce\_sd\_configs:

[ - [<gce\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<gce_sd_config>) ... ]

# List of Kubernetes service discovery configurations.

kubernetes\_sd\_configs:

[ - [<kubernetes\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<kubernetes_sd_config>) ... ]

# List of Marathon service discovery configurations.

marathon\_sd\_configs:

[ - [<marathon\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<marathon_sd_config>) ... ]

# List of AirBnB's Nerve service discovery configurations.

nerve\_sd\_configs:

[ - [<nerve\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<nerve_sd_config>) ... ]

# List of Zookeeper Serverset service discovery configurations.

serverset\_sd\_configs:

[ - [<serverset\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<serverset_sd_config>) ... ]

# List of Triton service discovery configurations.

triton\_sd\_configs:

[ - [<triton\_sd\_config>](https://prometheus.io/docs/operating/configuration/#<triton_sd_config>) ... ]

# List of labeled statically configured Alertmanagers.

static\_configs:

[ - [<static\_config>](https://prometheus.io/docs/operating/configuration/#<static_config>) ... ]

# List of Alertmanager relabel configurations.

relabel\_configs:

[ - [<relabel\_config>](https://prometheus.io/docs/operating/configuration/#<relabel_config>) ... ]

<remote\_write>

Remote write is experimental: breaking changes to configuration are likely in future releases.

write\_relabel\_configs is relabeling applied to samples before sending them to the remote endpoint. Write relabeling is applied after external labels. This could be used to limit which samples are sent.

There is a [small demo](https://github.com/prometheus/prometheus/tree/master/documentation/examples/remote_storage) of how to use this functionality.

# The URL of the endpoint to send samples to.

url: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# Timeout for requests to the remote write endpoint.

[ remote\_timeout: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 30s ]

# List of remote write relabel configurations.

write\_relabel\_configs:

[ - [<relabel\_config>](https://prometheus.io/docs/operating/configuration/#<relabel_config>) ... ]

# Sets the `Authorization` header on every remote write request with the

# configured username and password.

basic\_auth:

[ username: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

[ password: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Sets the `Authorization` header on every remote write request with

# the configured bearer token. It is mutually exclusive with `bearer\_token\_file`.

[ bearer\_token: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Sets the `Authorization` header on every remote write request with the bearer token

# read from the configured file. It is mutually exclusive with `bearer\_token`.

[ bearer\_token\_file: /path/to/bearer/token/file ]

# Configures the remote write request's TLS settings.

tls\_config:

[ [<tls\_config>](https://prometheus.io/docs/operating/configuration/#<tls_config>) ]

# Optional proxy URL.

[ proxy\_url: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

<remote\_read>

Remote read is experimental: breaking changes to configuration are likely in future releases.

# The URL of the endpoint to query from.

url: [<string>](https://prometheus.io/docs/operating/configuration/#<string>)

# Timeout for requests to the remote read endpoint.

[ remote\_timeout: [<duration>](https://prometheus.io/docs/operating/configuration/#<duration>) | default = 30s ]

# Sets the `Authorization` header on every remote read request with the

# configured username and password.

basic\_auth:

[ username: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

[ password: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Sets the `Authorization` header on every remote read request with

# the configured bearer token. It is mutually exclusive with `bearer\_token\_file`.

[ bearer\_token: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

# Sets the `Authorization` header on every remote read request with the bearer token

# read from the configured file. It is mutually exclusive with `bearer\_token`.

[ bearer\_token\_file: /path/to/bearer/token/file ]

# Configures the remote read request's TLS settings.

tls\_config:

[ [<tls\_config>](https://prometheus.io/docs/operating/configuration/#<tls_config>) ]

# Optional proxy URL.

[ proxy\_url: [<string>](https://prometheus.io/docs/operating/configuration/#<string>) ]

## Storage | Prometheus

https://prometheus.io/docs/operating/storage/

**STORAGE**

* [Memory usage](https://prometheus.io/docs/operating/storage/#memory-usage)
* [Disk usage](https://prometheus.io/docs/operating/storage/#disk-usage)
* [Chunk encoding](https://prometheus.io/docs/operating/storage/#chunk-encoding)
* [Settings for high numbers of time series](https://prometheus.io/docs/operating/storage/#settings-for-high-numbers-of-time-series)
* [Persistence urgency and “rushed mode”](https://prometheus.io/docs/operating/storage/#persistence-urgency-and-%E2%80%9Crushed-mode%E2%80%9D)
* [Settings for very long retention time](https://prometheus.io/docs/operating/storage/#settings-for-very-long-retention-time)
* [Crash recovery](https://prometheus.io/docs/operating/storage/#crash-recovery)
* [Data corruption](https://prometheus.io/docs/operating/storage/#data-corruption)
* [Helpful metrics](https://prometheus.io/docs/operating/storage/#helpful-metrics)

Prometheus has a sophisticated local storage subsystem. For indexes, it uses [LevelDB](https://github.com/google/leveldb). For the bulk sample data, it has its own custom storage layer, which organizes sample data in chunks of constant size (1024 bytes payload). These chunks are then stored on disk in one file per time series.

This sections deals with the various configuration settings and issues you might run into. To dive deeper into the topic, check out the following talks: \* [The Prometheus Time Series Database](https://www.youtube.com/watch?v=HbnGSNEjhUc). \*[Configuring Prometheus for High Performance](https://www.youtube.com/watch?v=hPC60ldCGm8).

**Memory usage**

Prometheus keeps all the currently used chunks in memory. In addition, it keeps as many most recently used chunks in memory as possible. You have to tell Prometheus how much memory it may use for this caching. The flag storage.local.target-heap-size allows you to set the heap size (in bytes) Prometheus aims not to exceed. Note that the amount of physical memory the Prometheus server will use is the result of complex interactions of the Go runtime and the operating system and very hard to predict precisely. As a rule of thumb, you should have at least 50% headroom in physical memory over the configured heap size. (Or, in other words, set storage.local.target-heap-size to a value of two thirds of the physical memory limit Prometheus should not exceed.)

prometheus\_local\_storage\_target\_heap\_size\_bytes

The default value of storage.local.target-heap-size is 2GiB and thus tailored to 3GiB of physical memory usage. If you have less physical memory available, you have to lower the flag value. If you have more memory available, you should raise the value accordingly. Otherwise, Prometheus will not make use of the memory and thus will perform much worse than it could.

Because Prometheus uses most of its heap for long-lived allocations of memory chunks, the [garbage collection target percentage](https://golang.org/pkg/runtime/debug/#SetGCPercent) is set to 40 by default. You can still override this setting via the GOGC environment variable as usual.

For high-performance set-ups, you might need to adjust more flags. Please read through the sections below for details.

Prior to v1.6, there was no flag storage.local.target-heap-size. Instead, the number of chunks kept in memory had to be configured using the flags storage.local.memory-chunks and storage.local.max-chunks-to-persist. These flags still exist for compatibility reasons. However, storage.local.max-chunks-to-persist has no effect anymore, and if storage.local.memory-chunks is set to a non-zero value x, it is used to override the value for storage.local.target-heap-size to 3072\*x.

**Disk usage**

Prometheus stores its on-disk time series data under the directory specified by the flag storage.local.path. The default path is ./data (relative to the working directory), which is good to try something out quickly but most likely not what you want for actual operations. The flag storage.local.retention allows you to configure the retention time for samples. Adjust it to your needs and your available disk space.

**Chunk encoding**

Prometheus currently offers three different types of chunk encodings. The chunk encoding for newly created chunks is determined by the -storage.local.chunk-encoding-version flag. The valid values are 0, 1, or 2.

Type 0 is the simple delta encoding implemented for Prometheus's first chunked storage layer. Type 1 is the current default encoding, a double-delta encoding with much better compression behavior than type 0. Both encodings feature a fixed byte width per sample over the whole chunk, which allows fast random access. While type 0 is the fastest encoding, the difference in encoding cost compared to encoding 1 is tiny. Due to the better compression behavior of type 1, there is really no reason to select type 0 except compatibility with very old Prometheus versions.

Type 2 is a variable bit-width encoding, i.e. each sample in the chunk can use a different number of bits. Timestamps are double-delta encoded, too, but with a slightly different algorithm. A number of different encoding schemes are available for sample values. The choice is made per chunk based on the nature of the sample values (constant, integer, regularly increasing, random…). Major parts of the type 2 encoding are inspired by a paper published by Facebook engineers: [Gorilla: A Fast, Scalable, In-Memory Time Series Database](http://www.vldb.org/pvldb/vol8/p1816-teller.pdf).

With type 2, access within a chunk has to happen sequentially, and the encoding and decoding cost is a bit higher. Overall, type 2 will cause more CPU usage and increased query latency compared to type 1 but offers a much improved compression ratio. The exact numbers depend heavily on the data set and the kind of queries. Below are results from a typical production server with a fairly expensive set of recording rules.

| **Chunk type** | **bytes per sample** | **cores** | **rule evaluation duration** |
| --- | --- | --- | --- |
| 1 | 3.3 | 1.6 | 2.9s |
| 2 | 1.3 | 2.4 | 4.9s |

You can change the chunk encoding each time you start the server, so experimenting with your own use case is encouraged. Take into account, however, that only newly created chunks will use the newly selected chunk encoding, so it will take a while until you see the effects.

For more details about the trade-off between the chunk encodings, see [this blog post](https://prometheus.io/blog/2016/05/08/when-to-use-varbit-chunks/).

**Settings for high numbers of time series**

Prometheus can handle millions of time series. However, with the above mentioned default setting forstorage.local.target-heap-size, you will be limited to about 200,000 time series simultaneously present in memory. For more series, you need more memory, and you need to configure Prometheus to make use of it as described above.

Each of the aforementioned chunks contains samples of a single time series. A time series is thus represented as a series of chunks, which ultimately end up in a time series file (one file per time series) on disk.

A series that has recently received new samples will have an open incomplete head chunk. Once that chunk is completely filled, or the series hasn't received samples in a while, the head chunk is closed and becomes a chunk waiting to be appended to its corresponding series file, i.e. it is waiting for persistence. After the chunk has been persisted to disk, it becomes evictable, provided it is not currently used by a query. Prometheus will evict evictable chunks from memory to satisfy the configured target heap size. A series with an open head chunk is called an active series. This is different from a memory series, which also includes series without an open head chunk but still other chunks in memory (whether waiting for persistence, used in a query, or evictable). A series without any chunks in memory may be archived, upon which it ceases to have any mandatory memory footprint.

The amount of chunks Prometheus can keep in memory depends on the flag value for storage.local.target-heap-size and on the amount of memory used by everything else. If there are not enough chunks evictable to satisfy the target heap size, Prometheus will throttle ingestion of more samples (by skipping scrapes and rule evaluations) until the heap has shrunk enough. Throttled ingestion is really bad for various reasons. You really do not want to be in that situation.

Open head chunks, chunks still waiting for persistence, and chunks being used in a query are not evictable. Thus, the reasons for the inability to evict enough chunks include the following:

1. Queries that use too many chunks.

2. Chunks are piling up waiting for persistence because the storage layer cannot keep up writing chunks.

3. There are too many active time series, which results in too many open head chunks.

Currently, Prometheus has no defence against case (1). Abusive queries will essentially OOM the server.

To defend against case (2), there is a concept of persistence urgency explained in the next section.

Case (3) depends on the targets you monitor. To mitigate an unplanned explosion of the number of series, you can limit the number of samples per individual scrape (see sample\_limit in the [scrape config](https://prometheus.io/docs/operating/configuration/#scrape_config)). If the number of active time series exceeds the number of memory chunks the Prometheus server can afford, the server will quickly throttle ingestion as described above. The only way out if this is to give Prometheus more RAM or reduce the number of time series to ingest.

In fact, you want many more memory chunks than you have series in memory. Prometheus tries to batch up disk writes as much as possible as it helps for both HDD (write as much as possible after each seek) and SSD (tiny writes create write amplification, which limits the effective throughput and burns much more quickly through the lifetime of the device). The more Prometheus can batch up writes, the more efficient is the process of persisting chunks to disk. which helps case (2).

In conclusion, to keep the Prometheus server healthy, make sure it has plenty of headroom of memory chunks available for the number of memory series. A factor of three is a good starting point. Refer to the [section about helpful metrics](https://prometheus.io/docs/operating/storage/#helpful-metrics) to find out what to look for. A very broad rule of thumb for an upper limit of memory series is the total available physical memory divided by 10,000, e.g. About 6M memory series on a 64GiB server.

If you combine a high number of time series with very fast and/or large scrapes, the number of pre-allocated mutexes for series locking might not be sufficient. If you see scrape hiccups while Prometheus is writing a checkpoint or processing expensive queries, try increasing the value of the storage.local.num-fingerprint-mutexes flag. Sometimes tens of thousands or even more are required.

PromQL queries that involve a high number of time series will make heavy use of the LevelDB-backed indexes. If you need to run queries of that kind, tweaking the index cache sizes might be required. The following flags are relevant:

* -storage.local.index-cache-size.label-name-to-label-values: For regular expression matching.
* -storage.local.index-cache-size.label-pair-to-fingerprints: Increase the size if a large number of time series share the same label pair or name.
* -storage.local.index-cache-size.fingerprint-to-metric and -storage.local.index-cache-size.fingerprint-to-timerange: Increase the size if you have a large number of archived time series, i.e. series that have not received samples in a while but are still not old enough to be purged completely.

You have to experiment with the flag values to find out what helps. If a query touches 100,000+ time series, hundreds of MiB might be reasonable. If you have plenty of memory available, using more of it for LevelDB cannot harm. More memory for LevelDB will effectively reduce the number of memory chunks Prometheus can afford.

**Persistence urgency and “rushed mode”**

Naively, Prometheus would all the time try to persist completed chunk to disk as soon as possible. Such a strategy would lead to many tiny write operations, using up most of the I/O bandwidth and keeping the server quite busy. Spinning disks will appear to be very slow because of the many slow seeks required, and SSDs will suffer from write amplification. Prometheus tries instead to batch up write operations as much as possible, which works better if it is allowed to use more memory.

Prometheus will also sync series files after each write (with storage.local.series-sync-strategy=adaptive, which is the default) and use the disk bandwidth for more frequent checkpoints (based on the count of “dirty series”, see [below](https://prometheus.io/docs/operating/storage/#crash-recovery)), both attempting to minimize data loss in case of a crash.

But what to do if the number of chunks waiting for persistence grows too much? Prometheus calculates a score for urgency to persist chunks. The score is between 0 and 1, where 1 corresponds to the highest urgency. Depending on the score, Prometheus will write to disk more frequently. Should the score ever pass the threshold of 0.8, Prometheus enters “rushed mode” (which you can see in the logs). In rushed mode, the following strategies are applied to speed up persisting chunks:

* Series files are not synced after write operations anymore (making better use of the OS's page cache at the price of an increased risk of losing data in case of a server crash – this behavior can be overridden with the flag storage.local.series-sync-strategy).
* Checkpoints are only created as often as configured via the storage.local.checkpoint-interval flag (freeing more disk bandwidth for persisting chunks at the price of more data loss in case of a crash and an increased time to run the subsequent crash recovery).
* Write operations to persist chunks are not throttled anymore and performed as fast as possible.

Prometheus leaves rushed mode once the score has dropped below 0.7.

Throttling of ingestion happens if the urgency score reaches 1. Thus, the rushed mode is not per se something to be avoided. It is, on the contrary, a measure the Prometheus server takes to avoid the really bad situation of throttled ingestion. Occasionally entering rushed mode is OK, if it helps and ultimately leads to leaving rushed mode again. If rushed mode is entered but the urgency score still goes up, the server has a real problem.

**Settings for very long retention time**

If you have set a very long retention time via the storage.local.retention flag (more than a month), you might want to increase the flag value storage.local.series-file-shrink-ratio.

Whenever Prometheus needs to cut off some chunks from the beginning of a series file, it will simply rewrite the whole file. (Some file systems support “head truncation”, which Prometheus currently does not use for several reasons.) To not rewrite a very large series file to get rid of very few chunks, the rewrite only happens if at least 10% of the chunks in the series file are removed. This value can be changed via the mentionedstorage.local.series-file-shrink-ratio flag. If you have a lot of disk space but want to minimize rewrites (at the cost of wasted disk space), increase the flag value to higher values, e.g. 0.3 for 30% of required chunk removal.

**Crash recovery**

Prometheus saves chunks to disk as soon as possible after they are complete. Incomplete chunks are saved to disk during regular checkpoints. You can configure the checkpoint interval with the flag storage.local.checkpoint-interval. Prometheus creates checkpoints more frequently than that if too many time series are in a “dirty” state, i.e. their current incomplete head chunk is not the one that is contained in the most recent checkpoint. This limit is configurable via the storage.local.checkpoint-dirty-series-limit flag.

More active time series to cycle through lead in general to more chunks waiting for persistence, which in turns leads to larger checkpoints and ultimately more time needed for checkpointing. There is a clear trade-off between limiting the loss of data in case of a crash and the ability to scale to high number of active time series. To not spend the majority of the disk throughput for checkpointing, you have to increase the checkpoint interval. Prometheus itself limits the time spent in checkpointing to 50% by waiting after each checkpoint's completion for at least as long as the previous checkpoint took.

Nevertheless, should your server crash, you might still lose data, and your storage might be left in an inconsistent state. Therefore, Prometheus performs a crash recovery after an unclean shutdown, similar to an fsck run for a file system. Details about the crash recovery are logged, so you can use it for forensics if required. Data that cannot be recovered is moved to a directory called orphaned (located under storage.local.path). Remember to delete that data if you do not need it anymore.

The crash recovery usually takes less than a minute. Should it take much longer, consult the log to find out what is going on. With increasing number of time series in the storage (archived or not), the re-indexing tends to dominate the recovery time and can take tens of minutes in extreme cases.

**Data corruption**

If you suspect problems caused by corruption in the database, you can enforce a crash recovery by starting the server with the flag storage.local.dirty.

If that does not help, or if you simply want to erase the existing database, you can easily start fresh by deleting the contents of the storage directory:

1. Stop Prometheus.
2. rm -r <storage path>/\*
3. Start Prometheus.

**Helpful metrics**

Out of the metrics that Prometheus exposes about itself, the following are particularly useful to tweak flags and find out about the required resources. They also help to create alerts to find out in time if a Prometheus server has problems or is out of capacity.

* prometheus\_local\_storage\_memory\_series: The current number of series held in memory.
* prometheus\_local\_storage\_open\_head\_chunks: The number of open head chunks.
* prometheus\_local\_storage\_chunks\_to\_persist: The number of memory chunks that still need to be persisted to disk.
* prometheus\_local\_storage\_memory\_chunks: The current number of chunks held in memory. If you substract the previous two, you get the number of persisted chunks (which are evictable if not currently in use by a query).
* prometheus\_local\_storage\_series\_chunks\_persisted: A histogram of the number of chunks persisted per batch.
* prometheus\_local\_storage\_persistence\_urgency\_score: The urgency score as discussed [above](https://prometheus.io/docs/operating/storage/#persistence-pressure-and-rushed-mode).
* prometheus\_local\_storage\_rushed\_mode is 1 if Prometheus is in “rushed mode”, 0 otherwise. Can be used to calculate the percentage of time Prometheus is in rushed mode.
* prometheus\_local\_storage\_checkpoint\_last\_duration\_seconds: How long the last checkpoint took.
* prometheus\_local\_storage\_checkpoint\_last\_size\_bytes: Size of the last checkpoint in bytes.
* prometheus\_local\_storage\_checkpointing is 1 while Prometheus is checkpointing, 0 otherwise. Can be used to calculate the percentage of time Prometheus is checkpointing.
* prometheus\_local\_storage\_inconsistencies\_total: Counter for storage inconsistencies found. If this is greater than 0, restart the server for recovery.
* prometheus\_local\_storage\_persist\_errors\_total: Counter for persist errors.
* prometheus\_local\_storage\_memory\_dirty\_series: Current number of dirty series.
* process\_resident\_memory\_bytes: Broadly speaking the physical memory occupied by the Prometheus process.
* go\_memstats\_alloc\_bytes: Go heap size (allocated objects in use plus allocated objects not in use anymore but not yet garbage-collected).

### When (not) to use varbit chunks | Prometheus

https://prometheus.io/blog/2016/05/08/when-to-use-varbit-chunks/

**When (not) to use varbit chunks**

Posted at: May 8, 2016 by Björn “Beorn” Rabenstein

The embedded time serie database (TSDB) of the Prometheus server organizes the raw sample data of each time series in chunks of constant 1024 bytes size. In addition to the raw sample data, a chunk contains some meta-data, which allows the selection of a different encoding for each chunk. The most fundamental distinction is the encoding version. You select the version for newly created chunks via the command line flag -storage.local.chunk-encoding-version. Up to now, there were only two supported versions: 0 for the original delta encoding, and 1 for the improved double-delta encoding. With release [0.18.0](https://github.com/prometheus/prometheus/releases/tag/0.18.0), we added version 2, which is another variety of double-delta encoding. We call it varbit encoding because it involves a variable bit-width per sample within the chunk. While version 1 is superior to version 0 in almost every aspect, there is a real trade-off between version 1 and 2. This blog post will help you to make that decision. Version 1 remains the default encoding, so if you want to try out version 2 after reading this article, you have to select it explicitly via the command line flag. There is no harm in switching back and forth, but note that existing chunks will not change their encoding version once they have been created. However, these chunks will gradually be phased out according to the configured retention time and will thus be replaced by chunks with the encoding specified in the command-line flag.

**What is varbit encoding?**

From the beginning, we designed the chunked sample storage for easy addition of new encodings. When Facebook published a [paper on their in-memory TSDB Gorilla](http://www.vldb.org/pvldb/vol8/p1816-teller.pdf), we were intrigued by a number of similarities between the independently developed approaches of Gorilla and Prometheus. However, there were also many fundamental differences, which we studied in detail, wondering if we could get some inspiration from Gorilla to improve Prometheus.

On the rare occasion of a free weekend ahead of me, I decided to give it a try. In a coding spree, I implemented what would later (after a considerable amount of testing and debugging) become the varbit encoding.

In a future blog post, I will describe the technical details of the encoding. For now, you only need to know a few characteristics for your decision between the new varbit encoding and the traditional double-delta encoding. (I will call the latter just “double-delta encoding” from now on but note that the varbit encoding also uses double deltas, just in a different way.)

**What are the advantages of varbit encoding?**

In short: It offers a way better compression ratio. While the double-delta encoding needs about 3.3 bytes per sample for real-life data sets, the varbit encoding went as far down as 1.28 bytes per sample on a typical large production server at SoundCloud. That's almost three times more space efficient (and even slightly better than the 1.37 bytes per sample reported for Gorilla – but take that with a grain of salt as the typical data set at SoundCloud might look different from the typical data set at Facebook).

Now think of the implications: Three times more samples in RAM, three times more samples on disk, only a third of disk ops, and since disk ops are currently the bottleneck for ingestion speed, it will also allow ingestion to be three times faster. In fact, the recently reported new ingestion record of 800,000 samples per second was only possible with varbit chunks – and with an SSD, obviously. With spinning disks, the bottleneck is reached far earlier, and thus the 3x gain matters even more.

All of this sounds too good to be true…

**So where is the catch?**

For one, the varbit encoding is more complex. The computational cost to encode and decode values is therefore somewhat increased, which fundamentally affects everything that writes or reads sample data. Luckily, it is only a proportional increase of something that usually contributes only a small part to the total cost of an operation.

Another property of the varbit encoding is potentially way more relevant: samples in varbit chunks can only be accessed sequentially, while samples in double-delta encoded chunks are randomly accessible by index. Since writes in Prometheus are append-only, the different access patterns only affect reading of sample data. The practical impact depends heavily on the nature of the originating PromQL query.

A pretty harmless case is the retrieval of all samples within a time interval. This happens when evaluating a range selector or rendering a dashboard with a resolution similar to the scrape frequency. The Prometheus storage engine needs to find the starting point of the interval. With double-delta chunks, it can perform a binary search, while it has to scan sequentially through a varbit chunk. However, once the starting point is found, all remaining samples in the interval need to be decoded sequentially anyway, which is only slightly more expensive with the varbit encoding.

The trade-off is different for retrieving a small number of non-adjacent samples from a chunk, or for plainly retrieving a single sample in a so-called instant query. Potentially, the storage engine has to iterate through a lot of samples to find the few samples to be returned. Fortunately, the most common source of instant queries are rule evaluations referring to the latest sample in each involved time series. Not completely by coincidence, I recently improved the retrieval of the latest sample of a time series. Essentially, the last sample added to a time series is cached now. A query that needs only the most recent sample of a time series doesn't even hit the chunk layer anymore, and the chunk encoding is irrelevant in that case.

Even if an instant query refers to a sample in the past and therefore has to hit the chunk layer, most likely other parts of the query, like the index lookup, will dominate the total query time. But there are real-life queries where the sequential access pattern required by varbit chunks will start to matter a lot.

**What is the worst-case query for varbit chunks?**

The worst case for varbit chunks is if you need just one sample from somewhere in the middle of each chunk of a very long time series. Unfortunately, there is a real use-case for that. Let's assume a time series compresses nicely enough to make each chunk last for about eight hours. That's about three chunks a day, or about 100 chunks a month. If you have a dashboard that displays the time series in question for the last month with a resolution of 100 data points, the dashboard will execute a query that retrieves a single sample from 100 different chunks. Even then, the differences between chunk encodings will be dominated by other parts of the query execution time. Depending on circumstances, my guess would be that the query might take 50ms with double-delta encoding and 100ms with varbit encoding.

However, if your dashboard query doesn't only touch a single time series but aggregates over thousands of time series, the number of chunks to access multiplies accordingly, and the overhead of the sequential scan will become dominant. (Such queries are frowned upon, and we usually recommend to use a [recording rule](https://prometheus.io/docs/querying/rules/#recording-rules) for queries of that kind that are used frequently, e.g. in a dashboard.) But with the double-delta encoding, the query time might still have been acceptable, let's say around one second. After the switch to varbit encoding, the same query might last tens of seconds, which is clearly not what you want for a dashboard.

**What are the rules of thumb?**

To put it as simply as possible: If you are neither limited on disk capacity nor on disk ops, don't worry and stick with the default of the classical double-delta encoding.

However, if you would like a longer retention time or if you are currently bottle-necked on disk ops, I invite you to play with the new varbit encoding. Start your Prometheus server with -storage.local.chunk-encoding-version=2and wait for a while until you have enough new chunks with varbit encoding to vet the effects. If you see queries that are becoming unacceptably slow, check if you can use [recording rules](https://prometheus.io/docs/querying/rules/#recording-rules) to speed them up. Most likely, those queries will gain a lot from that even with the old double-delta encoding.

If you are interested in how the varbit encoding works behind the scenes, stay tuned for another blog post in the not too distant future.

## Federation | Prometheus

https://prometheus.io/docs/operating/federation/

FEDERATION

* [Use cases](https://prometheus.io/docs/operating/federation/#use-cases)
  + [Hierarchical federation](https://prometheus.io/docs/operating/federation/#hierarchical-federation)
  + [Cross-service federation](https://prometheus.io/docs/operating/federation/#cross-service-federation)
* [Configuring federation](https://prometheus.io/docs/operating/federation/#configuring-federation)

Federation allows a Prometheus server to scrape selected time series from another Prometheus server.

**Use cases**

There are different use cases for federation. Commonly, it is used to either achieve scalable Prometheus monitoring setups or to pull related metrics from one service's Prometheus into another.

Hierarchical federation

Hierarchical federation allows Prometheus to scale to environments with tens of data centers and millions of nodes. In this use case, the federation topology resembles a tree, with higher-level Prometheus servers collecting aggregated time series data from a larger number of subordinated servers.

For example, a setup might consist of many per-datacenter Prometheus servers that collect data in high detail (instance-level drill-down), and a set of global Prometheus servers which collect and store only aggregated data (job-level drill-down) from those local servers. This provides an aggregate global view and detailed local views.

Cross-service federation

In cross-service federation, a Prometheus server of one service is configured to scrape selected data from another service's Prometheus server to enable alerting and queries against both datasets within a single server.

For example, a cluster scheduler running multiple services might expose resource usage information (like memory and CPU usage) about service instances running on the cluster. On the other hand, a service running on that cluster will only expose application-specific service metrics. Often, these two sets of metrics are scraped by separate Prometheus servers. Using federation, the Prometheus server containing service-level metrics may pull in the cluster resource usage metrics about its specific service from the cluster Prometheus, so that both sets of metrics can be used within that server.

**Configuring federation**

On any given Prometheus server, the /federate endpoint allows retrieving the current value for a selected set of time series in that server. At least one match[] URL parameter must be specified to select the series to expose. Each match[] argument needs to specify an [instant vector selector](https://prometheus.io/docs/querying/basics/#instant-vector-selectors) like up or {job="api-server"}. If multiple match[] parameters are provided, the union of all matched series is selected.

To federate metrics from one server to another, configure your destination Prometheus server to scrape from the /federate endpoint of a source server, while also enabling the honor\_labels scrape option (to not overwrite any labels exposed by the source server) and passing in the desired match[] parameters. For example, the following scrape\_config federates any series with the label job="prometheus" or a metric name starting with job: from the Prometheus servers at source-prometheus-{1,2,3}:9090 into the scraping Prometheus:

- job\_name: 'federate'

scrape\_interval: 15s

honor\_labels: true

metrics\_path: '/federate'

params:

'match[]':

- '{job="prometheus"}'

- '{\_\_name\_\_=~"job:.\*"}'

static\_configs:

- targets:

- 'source-prometheus-1:9090'

- 'source-prometheus-2:9090'

- 'source-prometheus-3:9090'

# Alerting

## Alerting overview | Prometheus

https://prometheus.io/docs/alerting/overview/

ALERTING OVERVIEW

Alerting with Prometheus is separated into two parts. Alerting rules in Prometheus servers send alerts to an Alertmanager. The [Alertmanager](https://prometheus.io/docs/alerting/alertmanager) then manages those alerts, including silencing, inhibition, aggregation and sending out notifications via methods such as email, PagerDuty and HipChat.

The main steps to setting up alerting and notifications are:

* Setup and [configure](https://prometheus.io/docs/alerting/configuration) the Alertmanager
* Configure Prometheus to talk to the Alertmanager with the -alertmanager.url flag
* Create [alerting rules](https://prometheus.io/docs/alerting/rules) in Prometheus

## Alertmanager | Prometheus

https://prometheus.io/docs/alerting/alertmanager/

**ALERTMANAGER**

* [Grouping](https://prometheus.io/docs/alerting/alertmanager/#grouping)
* [Inhibition](https://prometheus.io/docs/alerting/alertmanager/#inhibition)
* [Silences](https://prometheus.io/docs/alerting/alertmanager/#silences)
* [Client behavior](https://prometheus.io/docs/alerting/alertmanager/#client-behavior)

The [Alertmanager](https://github.com/prometheus/alertmanager) handles alerts sent by client applications such as the Prometheus server. It takes care of deduplicating, grouping, and routing them to the correct receiver integration such as email, PagerDuty, or OpsGenie. It also takes care of silencing and inhibition of alerts.

The following describes the core concepts the Alertmanager implements. Consult the [configuration documentation](https://prometheus.io/docs/alerting/configuration) to learn how to use them in more detail.

**Grouping**

Grouping categorizes alerts of similar nature into a single notification. This is especially useful during larger outages when many systems fail at once and hundreds to thousands of alerts may be firing simultaneously.

**Example:** Dozens or hundreds of instances of a service are running in your cluster when a network partition occurs. Half of your service instances can no longer reach the database. Alerting rules in Prometheus were configured to send an alert for each service instance if it cannot communicate with the database. As a result hundreds of alerts are sent to Alertmanager.

As a user, one only wants to get a single page while still being able to see exactly which service instances were affected. Thus one can configure Alertmanager to group alerts by their cluster and alertname so it sends a single compact notification.

Grouping of alerts, timing for the grouped notifications, and the receivers of those notifications are configured by a routing tree in the configuration file.

**Inhibition**

Inhibition is a concept of suppressing notifications for certain alerts if certain other alerts are already firing.

**Example:** An alert is firing that informs that an entire cluster is not reachable. Alertmanager can be configured to mute all other alerts concerning this cluster if that particular alert is firing. This prevents notifications for hundreds or thousands of firing alerts that are unrelated to the actual issue.

Inhibitions are configured through the Alertmanager's configuration file.

**Silences**

Silences are a straightforward way to simply mute alerts for a given time. A silence is configured based on matchers, just like the routing tree. Incoming alerts are checked whether they match all the equality or regular expression matchers of an active silence. If they do, no notifications will be sent out for that alert.

Silences are configured in the web interface of the Alertmanager.

**Client behavior**

The Alertmanager has [special requirements](https://prometheus.io/docs/alerting/clients) for behavior of its client. Those are only relevant for advanced use cases where Prometheus is not used to send alerts.

## Configuration | Prometheus

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

**CONFIGURATION**

* [Configuration file](https://prometheus.io/docs/alerting/configuration/#configuration-file)
* [<route>](https://prometheus.io/docs/alerting/configuration/#<route>)
  + [Example](https://prometheus.io/docs/alerting/configuration/#example)
* [<inhibit\_rule>](https://prometheus.io/docs/alerting/configuration/#<inhibit_rule>)
* [<receiver>](https://prometheus.io/docs/alerting/configuration/#<receiver>)
* [<email\_config>](https://prometheus.io/docs/alerting/configuration/#<email_config>)
* [<hipchat\_config>](https://prometheus.io/docs/alerting/configuration/#<hipchat_config>)
* [<pagerduty\_config>](https://prometheus.io/docs/alerting/configuration/#<pagerduty_config>)
* [<pushover\_config>](https://prometheus.io/docs/alerting/configuration/#<pushover_config>)
* [<slack\_config>](https://prometheus.io/docs/alerting/configuration/#<slack_config>)
* [<opsgenie\_config>](https://prometheus.io/docs/alerting/configuration/#<opsgenie_config>)
* [<victor\_ops\_config>](https://prometheus.io/docs/alerting/configuration/#<victor_ops_config>)
* [<webhook\_config>](https://prometheus.io/docs/alerting/configuration/#<webhook_config>)

[Alertmanager](https://github.com/prometheus/alertmanager) is configured via command-line flags and a configuration file. While the command-line flags configure immutable system parameters, the configuration file defines inhibition rules, notification routing and notification receivers.

The [visual editor](https://prometheus.io/webtools/alerting/routing-tree-editor) can assist in building routing trees.

To view all available command-line flags, run alertmanager -h.

Alertmanager can reload its configuration at runtime. If the new configuration is not well-formed, the changes will not be applied and an error is logged. A configuration reload is triggered by sending a SIGHUP to the process or sending a HTTP POST request to the /-/reload endpoint.

**Configuration file**

To specify which configuration file to load, use the -config.file flag.

./alertmanager -config.file=simple.yml

The file is written in the [YAML format](http://en.wikipedia.org/wiki/YAML), defined by the scheme described below. Brackets indicate that a parameter is optional. For non-list parameters the value is set to the specified default.

Generic placeholders are defined as follows:

* <duration>: a duration matching the regular expression [0-9]+(ms|[smhdwy])
* <labelname>: a string matching the regular expression [a-zA-Z\_][a-zA-Z0-9\_]\*
* <labelvalue>: a string of unicode characters
* <filepath>: a valid path in the current working directory
* <boolean>: a boolean that can take the values true or false
* <string>: a regular string
* <tmpl\_string>: a string which is template-expanded before usage

The other placeholders are specified separately.

A valid example file can be found [here](https://github.com/prometheus/alertmanager/blob/master/doc/examples/simple.yml).

The global configuration specifies parameters that are valid in all other configuration contexts. They also serve as defaults for other configuration sections.

global:

# ResolveTimeout is the time after which an alert is declared resolved

# if it has not been updated.

[ resolve\_timeout: [<duration>](https://prometheus.io/docs/alerting/configuration/#<duration>) | default = 5m ]

# The default SMTP From header field.

[ smtp\_from: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) ]

# The default SMTP smarthost used for sending emails.

[ smtp\_smarthost: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

# SMTP authentication information.

[ smtp\_auth\_username: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

[ smtp\_auth\_password: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

[ smtp\_auth\_secret: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

# The default SMTP TLS requirement.

[ smtp\_require\_tls: <bool> | default = true ]

# The API URL to use for Slack notifications.

[ slack\_api\_url: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

[ pagerduty\_url: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = "https://events.pagerduty.com/generic/2010-04-15/create\_event.json" ]

[ opsgenie\_api\_host: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = "https://api.opsgenie.com/" ]

[ hipchat\_url: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = "https://api.hipchat.com/" ]

[ hipchat\_auth\_token: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

# Files from which custom notification template definitions are read.

# The last component may use a wildcard matcher, e.g. 'templates/\*.tmpl'.

templates:

[ - [<filepath>](https://prometheus.io/docs/alerting/configuration/#<filepath>) ... ]

# The root node of the routing tree.

route: [<route>](https://prometheus.io/docs/alerting/configuration/#<route>)

# A list of notification receivers.

receivers:

- [<receiver>](https://prometheus.io/docs/alerting/configuration/#<receiver>) ...

# A list of inhibition rules.

inhibit\_rules:

[ - [<inhibit\_rule>](https://prometheus.io/docs/alerting/configuration/#<inhibit_rule>) ... ]

**<route>**

A route block defines a node in a routing tree and its children. Its optional configuration parameters are inherited from its parent node if not set.

Every alert enters the routing tree at the configured top-level route, which must match all alerts (i.e. not have any configured matchers). It then traverses the child nodes. If continue is set to false, it stops after the first matching child. If continue is true on a matching node, the alert will continue matching against subsequent siblings. If an alert does not match any children of a node (no matching child nodes, or none exist), the alert is handled based on the configuration parameters of the current node.

[ receiver: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

[ group\_by: '[' [<labelname>](https://prometheus.io/docs/alerting/configuration/#<labelname>), ... ']' ]

# Whether an alert should continue matching subsequent sibling nodes.

[ continue: [<boolean>](https://prometheus.io/docs/alerting/configuration/#<boolean>) | default = false ]

# A set of equality matchers an alert has to fulfill to match the node.

match:

[ [<labelname>](https://prometheus.io/docs/alerting/configuration/#<labelname>): [<labelvalue>](https://prometheus.io/docs/alerting/configuration/#<labelvalue>), ... ]

# A set of regex-matchers an alert has to fulfill to match the node.

match\_re:

[ [<labelname>](https://prometheus.io/docs/alerting/configuration/#<labelname>): <regex>, ... ]

# How long to initially wait to send a notification for a group

# of alerts. Allows to wait for an inhibiting alert to arrive or collect

# more initial alerts for the same group. (Usually ~0s to few minutes.)

[ group\_wait: [<duration>](https://prometheus.io/docs/alerting/configuration/#<duration>) ]

# How long to wait before sending a notification about new alerts that

# are added to a group of alerts for which an initial notification has

# already been sent. (Usually ~5min or more.)

[ group\_interval: [<duration>](https://prometheus.io/docs/alerting/configuration/#<duration>) ]

# How long to wait before sending a notification again if it has already

# been sent successfully for an alert. (Usually ~3h or more).

[ repeat\_interval: [<duration>](https://prometheus.io/docs/alerting/configuration/#<duration>) ]

# Zero or more child routes.

routes:

[ - [<route>](https://prometheus.io/docs/alerting/configuration/#<route>) ... ]

Example

# The root route with all parameters, which are inherited by the child

# routes if they are not overwritten.

route:

receiver: 'default-receiver'

group\_wait: 30s

group\_interval: 5m

repeat\_interval: 4h

group\_by: [cluster, alertname]

# All alerts that do not match the following child routes

# will remain at the root node and be dispatched to 'default-receiver'.

routes:

# All alerts with service=mysql or service=cassandra

# are dispatched to the database pager.

- receiver: 'database-pager'

group\_wait: 10s

match\_re:

service: mysql|cassandra

# All alerts with the team=frontend label match this sub-route.

# They are grouped by product and environment rather than cluster

# and alertname.

- receiver: 'frontend-pager'

group\_by: [product, environment]

match:

team: frontend

**<inhibit\_rule>**

An inhibition rule is a rule that mutes an alert matching a set of matchers under the condition that an alert exists that matches another set of matchers. Both alerts must have a set of equal labels.

# Matchers that have to be fulfilled in the alerts to be muted.

target\_match:

[ [<labelname>](https://prometheus.io/docs/alerting/configuration/#<labelname>): [<labelvalue>](https://prometheus.io/docs/alerting/configuration/#<labelvalue>), ... ]

target\_match\_re:

[ [<labelname>](https://prometheus.io/docs/alerting/configuration/#<labelname>): <regex>, ... ]

# Matchers for which one or more alerts have to exist for the

# inhibition to take effect.

source\_match:

[ [<labelname>](https://prometheus.io/docs/alerting/configuration/#<labelname>): [<labelvalue>](https://prometheus.io/docs/alerting/configuration/#<labelvalue>), ... ]

source\_match\_re:

[ [<labelname>](https://prometheus.io/docs/alerting/configuration/#<labelname>): <regex>, ... ]

# Labels that must have an equal value in the source and target

# alert for the inhibition to take effect.

[ equal: '[' [<labelname>](https://prometheus.io/docs/alerting/configuration/#<labelname>), ... ']' ]

**<receiver>**

Receiver is a named configuration of one or more notification integrations.

**Other receiver implementations available in version 0.0.4 of Alertmanager are not implemented yet. We are gladly accepting any contributions to add them to the new implementation.**

# The unique name of the receiver.

name: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>)

# Configurations for several notification integrations.

email\_configs:

[ - [<email\_config>](https://prometheus.io/docs/alerting/configuration/#<email_config>), ... ]

hipchat\_configs:

[ - [<hipchat\_config>](https://prometheus.io/docs/alerting/configuration/#<hipchat_config>), ... ]

pagerduty\_configs:

[ - [<pagerduty\_config>](https://prometheus.io/docs/alerting/configuration/#<pagerduty_config>), ... ]

pushover\_configs:

[ - [<pushover\_config>](https://prometheus.io/docs/alerting/configuration/#<pushover_config>), ... ]

slack\_configs:

[ - [<slack\_config>](https://prometheus.io/docs/alerting/configuration/#<slack_config>), ... ]

opsgenie\_configs:

[ - [<opsgenie\_config>](https://prometheus.io/docs/alerting/configuration/#<opsgenie_config>), ... ]

webhook\_configs:

[ - [<webhook\_config>](https://prometheus.io/docs/alerting/configuration/#<webhook_config>), ... ]

**<email\_config>**

# Whether or not to notify about resolved alerts.

[ send\_resolved: [<boolean>](https://prometheus.io/docs/alerting/configuration/#<boolean>) | default = false ]

# The email address to send notifications to.

to: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>)

# The sender address.

[ from: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = global.smtp\_from ]

# The SMTP host through which emails are sent.

[ smarthost: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = global.smtp\_smarthost ]

# SMTP authentication information.

[ auth\_username: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

[ auth\_password: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

[ auth\_secret: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

[ auth\_identity: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

[ require\_tls: <bool> | default = global.smtp\_require\_tls ]

# The HTML body of the email notification.

[ html: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "email.default.html" . }}' ]

# Further headers email header key/value pairs. Overrides any headers

# previously set by the notification implementation.

[ headers: { [<string>](https://prometheus.io/docs/alerting/configuration/#<string>): [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>), ... } ]

**<hipchat\_config>**

HipChat notifications use a [Build Your Own](https://confluence.atlassian.com/hc/integrations-with-hipchat-server-683508267.html) integration.

# Whether or not to notify about resolved alerts.

[ send\_resolved: [<boolean>](https://prometheus.io/docs/alerting/configuration/#<boolean>) | default = false ]

# The HipChat Room ID.

room\_id: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>)

# The auth token.

[ auth\_token: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = global.hipchat\_auth\_token ]

# The URL to send API requests to.

[ url: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = global.hipchat\_url ]

# See https://www.hipchat.com/docs/apiv2/method/send\_room\_notification

# A label to be shown in addition to the sender's name.

[ from: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "hipchat.default.from" . }}' ]

# The message body.

[ message: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "hipchat.default.message" . }}' ]

# Whether this message should trigger a user notification.

[ notify: [<boolean>](https://prometheus.io/docs/alerting/configuration/#<boolean>) | default = false ]

# Determines how the message is treated by the alertmanager and rendered inside HipChat. Valid values are 'text' and 'html'.

[ message\_format: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = 'text' ]

# Background color for message.

[ color: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ if eq .Status "firing" }}red{{ else }}green{{ end }}' ]

**<pagerduty\_config>**

PagerDuty notifications are sent via the [PagerDuty API](https://developer.pagerduty.com/documentation/integration/events).

# Whether or not to notify about resolved alerts.

[ send\_resolved: [<boolean>](https://prometheus.io/docs/alerting/configuration/#<boolean>) | default = true ]

# The PagerDuty service key.

service\_key: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>)

# The URL to send API requests to

[ url: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = global.pagerduty\_url ]

# The client identification of the Alertmanager.

[ client: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "pagerduty.default.client" . }}' ]

# A backlink to the sender of the notification.

[ client\_url: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "pagerduty.default.clientURL" . }}' ]

# A description of the incident.

[ description: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "pagerduty.default.description" .}}' ]

# A set of arbitrary key/value pairs that provide further detail

# about the incident.

[ details: { [<string>](https://prometheus.io/docs/alerting/configuration/#<string>): [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>), ... } | default = {

firing: '{{ template "pagerduty.default.instances" .Alerts.Firing }}'

resolved: '{{ template "pagerduty.default.instances" .Alerts.Resolved }}'

num\_firing: '{{ .Alerts.Firing | len }}'

num\_resolved: '{{ .Alerts.Resolved | len }}'

} ]

**<pushover\_config>**

Pushover notifications are sent via the [Pushover API](https://pushover.net/api).

# The recipient user’s user key.

user\_key: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>)

# Your registered application’s API token, see https://pushover.net/apps

token: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>)

# Notification title.

[ title: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "pushover.default.title" . }}' ]

# Notification message.

[ message: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "pushover.default.message" . }}' ]

# A supplementary URL shown alongside the message.

[ url: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "pushover.default.url" . }}' ]

# Priority, see https://pushover.net/api#priority

[ priority: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ if eq .Status "firing" }}2{{ else }}0{{ end }}' ]

# How often the Pushover servers will send the same notification to the user.

# Must be at least 30 seconds.

[ retry: [<duration>](https://prometheus.io/docs/alerting/configuration/#<duration>) | default = 1m ]

# How long your notification will continue to be retried for, unless the user

# acknowledges the notification.

[ expire: [<duration>](https://prometheus.io/docs/alerting/configuration/#<duration>) | default = 1h ]

**<slack\_config>**

Slack notifications are sent via [Slack webhooks](https://api.slack.com/incoming-webhooks).

# Whether or not to notify about resolved alerts.

[ send\_resolved: [<boolean>](https://prometheus.io/docs/alerting/configuration/#<boolean>) | default = false ]

# The Slack webhook URL.

[ api\_url: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = global.slack\_api\_url ]

# The channel or user to send notifications to.

channel: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>)

# API request data as defined by the Slack webhook API.

[ color: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ if eq .Status "firing" }}danger{{ else }}good{{ end }}' ]

[ username: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "slack.default.username" . }}' ]

[ title: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "slack.default.title" . }}' ]

[ title\_link: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "slack.default.titlelink" . }}' ]

[ icon\_emoji: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) ]

[ icon\_url: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) ]

[ pretext: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "slack.default.pretext" . }}' ]

[ text: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "slack.default.text" . }}' ]

[ fallback: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "slack.default.fallback" . }}' ]

**<opsgenie\_config>**

OpsGenie notifications are sent via the [OpsGenie API](https://www.opsgenie.com/docs/web-api/alert-api).

# Whether or not to notify about resolved alerts.

[ send\_resolved: [<boolean>](https://prometheus.io/docs/alerting/configuration/#<boolean>) | default = true ]

# The API key to use when talking to the OpsGenie API.

api\_key: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>)

# The host to send OpsGenie API requests to.

[ api\_host: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = global.opsgenie\_api\_host ]

# A description of the incident.

[ description: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "opsgenie.default.description" . }}' ]

# A backlink to the sender of the notification.

[ source: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) | default = '{{ template "opsgenie.default.source" . }}' ]

# A set of arbitrary key/value pairs that provide further detail

# about the incident.

[ details: { [<string>](https://prometheus.io/docs/alerting/configuration/#<string>): [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>), ... } ]

# Comma separated list of team responsible for notifications.

[ teams: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) ]

# Comma separated list of tags attached to the notifications.

[ tags: [<tmpl\_string>](https://prometheus.io/docs/alerting/configuration/#<tmpl_string>) ]

**<victor\_ops\_config>**

VictorOps notifications are sent out via the [VictorOps API](https://help.victorops.com/knowledge-base/victorops-restendpoint-integration/)

# The API key to use when talking to the VictorOps API.

api\_key: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>)

# The VictorOps API URL.

[ api\_url: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = global.victor\_ops\_api\_url ]

# A key used to map the alert to a team.

[ routing\_key: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

# Describes the behavior of the alert (Critical, Acknowledgement, Info, Recovery).

[ message\_type: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) ]

# Contains explanation of the alerted problem.

[ state\_message: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = '{{ template "victorops.default.state\_message" . }}' ]

# The monitoring tool the state message is from.

[ monitoring\_tool: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>) | default = '{{ template "victorops.default.monitoring\_tool" . }}' ]

**<webhook\_config>**

The webhook receiver allows configuring a generic receiver.

# Whether or not to notify about resolved alerts.

[ send\_resolved: [<boolean>](https://prometheus.io/docs/alerting/configuration/#<boolean>) | default = true ]

# The endpoint to send HTTP POST requests to.

url: [<string>](https://prometheus.io/docs/alerting/configuration/#<string>)

The Alertmanager will send HTTP POST requests in the following JSON format to the configured endpoint:

{

"version": "3",

"groupKey": <number> // key identifying the group of alerts (e.g. to deduplicate)

"status": "<resolved|firing>",

"receiver": [<string>](https://prometheus.io/docs/alerting/configuration/#<string>),

"groupLabels": <object>,

"commonLabels": <object>,

"commonAnnotations": <object>,

"externalURL": [<string>](https://prometheus.io/docs/alerting/configuration/#<string>), // backling to the Alertmanager.

"alerts": [

{

"labels": <object>,

"annotations": <object>,

"startsAt": "<rfc3339>",

"endsAt": "<rfc3339>"

},

...

]

}

## Alerting rules | Prometheus

https://prometheus.io/docs/alerting/rules/

**ALERTING RULES**

* [Defining alerting rules](https://prometheus.io/docs/alerting/rules/#defining-alerting-rules)
* [Inspecting alerts during runtime](https://prometheus.io/docs/alerting/rules/#inspecting-alerts-during-runtime)
* [Sending alert notifications](https://prometheus.io/docs/alerting/rules/#sending-alert-notifications)

Alerting rules allow you to define alert conditions based on Prometheus expression language expressions and to send notifications about firing alerts to an external service. Whenever the alert expression results in one or more vector elements at a given point in time, the alert counts as active for these elements' label sets.

Alerting rules are configured in Prometheus in the same way as [recording rules](https://prometheus.io/docs/querying/rules).

Defining alerting rules

Alerting rules are defined in the following syntax:

ALERT <alert name>

IF <expression>

[ FOR <duration> ]

[ LABELS <label set> ]

[ ANNOTATIONS <label set> ]

The optional FOR clause causes Prometheus to wait for a certain duration between first encountering a new expression output vector element (like an instance with a high HTTP error rate) and counting an alert as firing for this element. Elements that are active, but not firing yet, are in pending state.

The LABELS clause allows specifying a set of additional labels to be attached to the alert. Any existing conflicting labels will be overwritten. The label values can be templated.

The ANNOTATIONS clause specifies another set of labels that are not identifying for an alert instance. They are used to store longer additional information such as alert descriptions or runbook links. The annotation values can be templated.

Templating

Label and annotation values can be templated using [console templates](https://prometheus.io/docs/visualization/consoles). The $labels variable holds the label key/value pairs of an alert instance and $value holds the evaluated value of an alert instance.

# To insert a firing element's label values:

{{ $labels.<labelname> }}

# To insert the numeric expression value of the firing element:

{{ $value }}

Examples:

# Alert for any instance that is unreachable for >5 minutes.

ALERT InstanceDown

IF up == 0

FOR 5m

LABELS { severity = "page" }

ANNOTATIONS {

summary = "Instance {{ $labels.instance }} down",

description = "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 5 minutes.",

}

# Alert for any instance that have a median request latency >1s.

ALERT APIHighRequestLatency

IF api\_http\_request\_latencies\_second{quantile="0.5"} > 1

FOR 1m

ANNOTATIONS {

summary = "High request latency on {{ $labels.instance }}",

description = "{{ $labels.instance }} has a median request latency above 1s (current value: {{ $value }}s)",

}

Inspecting alerts during runtime

To manually inspect which alerts are active (pending or firing), navigate to the "Alerts" tab of your Prometheus instance. This will show you the exact label sets for which each defined alert is currently active.

For pending and firing alerts, Prometheus also stores synthetic time series of the form ALERTS{alertname="<alert name>", alertstate="pending|firing", <additional alert labels>}. The sample value is set to 1 as long as the alert is in the indicated active (pending or firing) state, and a single 0 value gets written out when an alert transitions from active to inactive state. Once inactive, the time series does not get further updates.

Sending alert notifications

Prometheus's alerting rules are good at figuring what is broken right now, but they are not a fully-fledged notification solution. Another layer is needed to add summarization, notification rate limiting, silencing and alert dependencies on top of the simple alert definitions. In Prometheus's ecosystem, the [Alertmanager](https://prometheus.io/docs/alerting/alertmanager) takes on this role. Thus, Prometheus may be configured to periodically send information about alert states to an Alertmanager instance, which then takes care of dispatching the right notifications. The Alertmanager instance may be configured via the -alertmanager.url command line flag.

## Clients | Prometheus

https://prometheus.io/docs/alerting/clients/

**Disclaimer: Prometheus automatically takes care of sending alerts generated by its configured** [**alerting rules**](https://prometheus.io/docs/alerting/rules)**. It is highly recommended to configure alerting rules in Prometheus based on time series data rather than implementing a direct client.**

The Alertmanager listens for alerts on an API endpoint at /api/v1/alerts. Clients are expected to continuously re-send alerts as long as they are still active (usually on the order of 30 seconds to 3 minutes). Clients can push a list of alerts to that endpoint via a POST request of the following format:

[

{

"labels": {

"<labelname>": "<labelvalue>",

...

},

"annotations": {

"<labelname>": "<labelvalue>",

},

"startsAt": "<rfc3339>",

"endsAt": "<rfc3339>"

"generatorURL": "<generator\_url>"

},

...

]

The labels are used to identify identical instances of an alert and to perform deduplication. The annotations are always set to those received most recently and are not identifying an alert.

Both timestamps are optional. If startsAt is omitted, the current time is assigned by the Alertmanager. endsAt is only set if the end time of an alert is known. Otherwise it will be set to a configurable timeout period from the time since the alert was last received.

The generatorURL field is a unique back-link which identifies the causing entity of this alert in the client.

Alertmanager also supports a legacy endpoint on /api/alerts which is compatible with Prometheus versions 0.16.2 and lower.

## prometheus/alertmanager · Quay

https://quay.io/repository/prometheus/alertmanager

docker pull quay.io/prometheus/alertmanager

**Description**

**Alertmanager**

The Alertmanager handles alerts sent by client applications such as the Prometheus server. It takes care of deduplicating, grouping, and routing them to the correct receiver integration such as email, PagerDuty, or OpsGenie. It also takes care of silencing and inhibition of alerts.

[Documentation](http://prometheus.io/docs/alerting/alertmanager/)

**Installation**

**Build dependencies**

These dependencies are necessary for building Alertmanager. There are no runtime dependencies, as the resulting binary is statically linked.

Debian family:

sudo apt-get install build-essential libc6-dev

Red Hat family:

sudo yum install glibc-static

**Compiling the binary**

The current version has to be run from the repository folder as UI assets and notification templates are not yet statically compiled into the binary.

You can either go get it:

$ GO15VENDOREXPERIMENT=1 go get github.com/prometheus/alertmanager

# cd $GOPATH/src/github.com/prometheus/alertmanager

$ alertmanager -config.file=<your\_file>

Or checkout the source code and build manually:

$ mkdir -p $GOPATH/src/github.com/prometheus

$ cd $GOPATH/src/github.com/prometheus

$ git clone https://github.com/prometheus/alertmanager.git

$ cd alertmanager

$ make build

$ ./alertmanager -config.file=<your\_file>

**Example**

This is an example configuration that should cover most relevant aspects of the new YAML configuration format. Authoritative source for now is the [code](https://github.com/prometheus/alertmanager/tree/dev/config).

global:

# The smarthost and SMTP sender used for mail notifications.

smtp\_smarthost: 'localhost:25'

smtp\_from: 'alertmanager@example.org'

# The root route on which each incoming alert enters.

route:

# The root route must not have any matchers as it is the entry point for

# all alerts. It needs to have a receiver configured so alerts that do not

# match any of the sub-routes are sent to someone.

receiver: 'team-X-mails'

# The labels by which incoming alerts are grouped together. For example,

# multiple alerts coming in for cluster=A and alertname=LatencyHigh would

# be batched into a single group.

group\_by: ['alertname', 'cluster']

# When a new group of alerts is created by an incoming alert, wait at

# least 'group\_wait' to send the initial notification.

# This way ensures that you get multiple alerts for the same group that start

# firing shortly after another are batched together on the first

# notification.

group\_wait: 30s

# When the first notification was sent, wait 'group\_interval' to send a batch

# of new alerts that started firing for that group.

group\_interval: 5m

# If an alert has successfully been sent, wait 'repeat\_interval' to

# resend them.

repeat\_interval: 3h

# All the above attributes are inherited by all child routes and can

# overwritten on each.

# The child route trees.

routes:

# This routes performs a regular expression match on alert labels to

# catch alerts that are related to a list of services.

- match\_re:

service: ^(foo1|foo2|baz)$

receiver: team-X-mails

# The service has a sub-route for critical alerts, any alerts

# that do not match, i.e. severity != critical, fall-back to the

# parent node and are sent to 'team-X-mails'

routes:

- match:

severity: critical

receiver: team-X-pager

- match:

service: files

receiver: team-Y-mails

routes:

- match:

severity: critical

receiver: team-Y-pager

# This route handles all alerts coming from a database service. If there's

# no team to handle it, it defaults to the DB team.

- match:

service: database

receiver: team-DB-pager

# Also group alerts by affected database.

group\_by: [alertname, cluster, database]

routes:

- match:

owner: team-X

receiver: team-X-pager

- match:

owner: team-Y

receiver: team-Y-pager

# Inhibition rules allow to mute a set of alerts given that another alert is

# firing.

# We use this to mute any warning-level notifications if the same alert is

# already critical.

inhibit\_rules:

- source\_match:

severity: 'critical'

target\_match:

severity: 'warning'

# Apply inhibition if the alertname is the same.

equal: ['alertname']

receivers:

- name: 'team-X-mails'

email\_configs:

- to: 'team-X+alerts@example.org'

- name: 'team-X-pager'

email\_configs:

- to: 'team-X+alerts-critical@example.org'

pagerduty\_configs:

- service\_key: <team-X-key>

- name: 'team-Y-mails'

email\_configs:

- to: 'team-Y+alerts@example.org'

- name: 'team-Y-pager'

pagerduty\_configs:

- service\_key: <team-Y-key>

- name: 'team-DB-pager'

pagerduty\_configs:

- service\_key: <team-DB-key>

**Testing**

If you want to test the new Alertmanager while running the current version, you can mirror traffic to the new one with a simple nginx configuration similar to this:

server {

server\_name <your\_current\_alertmanager>;

location / {

proxy\_pass http://localhost:9093;

post\_action @forward;

}

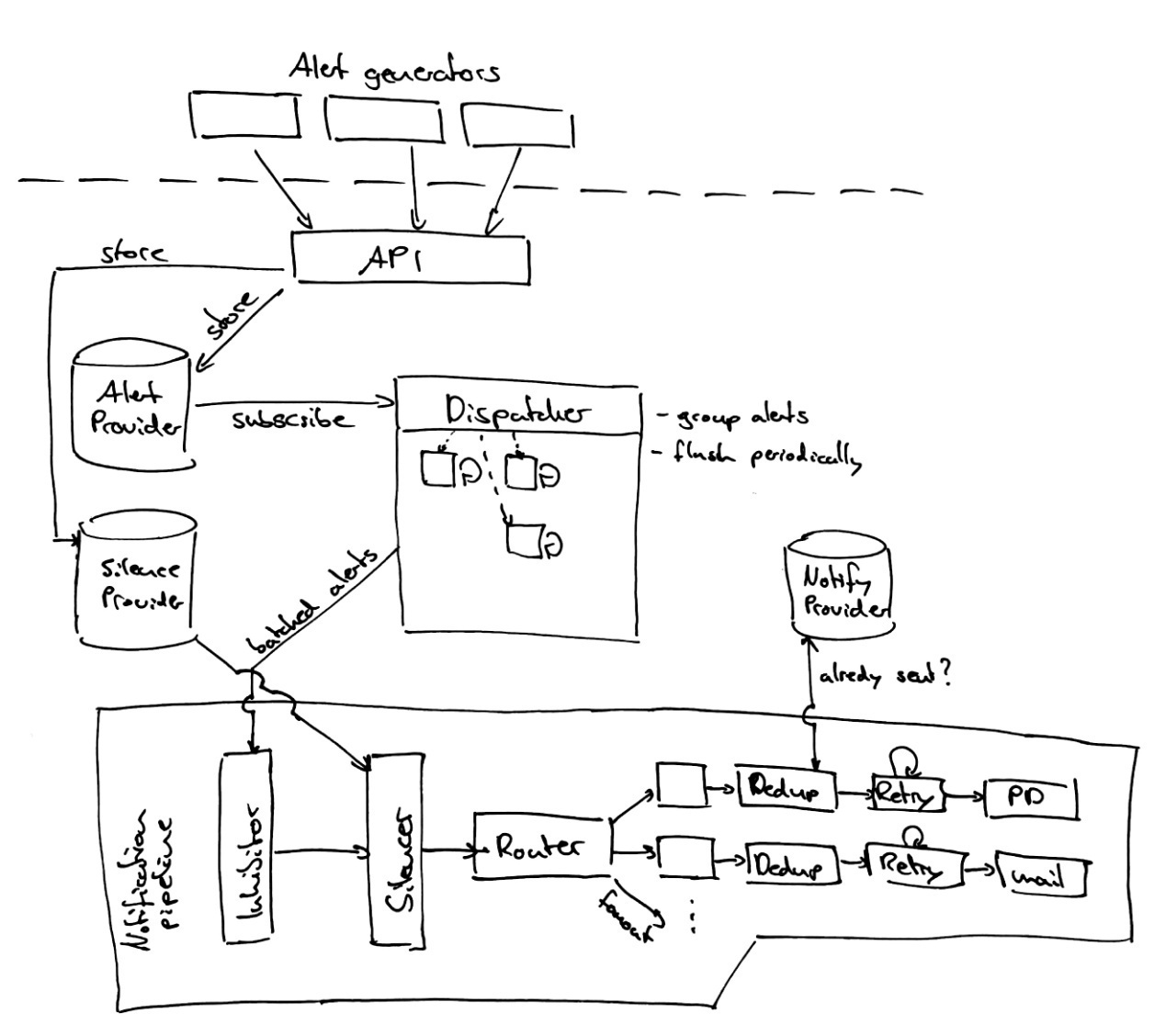
location @forward {

proxy\_pass http://<your\_new\_alertmanager>:9093;

}

}

**Architecture**



# Best Practices

## Metric and label naming | Prometheus

https://prometheus.io/docs/practices/naming/

The metric and label conventions presented in this document are not required for using Prometheus, but can serve as both a style-guide and a collection of best practices. Individual organizations may want to approach some of these practices, e.g. naming conventions, differently.

**Metric names**

A metric name...

* ...should have a (single-word) application prefix relevant to the domain the metric belongs to. The prefix is sometimes referred to as namespace by client libraries. For metrics specific to an application, the prefix is usually the application name itself. Sometimes, however, metrics are more generic, like standardized metrics exported by client libraries. Examples:
  + **prometheus**\_notifications\_total (specific to the Prometheus server)
  + **process**\_cpu\_seconds\_total (exported by many client libraries)
  + **http**\_request\_duration\_seconds (for all HTTP requests)
* ...must have a single unit (i.e. do not mix seconds with milliseconds, or seconds with bytes).
* ...should use base units (e.g. seconds, bytes, meters - not milliseconds, megabytes, kilometers).
* ...should have a suffix describing the unit, in plural form. Note that an accumulating count has total as a suffix, in addition to the unit if applicable.
  + http\_request\_duration\_**seconds**
  + node\_memory\_usage\_**bytes**
  + http\_requests\_**total** (for a unit-less accumulating count)
  + process\_cpu\_**seconds\_total** (for an accumulating count with unit)
* ...should represent the same logical thing-being-measured across all label dimensions.
  + request duration
  + bytes of data transfer
  + instantaneous resource usage as a percentage

As a rule of thumb, either the sum() or the avg() over all dimensions of a given metric should be meaningful (though not necessarily useful). If it is not meaningful, split the data up into multiple metrics. For example, having the capacity of various queues in one metric is good, while mixing the capacity of a queue with the current number of elements in the queue is not.

**Labels**

Use labels to differentiate the characteristics of the thing that is being measured:

* api\_http\_requests\_total - differentiate request types: type="create|update|delete"
* api\_request\_duration\_seconds - differentiate request stages: stage="extract|transform|load"

Do not put the label names in the metric name, as this introduces redundancy and will cause confusion if the respective labels are aggregated away.

**CAUTION:** Remember that every unique combination of key-value label pairs represents a new time series, which can dramatically increase the amount of data stored. Do not use labels to store dimensions with high cardinality (many different label values), such as user IDs, email addresses, or other unbounded sets of values.

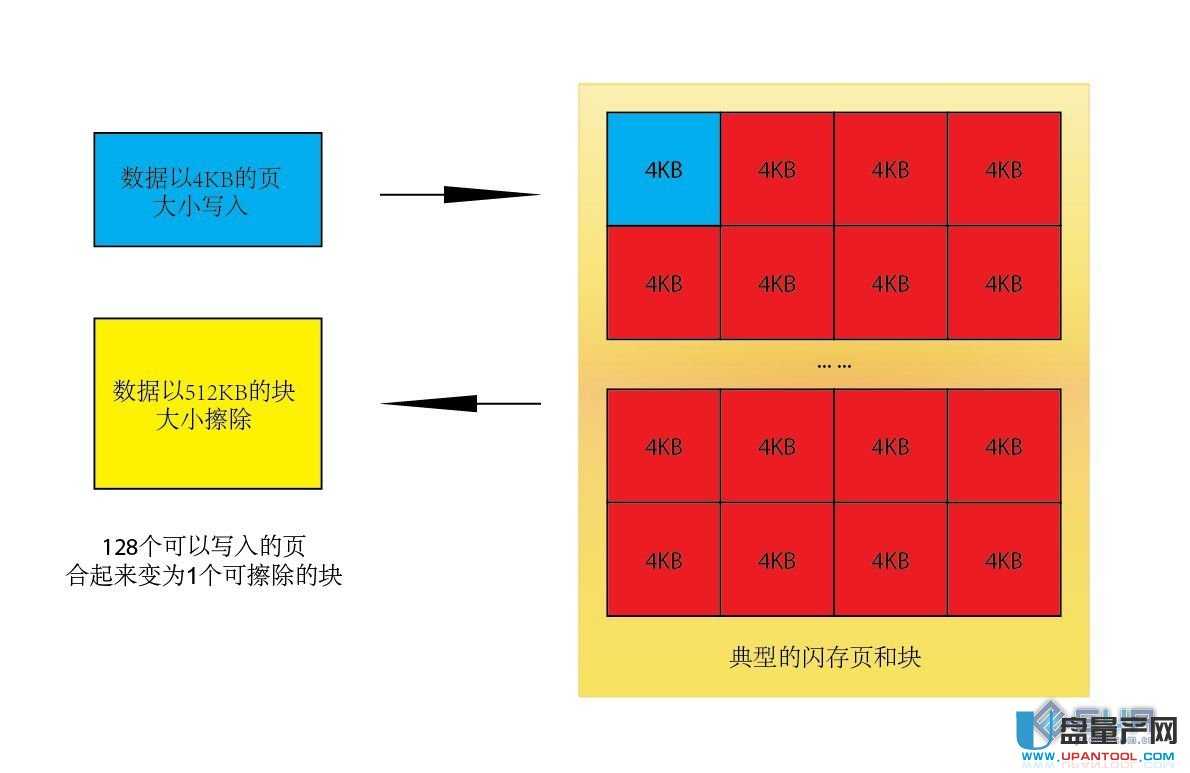
# 其他

## 2@SSD的写入放大技术是什么-Write amplification

-固态硬盘教程-U盘量产网

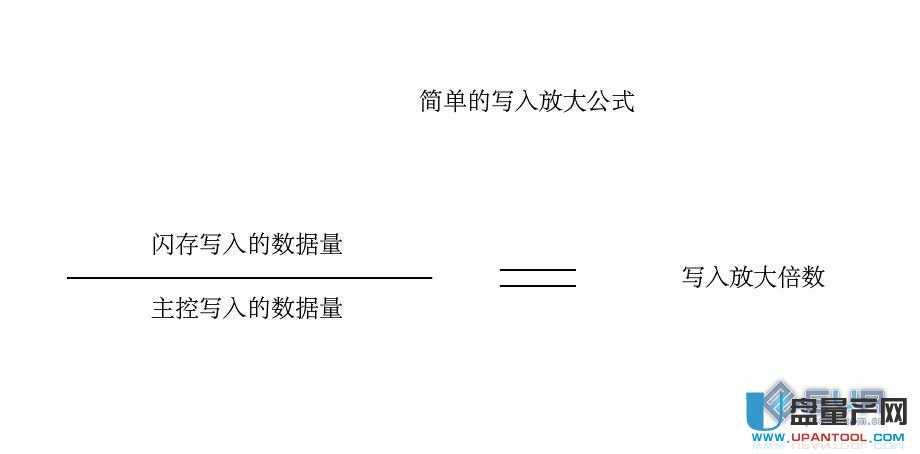
http://www.upantool.com/jiaocheng/ssd/2012/1499.html

**基本SSD操作方式**

写入放大（WA）是闪存和固态硬盘之间相关联的一个属性，因为闪存必须先删除才能改写（我们也叫“编程“），在执行这些操作的时候，移动（或重写）用户数 据和元数据(metadata)不止一次。这些多次的操作，不但增加了写入数据量，减少了SSD的使用寿命，而且还吃光了闪存的带宽（间接地影响了随机写 入性能）。许多因素会影响到SSD的写入放大，下面我就来稍微详细的解释一下。  
  
早在2008年，Intel公司和SiliconSystems公司（2009 年被西部数字收购）第一次提出了写入放大并在公开稿件里用到这个术语。他们当时的说法是，写入算法不可能低于1，但是这种说法在2009年被 SandForce打破，SandForce说他们的写入放大是0.5。  
  
  
由于闪存的运作特性，数据不能像在普通机械硬盘里那样被直接覆盖。当数据第一次写入SSD的时候，由于SSD内所有的颗粒都为已擦除状态，所以数据能够以 页为最小单位直接写入进去（一般是4K，参考颗粒资料），SSD上的主控制器，使用了逻辑和物理的映射系统来管理着闪存。（逻辑我们一般指的是LBA,而 物理指的是FTL)。当有新的数据写入时需要替换旧的数据时，SSD主控制器将把新的数据写入到另外的空白的闪存空间上（已擦除状态）然后更新逻辑LBA 地址来指向到新的物理FTL地址。而旧的地址内容就变成了无效的数据，但是要在上面再次写入的话，就需要首先擦除掉这个无效数据。（闪存运作特性，写入最 小单位是页，而擦除最小单位是块，一般为128~256个页）

那么问题就来了，闪存有编程和擦除的次数限制，这样每次的编程/擦除就叫做1个P/E(program/erase cycles)周期，大家应该都知道MLC一般是5000~10000次，而SLC是10万次左右（查闪存资料）。也就是说写入放大越低，P/E周期就越 少，闪存寿命就越久。

**写入放大的计算**

2008年，Intel公司和SiliconSystems公司（2009 年被西部数字收购）第一次提出了写入放大并在公开稿件里用到这个术语。所有的SSD都有一个写入放大值，这个数值是非固定的，取决于这个SSD写入的数据 是随机的还是持续的？写入量是多少？主控做了那些操作，等等。  
  
计算写入放大的公式大致是这样:  
  
对于单次操作，最简单的例子，比如我要写入一个4KB的数据，最坏的情况就是，一个块里已经没有干净空间了，但是有无效数据可以擦除，所以主控就把所有的 数据读出来，擦除块，再加上这个4KB新数据写回去，这个操作带来的写入放大就是: 我实际写4K的数据，造成了整个块（512KB）的写入操作，那就是128倍放大。同时带来了原本只需要简单的写4KB的操作变成读取(512KB)，擦 (512KB)，改写(512KB)，造成了延迟大大增加，速度慢是自然了。

**影响写入放大的因素**

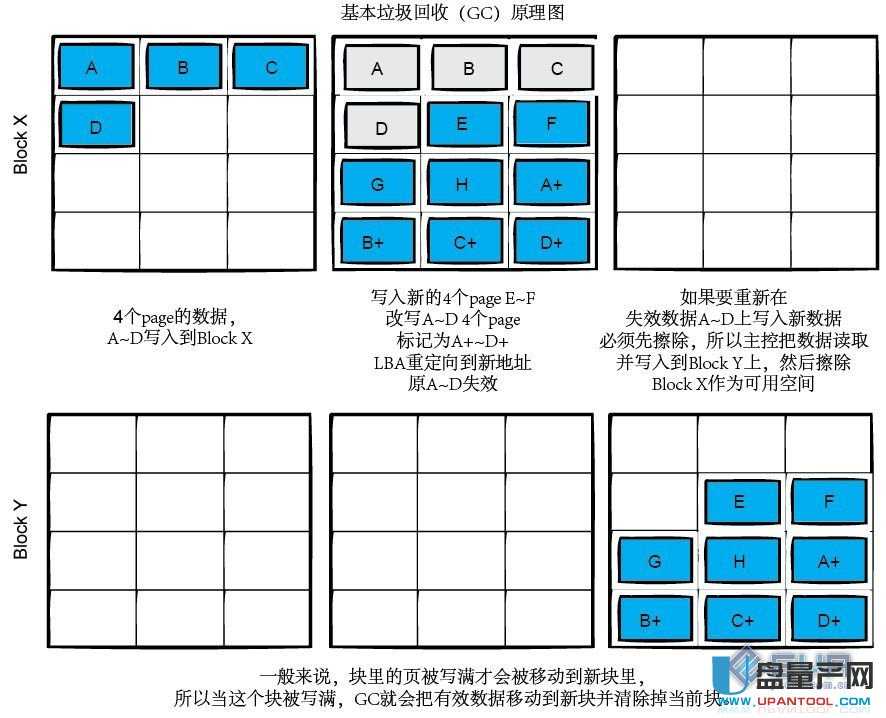
许多因素影响SSD的写入放大。下面我列出了主要因素，以及它们如何影响写放大。

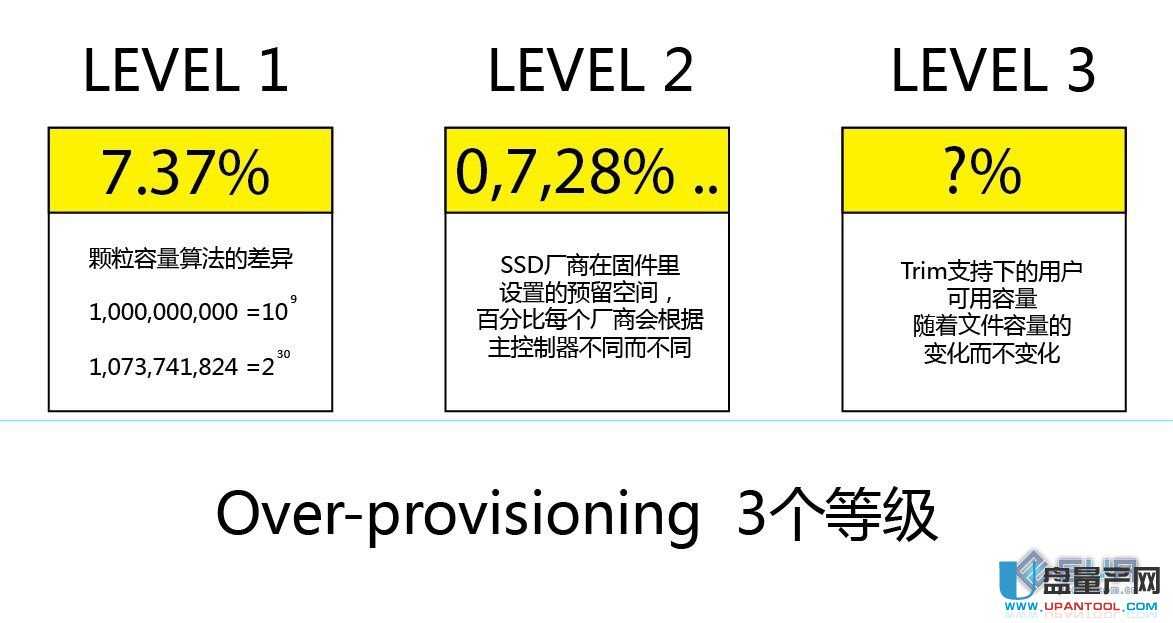
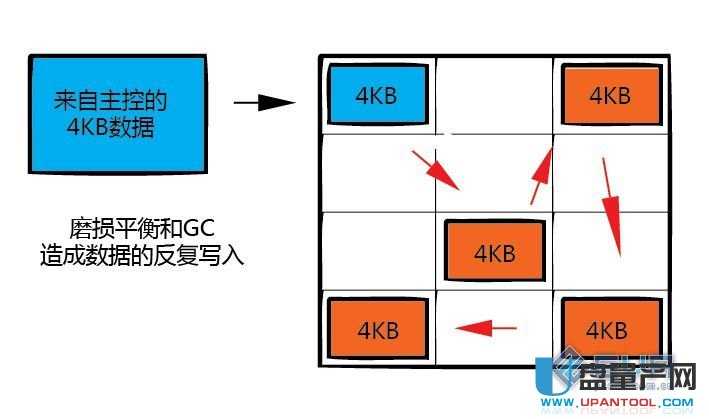
1. 垃圾回收(GC) Garbage collection    ---   虽然增加了写入放大，但是速度有提升。

这个比较特殊的算法用来整理，移动，合并，删除闪存块来提升效率。

2. 预留空间（OP) Over-provisioning  ---  减少写入放大，好。(预留空间越大，写入放大越低）    
  
在SSD的闪存上划出一部分空间留给主控做优化，用户不能操作的空间。  
  
3. TRIM   开启后可以减少写入放大，好。  
  
一个ATA指令，由操作系统发送给SSD主控，告诉主控哪些数据是无效的并且可以不用做垃圾回收操作。  
  
4. 可用容量   减少写入放大，好。(可用空间越大，写入放大越低）  
  
用户使用中没有用到的空间，需要有Trim支持，不然不会影响写入放大。  
  
5. 安全擦除 Secure Erase  减少写入放大，好  
  
清除所有用户数据和相关元数据，让SSD重置到初始性能。  
  
6. 静动数据分离 Separating Static and Dynamic Data 减少写入放大，好  
  
分组常改写和不常改写的数据。  
  
7. 持续写入 Sequential writes        减少写入放大，好  
  
理论上来说，持续写入的写入放大为1，但是某些因素还是会影响这个数值。  
  
8. 随机写入 Random writes  提高写入放大，不好  
  
随机写入会写入很多非连续的LBA,将会大大提升写入放大。  
  
9. 磨损平衡（WL） Wear Leveling    直接提高写入放大，不好  
  
确保闪存的每个块被写入的次数相等的一种机制。

**详细解释**

垃圾回收 Garbage collection  
  
 [](http://www.upantool.com/uploads/allimg/120701/2320163T3-2.jpg)  
一旦SSD的所有块都已经写入了一次，SSD主控制器将会初始化那些包含无效数据的块。（陈旧数据，这些块里的数据已经被更新的数据替换，已经无效了，没 了LBA地址），现在他们正在等待被删除，以便新的数据可以写入其中,如何优化并整理这些个等待被删除的无效数据，这个算法被称为垃圾收集（GC）。我们 可以看出这个操作是要有前提的，就是SSD必须要支持Trim技术，不然GC就显不出他的优势了（这也是为啥目前只有支持Trim的SSD才会有GC功 能），而GC的本质区别是它们何时处理？效率多少？

数据的最小写入单位是页，然而擦除的最小单位是块（大小取决于闪存，自己查资料，一般128~256页）。如果在块上的某些页中的数据不再需要，与在该块 内好的数据的其他所有页必须全部读取并重新写入到新的已擦除的块内。这个操作叫做Copy Block，每个主控都会带。（包括U盘主控，这也是为了磨损平衡考虑）然后主控制器再删除掉这个块，用来给下一次写入数据用。这种操作一切指令来自主控 而非用户的叫做GC，将会影响写入放大。请记住，GC有点像整理硬盘，所以要保证有一定的可用容量，可用容量越大，GC效率越高  
  
GC分为后台GC和主动GC  
  
我们知道垃圾的收集过程，包括读取和重写数据到闪存。意味着这样操作会大大降低主控的性能，因为占用了主控的能力和带宽。所以，一些SSD控制器采用所谓 的后台垃圾收集算法（也称为闲置垃圾收集），该控制器会使用空闲的时间来做垃圾收集，让主控在使用时一直保持高性能。试想一下如果垃圾回收把所有的空间都 整理合并过了，那样在性能提升的同时，也增加了写入放大，所以像barefoot主控的SSD（闲置GC）一般只垃圾回收一小部分的空白空间来限制过多的 写入操作。另一种方案就是主动GC，这需要有相当性能的主控制器，来保证在操作数据的同时进行GC操作，这类GC适合在服务器里用到，因为个人用户可以把 电脑闲置了做GC，但是服务器可不行，所以要保证性能的话必须在运行的同时做GC，这对主控制器的性能提出了很高的要求，SandForce的主控就是这 类。。  
  
手动GC  
  
记得以前我曾讨论过手动GC的操作  
  
乘着现在正好讨论到写入放大，我也再次强调下为何写入GC能成功的道理：  
  
不管你的SSD是不是支持Trim或者在RAID阵列状态下，手动GC或多或少都会有点作用（特殊的除外），为何？  
  
SSD的NAND颗粒有2个状态，物理上来说就是颗粒充电表示1，颗粒放电表示0，擦除数据意味着全盘写1（充电）。颗粒必须以块为最小单位一下子充电， 能以页为单位一个个放电。前面我们得知，如果SSD不支持Trim的话（RAID阵列目前都不支持），在全部SSD写满后，主控并不知道颗粒的块中哪些数 据是无效的，所以它认为他们都有效，操作系统的LBA却知道（因为新数据有LBA,无效数据的LBA已经被重定向了），所以不支持Trim的SSD就意味 着全盘颗粒写满后（指的是颗粒写满，其中包括有效和无效数据，不是我们通常看到的系统里可用容量满）再写入数据要等待写前的擦除操作，系统就会在没用到的 LBA地址下做写入操作。说白了就是系统知道哪里可以写而SSD主控不知道。手动GC正是利用了这一点，做了全盘填FF操作，FF在逻辑上就是1，那么就 是全盘颗粒里的无效数据处填1，1在NAND颗粒上代表充电，代表擦除。所以随着用户的可用容量越来越少，然后再一下子删除这个生成出来的大文件，代表了 全部可用容量区域都为逻辑1（擦除状态），这个状态就是GC操作后删除无效数据区块后的状态，所以叫做手动GC，自然这些区块在之后的操作中可以直接写入 而不需要再擦除了。  
  
预留空间 Over-provisioning  
  
预留空间一般是指用户不可操作的容量，为实际物理闪存容量减去用户可用容量。这块取用一般被用来做优化，包括磨损均衡，GC,Trim和坏块映射。  
  
预留空间分为3层：  
[](http://www.upantool.com/uploads/allimg/120701/23201C012-3.jpg)  
第一层为固定的7.37%，这个数字是如何得出的哪？我们知道机械硬盘和SSD的厂商容量是这样算的，1GB是1,000,000,000字节（10的9 次方），但是闪存的实际容量是每GB=1,073,741,824，(2的30次方) ，2者相差7.37%。所以说假设1块128GB的SSD，用户得到的容量是128,000,000,000字节，多出来的那个7.37%就被主控固件用 做OP了。  
  
第二层来自制造商的设置，通常为0%，7%和28%等，打个比方，对于128G颗粒的SandForce主控SSD,市场上会有120G和100G两种型号卖，这个取决于厂商的固件设置，这个容量不包括之前的第一层7.37%。  
  
第三层是用户在日常使用中可以分配的预留空间，像Fusion-IO公司还给用户工具自己调节大小来满足不同的耐用度和性能，而用户也可以自己在分区的时候，不分到完全的SSD容量来达到同样的目的。（要有Trim支持）  
  
预留空间虽然让SSD的可用容量小了，但是带来了减少写入放大，提高耐久，提高性能的效果。  
  
TRIM  
  
Trim是一种SATA命令，他能让操作系统在删除某个文件或者格式化后告诉SSD主控这个数据块不再需要了。  
  
一般情况下，当LBA被操作系统更新后，只有随着之后的每次数据写入（其实等于覆盖），SSD主控制器才知道这个地址原来早已经失效了。（之前认为每个数 据都是有效的）在Win7里，由于Trim的引入解决了这个问题，当某些文件被删除或者格式化了整个分区，操作系统把Trim指令和在操作中更新的LBA 一起发给SSD主控制器（其中包含了无效数据地址），这样在之后的GC操作中，无效数据就能被清空了，减少了写入放大同时也提升了性能。  
  
可以看这个帖子： [Trim解析](http://www.upantool.com/jiaocheng/ssd/2012/1500.html)  
  
Trim的依赖性和局限性  
  
1. Trim命令需要SSD的支持，某些老型号的SSD可以靠刷新固件得到Trim支持（G2,barefoot,YK40），或者用一些独特的工具（barefoot wiper)提取出系统里所有无效的LBA告诉SSD主控并清除。  
  
2. Trim命令之后，速度并不一定是立马就能提升的，因为Trim后的干净空间可能随机的包含在每个块里，只有等着多次的copy block操作和主控的GC操作才能明显感觉到速度的提升。  
  
3. 就算操作系统，驱动，SSD主控固件都满足Trim命令了，也不代表在某些特定环境下能工作，比如RAID阵列和数据库（至少到目前为止）。  
  
可用空间  
  
SSD控制器会使用所有的可用空间做垃圾回收和磨损均衡。保证一定的可用空间可以提升SSD效率，减少写入放大。（前提是支持Trim)  
  
安全擦除 Secure erase  
  
ATA安全擦除命令用来清除在磁盘上的所有用户数据，这个指令会让SSD回到出厂性能（最优性能，最少写入放大），但是随着之后的使用，GC，写入放大又会慢慢增加回来。  
  
许多软件使用ATA安全擦除指令来重置磁盘，最著名的为HDDErase。对SSD来说，重置就是全盘加电（逻辑1），瞬间即可完成清除所有数据让SSD回到初始状态。  
  
静动数据分离 Separating Static and Dynamic Data  
  
高端SSD主控制器支持静态和动态数据的分离处理，此操作要求SSD主控制器对LBA里经常写入（动态数据，热数据）和不经常写入（静态数据，冷数据）的 数据块进行归类，因为如果块里包含了静态和动态数据，在做GC操作的时候会为了改写其实没必要改写的静态数据而增加了写入放大，所以把包含静态数据的块归 类后，因为不常改写，减少了写入放大。但是迟早SSD主控会把这块静态的数据转移到别的地方来弥补平衡磨损。（因为静态数据占着的数据块一直不改写，编程 次数低于平均值的话，会造成颗粒磨损不平衡，违背了WL，真够矛盾的。）  
  
持续写入  
  
当SSD持续的写入数据时，写入放大一般为1，原因是随着数据写入，整个块都是持续的填充着同一个文件，如果系统确认这个文件需要改写或者删除，整个块都 可以被标记为无效（需要Trim支持），自然就不需要之后的GC操作了。（读取整个块并写入新的块）这个块只需要擦除，比读，改，写更快速有效。  
  
随机写入  
  
一个SSD主控的随机写入峰值速度一般发生在安全擦除后，完全GC，全盘Trim，或新安装的状态下。  
而随机写入的能力取决于主控制器的1.通道数。2.固件效率。3.闪存颗粒写入页面的性能。  
然后就是写入放大的多少，越接近于1越好，小于1更那好。  
当全盘颗粒都被写过后，GC功能就将被启用，速度就会受到影响，之后的写入放大就会达到SSD主控制器的最大倍数。大量的随机小文件的写入是“闪存杀手”。  
  
磨损平衡（WL） Wear Leveling  
[](http://www.upantool.com/uploads/allimg/120701/23201A035-4.jpg)  
假设一个特定的块被持续的编程写入而不编程写入到别的块，那么这个块将很快被消耗掉编程寿命，造成整个SSD的报废。处于这个原因，SSD主控制器要平均 分配每个块的编程次数，这个技术叫做磨损平衡。在最乐观的情况下，这个技术会让全盘的颗粒磨损程度接近并同时报废。可惜的是这个技术要牺牲写入放大，假设 对于冷数据，必须经常的移动到别的块，再把热数据移过来，保证2边的块都是一样的磨损度，无谓的增加了写入次数。关键就是要找一个最优化的算法来尽可能的 同时最佳化这2个矛盾的条件。  
  
总结：  
  
写入放大是个很关键的SSD指标，我们知道闪存的写入速度比读取速度慢的多，所以现在的SSD主控制器用多通道来提升写入速度（相对写入速度，读取速度其实早已经不是啥大问题了），但是如果写入放大太高，意味着一样的操作需要写入更多的数据，这样速度自然就快不起来了。  
  
废话那么多，就是想说好主控和坏主控之间的性能差别主要就是体现在放大上，你想想，假设大家都一样的通道数，颗粒也相同，可是差别为何那么大？说白了就是一个写的少而另个写的多了嘛。一切算法技术的根本目标是要尽可能的减少写入放大。

## 运维服务器手段（监控宝，Nagios，百度通告平台）

- 在思索中前行！ - 博客频道 - CSDN.NET

http://blog.csdn.net/txl199106/article/details/41680437

[**站在“巨人”的肩膀上运维**](http://blog.csdn.net/sylcc_/article/details/40475097)

现实问题

之前在论坛看到一个运维工程师的帖子，内容如下：

“现在的一个IT工作者最头疼的就是加班，秃顶的是IT工作者最多、单身的是IT工作者最多、没有约会，没有休闲，没有旅游还是IT工作者最多。这可怎么办呢？我是一名IT运维工程师，每天的工作量很大，更不敢离开机器半步，长期跟设备、服务器打交道，因为怕万一机器出毛病，自己负担不起业务上的麻烦，又怕经理的“炒鱿鱼”，况且公司还不肯出钱雇很多IT运维人员，让我一个人看着这么多的设备，真是连眼都不敢眨一下，雇来几个人也是呆不长就走人，公司一直没有一个专门潜心在这干的，也招不到专门的运维人员，出了大事，还显不出我的本领，经理直接去找专门的网络公司协助解决，在以上这种状况下，让运维工程师情何以堪？所以我请大家帮忙推荐有没有好用、免费的运维软件，主要就是能监控服务器和网络设备就行，还有没有可以实现自动化巡检的软件？求大家推荐....................  
求助！！！”

产生原因

上述的情况还是处于传统运维管理方式，这种传统的运维管理方式让运维工程师疲惫不堪，主要表现在两个方面：

1， 运维工程师被动，效率低。在运维过程，只有当事件已经发生并且对业务照成影响时才能被发现和处理，这种被动“救火”使运维工程师终日忙碌，运维质量很难提高，导致业务部门对运维部门的服务满意度不高，这种来自其它部门的不满也增加了运维工程师的压力。

2， 缺乏运维技术工具。随着技术的发展和多元化，企业的IT系统越来越复杂，各式各样的网络设备、服务器以及在服务器上运行的各种服务让运维工程师难以应付，即使加班加点维护，也可能因为设备或者网络的原因导致服务不可访问，给公司带来不可挽回的损失。出现这种情况的原因就是没有使用高效的运维技术工具进行监控，通告，让运维工程师能够快速主动处理。

解决方案

使用监控工具

监控宝

监控宝成立于2009年，是一家面向企业或个人站长提供网站监控和预警服务的网站，目前监控宝可以监控的内容包括：网站的可访问性以及速度；服务器硬件性能（CPU、内存、宽带流量、磁盘空间、负载等）；服务器软件性能（Apache、[**MySQL**](http://lib.csdn.net/base/mysql) 等）。

主要功能

* 站点监控
* 服务和应用监控
* 服务器性能监控
* 内容监控
* 用户访问速度监控
* 警告通知

主要是通告web的方式设置，上手相对容易。

Nagios

Nagios是一款开源的免费网络监视工具，Nagios能监视所指定的本地或远程主机以及服务，同时提供异常通知功能等。

主要功能

* 网络服务监控（SMTP、POP3、HTTP、NNTP、ICMP、SNMP、FTP、SSH）
* 主机资源监控（CPU load、disk usage、system logs），也包括Windows主机（使用NSClient++ plugin）
* 可以指定自己编写的Plugin通过网络收集数据来监控任何情况（温度、警告……）
* 可以通过配置Nagios远程执行插件远程执行脚本
* 远程监控支持SSH或SSL加通道方式进行监控
* 简单的plugin设计允许用户很容易的开发自己需要的检查服务，支持很多开发语言（shell scripts、C++、Perl、ruby、Python、PHP、C#等）
* 包含很多图形化数据Plugins（Nagiosgraph、Nagiosgrapher、PNP4Nagios等）
* 可并行服务检查
* 能够定义网络主机的层次，允许逐级检查，就是从父主机开始向下检查
* 当服务或主机出现问题时发出通告，可通过email, pager, sms 或任意用户自定义的plugin进行通知
* 能够自定义事件处理机制重新激活出问题的服务或主机
* 自动日志循环
* 支持冗余监控
* 包括Web界面可以查看当前网络状态，通知，问题历史，日志文件等

主要通过配置文件进行配置，但是开源免费，拥有大量的插件可以完成日常的监控需求，对入门门槛相对比较高。

使用通告工具

百度通告平台

在监控工具检测到异常后，在通知运维工程师的过程出现遗漏的情况也会造成不可估计的后果。所以需要使用专业的通告工具——百度通告平台来保证通告无遗漏。

主要功能

* 多渠道通告方式：一种通告、多个渠道，多重渠道保障通告及时告知目标。随心所欲的选用通告渠道：电话、SNS、短信、邮件等。
* 报警管理：提供web版和手机APP版，可时刻响应通告，高效办公处理，让您工作生活两不误。
* 自动逐级报警：报警从不被错过，遗漏的报警会自动升级到合适的等级，发送给合适的人，保证通告无遗漏告知。
* 接入快捷：大量系统组件支持各监控系统（监控宝、加速乐等）高效接入，丰富API使系统接入更加自由定制化。
* 通告群发：通知、报警，通告想发就发，随心随意。多人(组)选择，轻轻点击，一触即发。

[Linux下Nagios的安装与配置](http://www.cnblogs.com/mchina/archive/2013/02/20/2883404.html" \t "_blank)

一、Nagios简介

[Nagios logo](http://images.cnitblog.com/blog/370046/201301/30172010-1a4517cdb03747cc8d37837d6f550d8a.png)

　　Nagios是一款开源的电脑系统和网络监视工具，能有效监控Windows、[**Linux**](http://lib.csdn.net/base/linux)和Unix的主机状态，交换机路由器等网络设置，打印机等。在系统或服务状态异常时发出邮件或短信报警第一时间通知网站运维人员，在状态恢复后发出正常的邮件或短信通知。

　　Nagios原名为NetSaint，由Ethan Galstad开发并维护至今。NAGIOS是一个缩写形式: "Nagios Ain't Gonna Insist On Sainthood" Sainthood 翻译为圣徒，而"Agios"是"saint"的希腊表示方法。Nagios被开发在Linux下使用，但在Unix下也工作得非常好。

主要功能

* 网络服务监控（SMTP、POP3、HTTP、NNTP、ICMP、SNMP、FTP、SSH）
* 主机资源监控（CPU load、disk usage、system logs），也包括Windows主机（使用NSClient++ plugin）
* 可以指定自己编写的Plugin通过网络收集数据来监控任何情况（温度、警告……）
* 可以通过配置Nagios远程执行插件远程执行脚本
* 远程监控支持SSH或SSL加通道方式进行监控
* 简单的plugin设计允许用户很容易的开发自己需要的检查服务，支持很多开发语言（shell scripts、C++、Perl、ruby、Python、PHP、C#等）
* 包含很多图形化数据Plugins（Nagiosgraph、Nagiosgrapher、PNP4Nagios等）
* 可并行服务检查
* 能够定义网络主机的层次，允许逐级检查，就是从父主机开始向下检查
* 当服务或主机出现问题时发出通告，可通过email, pager, sms 或任意用户自定义的plugin进行通知
* 能够自定义事件处理机制重新激活出问题的服务或主机
* 自动日志循环
* 支持冗余监控
* 包括Web界面可以查看当前网络状态，通知，问题历史，日志文件等

二、Nagios工作原理

　　Nagios的功能是监控服务和主机，但是他自身并不包括这部分功能，所有的监控、检测功能都是通过各种插件来完成的。

　　启动Nagios后，它会周期性的自动调用插件去检测服务器状态，同时Nagios会维持一个队列，所有插件返回来的状态信息都进入队列，Nagios每次都从队首开始读取信息，并进行处理后，把状态结果通过web显示出来。

　　Nagios提供了许多插件，利用这些插件可以方便的监控很多服务状态。安装完成后，在nagios主目录下的/libexec里放有nagios自带的可以使用的所有插件，如，check\_disk是检查磁盘空间的插件，check\_load是检查CPU负载的，等等。每一个插件可以通过运行./check\_xxx –h 来查看其使用方法和功能。

　　Nagios可以识别4种状态返回信息，即 0(OK)表示状态正常/绿色、1(WARNING)表示出现警告/黄色、2(CRITICAL)表示出现非常严重的错误/红色、3(UNKNOWN)表示未知错误/深黄色。Nagios根据插件返回来的值，来判断监控对象的状态，并通过web显示出来，以供管理员及时发现故障。

四种监控状态

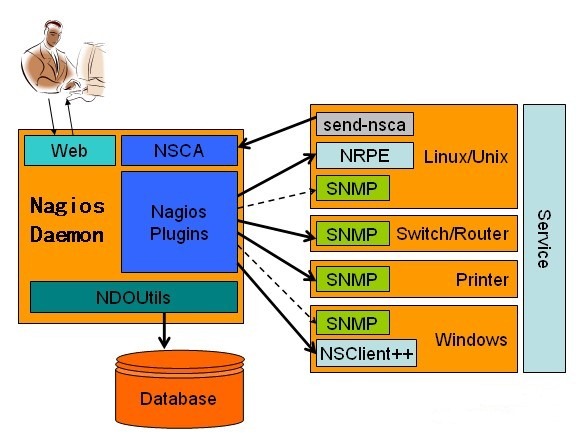
[](http://images.cnitblog.com/blog/370046/201301/30172012-5e48d4a0cbc542b88faf0cdaab257bef.jpg)

　　再说报警功能，如果监控系统发现问题不能报警那就没有意义了，所以报警也是nagios很重要的功能之一。但是，同样的，Nagios 自身也没有报警部分的代码，甚至没有插件，而是交给用户或者其他相关开源项目组去完成的。

　　Nagios 安装，是指基本平台，也就是Nagios软件包的安装。它是监控体系的框架，也是所有监控的基础。

　　打开Nagios官方的文档，会发现Nagios基本上没有什么依赖包，只要求系统是Linux或者其他Nagios支持的系统。不过如果你没有安装apache（http服务），那么你就没有那么直观的界面来查看监控信息了，所以apache姑且算是一个前提条件。关于apache的安装，网上有很多，照着安装就是了。安装之后要检查一下是否可以正常工作。

　　知道Nagios 是如何通过插件来管理服务器对象后，现在开始研究它是如何管理远端服务器对象的。Nagios 系统提供了一个插件NRPE。Nagios 通过周期性的运行它来获得远端服务器的各种状态信息。它们之间的关系如下图所示：

[](http://images.cnitblog.com/blog/370046/201301/30172017-a0cab434c3184ee48163b3c2f86d0299.jpg)

Nagios 通过NRPE 来远端管理服务

1. Nagios 执行安装在它里面的check\_nrpe 插件，并告诉check\_nrpe 去检测哪些服务。

2. 通过SSL，check\_nrpe 连接远端机子上的NRPE daemon

3. NRPE 运行本地的各种插件去检测本地的服务和状态(check\_disk,..etc)

4. 最后，NRPE 把检测的结果传给主机端的check\_nrpe，check\_nrpe 再把结果送到Nagios状态队列中。

5. Nagios 依次读取队列中的信息，再把结果显示出来。

三、实验环境

|  |  |  |  |
| --- | --- | --- | --- |
| Host Name | OS | IP | Software |
| Nagios-Server | CentOS release 6.3 (Final) | 192.168.1.108 | Apache、Php、Nagios、nagios-plugins |
| Nagios-Linux | CentOS release 5.8 (Final) | 192.168.1.111 | nagios-plugins、nrpe |
| Nagios-Windows | Windows XP | 192.168.1.113 | NSClient++ |

Server 安装了nagios软件，对监控的数据做处理，并且提供web界面查看和管理。当然也可以对本机自身的信息进行监控。

Client 安装了NRPE等客户端，根据监控机的请求执行监控，然后将结果回传给监控机。

防火墙已关闭/iptables: Firewall is not running.

SELINUX=disabled

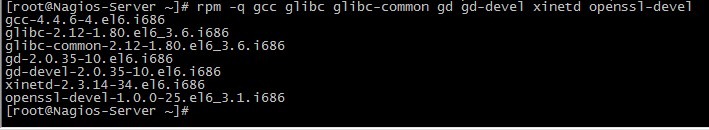
四、实验目标



五、Nagios服务端安装

5.1 基础支持套件：gcc glibc glibc-common gd gd-devel xinetd openssl-devel

# rpm -q gcc glibc glibc-common gd gd-devel xinetd openssl-devel

[](http://images.cnitblog.com/blog/370046/201301/30172023-cbd356b06f1c4fb796b99237b3554b2f.jpg)

如果系统中没有这些套件，使用yum 安装

# yum install -y gcc glibc glibc-common gd gd-devel xinetd openssl-devel

5.2 创建nagios用户和用户组

# useradd -s /sbin/nologin nagios

# mkdir /usr/local/nagios

# chown -R nagios.nagios /usr/local/nagios

[](http://images.cnitblog.com/blog/370046/201301/30172026-dd983461e7e24d3fa12c39d213616afc.jpg)

查看nagios 目录的权限

# ll -d /usr/local/nagios/

[7](http://images.cnitblog.com/blog/370046/201301/30172027-2efbcf5f62f34eae857b7c48c0f94d02.jpg)

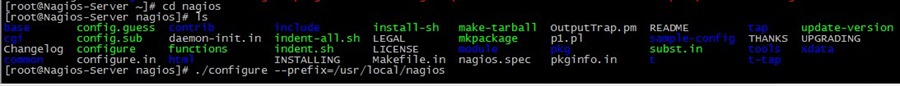
5.3 编译安装Nagios

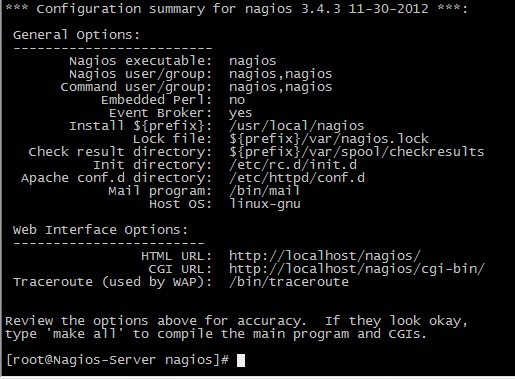
# wget <http://prdownloads.sourceforge.net/sourceforge/nagios/nagios-3.4.3.tar.gz>

# tar zxvf nagios-3.4.3.tar.gz

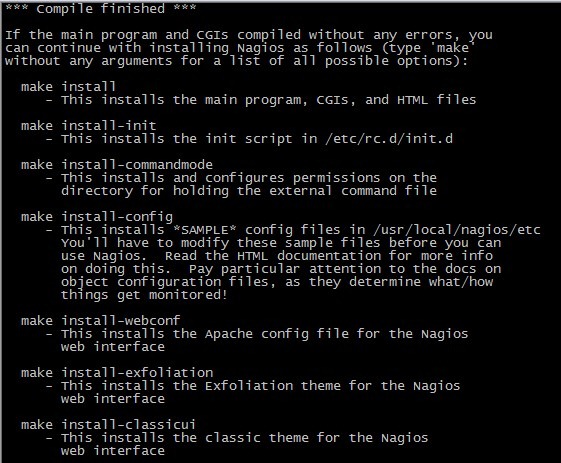
# cd nagios

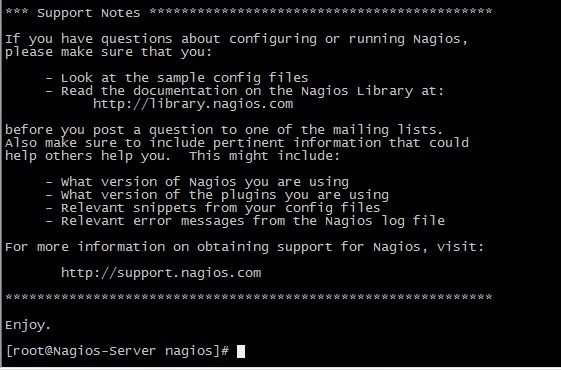
# ./configure --prefix=/usr/local/nagios

[](http://images.cnitblog.com/blog/370046/201301/30172029-51aac459f3a34106af7db6aaaa403d9e.jpg)

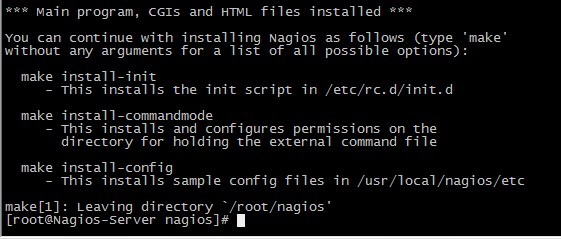
[](http://images.cnitblog.com/blog/370046/201301/30172030-18fdb7961c474b75a2b033e2a4171519.jpg)

# make all

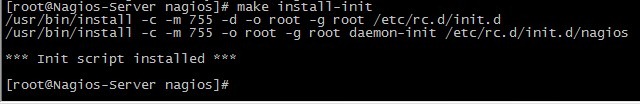
[](http://images.cnitblog.com/blog/370046/201301/30172031-5c6a7463c11d4544bee8f2eed12d7647.jpg)

[](http://images.cnitblog.com/blog/370046/201301/30172034-ebe3fcafffe94564b8161e6b78fe64c8.jpg)

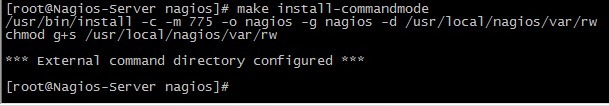
# make install

[](http://images.cnitblog.com/blog/370046/201301/30172035-b01026f0079c477c815351fb70635f40.jpg)

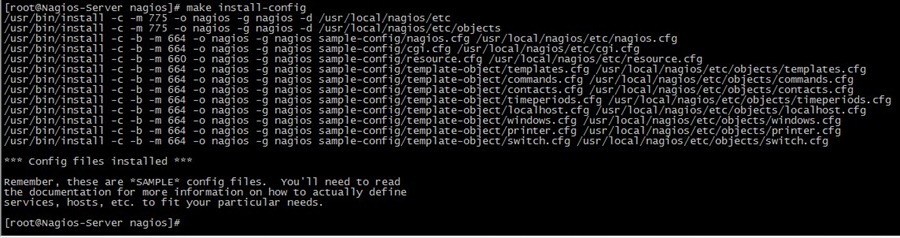
# make install-init

[](http://images.cnitblog.com/blog/370046/201301/30172038-629553af940645d3a1e1b34db39106e7.jpg)

# make install-commandmode

[](http://images.cnitblog.com/blog/370046/201301/30172039-977a570a3633463e905c54ae6b86b8dc.jpg)

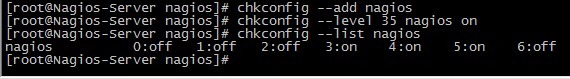
# make install-config

[](http://images.cnitblog.com/blog/370046/201301/30172042-cdce5169280944e3b61f4df6cc33311d.jpg)

# chkconfig --add nagios

# chkconfig --level 35 nagios on

# chkconfig --list nagios

[](http://images.cnitblog.com/blog/370046/201301/30172044-219e4c37980a450c904fb0cb283c73cd.jpg)

5.4 验证程序是否被正确安装

切换目录到安装路径（这里是/usr/local/nagios），看是否存在etc、bin、sbin、share、var 这五个目录，如果存在则可以表明程序被正确的安装到系统了。Nagios 各个目录用途说明如下：

|  |  |
| --- | --- |
| bin | Nagios 可执行程序所在目录 |
| etc | Nagios 配置文件所在目录 |
| sbin | Nagios CGI 文件所在目录，也就是执行外部命令所需文件所在的目录 |
| share | Nagios网页文件所在的目录 |
| libexec | Nagios 外部插件所在目录 |
| var | Nagios 日志文件、lock 等文件所在的目录 |
| var/archives | Nagios 日志自动归档目录 |
| var/rw | 用来存放外部命令文件的目录 |

5.5 安装Nagios 插件

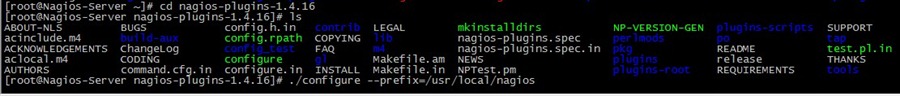
# wget <http://prdownloads.sourceforge.net/sourceforge/nagiosplug/nagios-plugins-1.4.16.tar.gz>

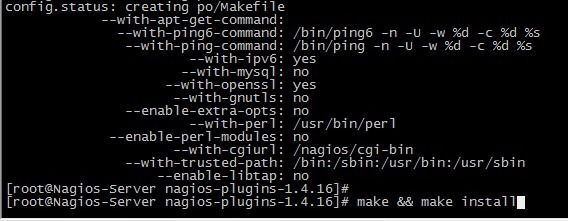
# tar zxvf nagios-plugins-1.4.16.tar.gz

# cd nagios-plugins-1.4.16

# ./configure --prefix=/usr/local/nagios

# make && make install

[](http://images.cnitblog.com/blog/370046/201301/30172045-9f33e94cc4034186b2b81b92370fe259.jpg)

[](http://images.cnitblog.com/blog/370046/201301/30172046-64f935847ba9428d979a2d5c8b0987d2.jpg)

5.6 安装与配置Apache和[**PHP**](http://lib.csdn.net/base/php)

Apache 和Php 不是安装nagios 所必须的，但是nagios提供了web监控界面，通过web监控界面可以清晰的看到被监控主机、资源的运行状态，因此，安装一个web服务是很必要的。   
需要注意的是，nagios在nagios3.1.x版本以后，配置web监控界面时需要php的支持。这里我们下载的nagios版本为nagios-3.4.3，因此在编译安装完成apache后，还需要编译php模块，这里选取的php版本为php5.4.10。

a. 安装Apache

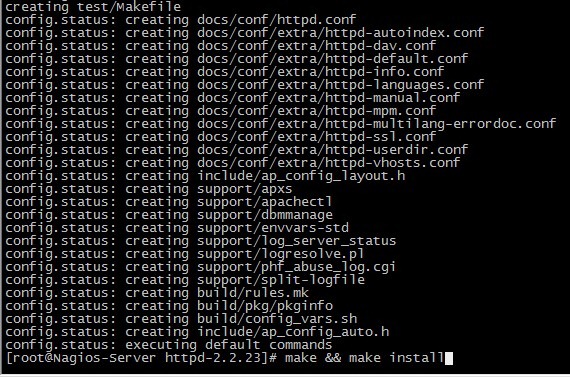
# wget <http://archive.apache.org/dist/httpd/httpd-2.2.23.tar.gz>

# tar zxvf httpd-2.2.23.tar.gz

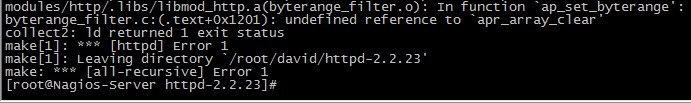
# cd httpd-2.2.23

# ./configure --prefix=/usr/local/apache2

# make && make install

[](http://images.cnitblog.com/blog/370046/201301/30172048-277b391cdc8a4e7da295d6bf9c3dfe6f.jpg)

若出现错误：



则在编译时入加 --with-included-apr 即可解决。

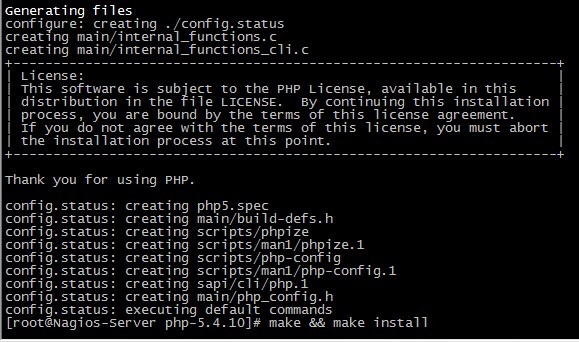
b. 安装Php

# wget <http://cn2.php.net/distributions/php-5.4.10.tar.gz>

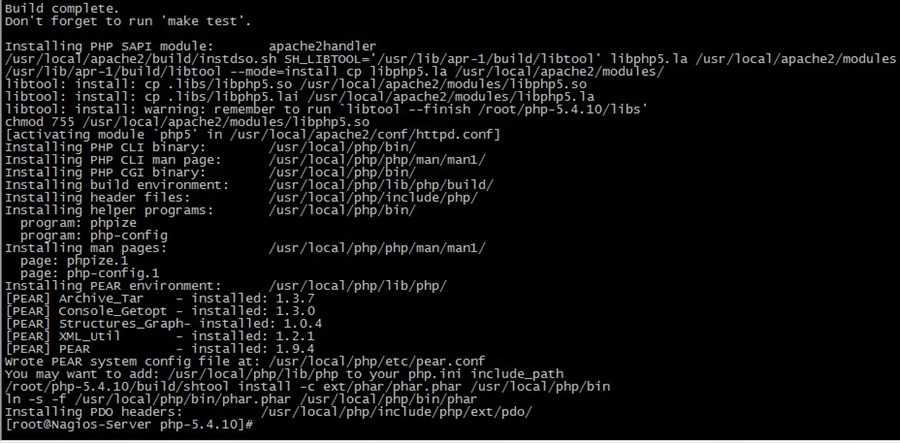
# tar zxvf php-5.4.10.tar.gz

# cd php-5.4.10

# ./configure --prefix=/usr/local/php --with-apxs2=/usr/local/apache2/bin/apxs

[](http://images.cnitblog.com/blog/370046/201301/30172050-6bda75716f614a6ab7f2acf8de422062.jpg)

# make && make install

[](http://images.cnitblog.com/blog/370046/201301/30172054-97af3f5b712041f78d9e44d94ac17bbd.jpg)

c. 配置apache   
找到apache 的配置文件/usr/local/apache2/conf/httpd.conf   
找到：

User daemon

Group daemon

修改为

User nagios

Group nagios

然后找到

<IfModule dir\_module>

　　DirectoryIndex index.html

</IfModule>

修改为

<IfModule dir\_module>

　　DirectoryIndex index.html index.php

</IfModule>

接着增加如下内容：

AddType application/x-httpd-php .php

为了安全起见，一般情况下要让nagios 的web 监控页面必须经过授权才能访问，这需要增加验证配置，即在httpd.conf 文件最后添加如下信息：

复制代码

#setting for nagios

ScriptAlias /nagios/cgi-bin "/usr/local/nagios/sbin"

<Directory "/usr/local/nagios/sbin">

AuthType Basic

Options ExecCGI

AllowOverride None

Order allow,deny

Allow from all

AuthName "Nagios Access"

AuthUserFile /usr/local/nagios/etc/htpasswd //用于此目录访问身份验证的文件

Require valid-user

</Directory>

Alias /nagios "/usr/local/nagios/share"

<Directory "/usr/local/nagios/share">

AuthType Basic

Options None

AllowOverride None

Order allow,deny

Allow from all

AuthName "nagios Access"

AuthUserFile /usr/local/nagios/etc/htpasswd

Require valid-user

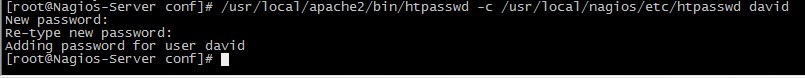
</Directory>

复制代码

d. 创建apache目录验证文件

在上面的配置中，指定了目录验证文件htpasswd，下面要创建这个文件：

# /usr/local/apache2/bin/htpasswd -c /usr/local/nagios/etc/htpasswd david

[](http://images.cnitblog.com/blog/370046/201301/30172056-36df7ebedd9d448a8b38a2ccd4633ebe.jpg)   
这样就在/usr/local/nagios/etc 目录下创建了一个htpasswd 验证文件，当通过[http://192.168.1.108/nagios/](http://172.16.1.124/nagios/) 访问时就需要输入用户名和密码了。

e. 查看认证文件的内容

# cat /usr/local/nagios/etc/htpasswd

[26](http://images.cnitblog.com/blog/370046/201301/30172058-a256928a00cf40769bc3a5815401732b.jpg)

f. 启动apache 服务

# /usr/local/apache2/bin/apachectl start

到这里nagios 的安装也就基本完成了，你可以通过web来访问了。



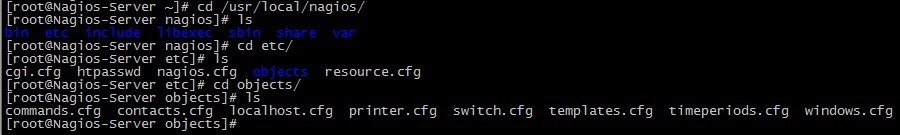


六、配置Nagios

Nagios 主要用于监控一台或者多台本地主机及远程的各种信息，包括本机资源及对外的服务等。默认的Nagios 配置没有任何监控内容，仅是一些模板文件。若要让Nagios 提供服务，就必须修改配置文件，增加要监控的主机和服务，下面将详细介绍。

6.1 默认配置文件介绍

Nagios 安装完毕后，默认的配置文件在/usr/local/nagios/etc目录下。

[](http://images.cnitblog.com/blog/370046/201301/30172103-011c4098757a465fa8e4462f89768c84.jpg)

每个文件或目录含义如下表所示：

|  |  |
| --- | --- |
| 文件名或目录名 | 用途 |
| cgi.cfg | 控制CGI访问的配置文件 |
| nagios.cfg | Nagios 主配置文件 |
| resource.cfg | 变量定义文件，又称为资源文件，在些文件中定义变量，以便由其他配置文件引用，如$USER1$ |
| objects | objects 是一个目录，在此目录下有很多配置文件模板，用于定义Nagios 对象 |
| objects/commands.cfg | 命令定义配置文件，其中定义的命令可以被其他配置文件引用 |
| objects/contacts.cfg | 定义联系人和联系人组的配置文件 |
| objects/localhost.cfg | 定义监控本地主机的配置文件 |
| objects/printer.cfg | 定义监控打印机的一个配置文件模板，默认没有启用此文件 |
| objects/switch.cfg | 定义监控路由器的一个配置文件模板，默认没有启用此文件 |
| objects/templates.cfg | 定义主机和服务的一个模板配置文件，可以在其他配置文件中引用 |
| objects/timeperiods.cfg | 定义Nagios 监控时间段的配置文件 |
| objects/windows.cfg | 监控Windows 主机的一个配置文件模板，默认没有启用此文件 |

6.2 配置文件之间的关系

在nagios的配置过程中涉及到的几个定义有：主机、主机组，服务、服务组，联系人、联系人组，监控时间，监控命令等，从这些定义可以看出，nagios各个配置文件之间是互为关联，彼此引用的。

成功配置出一台nagios监控系统，必须要弄清楚每个配置文件之间依赖与被依赖的关系，最重要的有四点：

第一：定义监控哪些主机、主机组、服务和服务组；   
第二：定义这个监控要用什么命令实现；   
第三：定义监控的时间段；   
第四：定义主机或服务出现问题时要通知的联系人和联系人组。

6.3 配置Nagios

为了能更清楚的说明问题，同时也为了维护方便，建议将nagios各个定义对象创建独立的配置文件：

* 创建hosts.cfg文件来定义主机和主机组
* 创建services.cfg文件来定义服务
* 用默认的contacts.cfg文件来定义联系人和联系人组
* 用默认的commands.cfg文件来定义命令
* 用默认的timeperiods.cfg来定义监控时间段
* 用默认的templates.cfg文件作为资源引用文件

a. templates.cfg文件

nagios主要用于监控主机资源以及服务，在nagios配置中称为对象，为了不必重复定义一些监控对象，Nagios引入了一个模板配置文件，将一些共性的属性定义成模板，以便于多次引用。这就是templates.cfg的作用。

下面详细介绍下templates.cfg文件中每个参数的含义：

复制代码

define contact{

name generic-contact ; 联系人名称

service\_notification\_period 24x7 ; 当服务出现异常时，发送通知的时间段，这个时间段"24x7"在timeperiods.cfg文件中定义

host\_notification\_period 24x7 ; 当主机出现异常时，发送通知的时间段，这个时间段"24x7"在timeperiods.cfg文件中定义

service\_notification\_options w,u,c,r ; 这个定义的是“通知可以被发出的情况”。w即warn，表示警告状态，u即unknown，表示不明状态;  
 ; c即criticle，表示紧急状态，r即recover，表示恢复状态;  
 ; 也就是在服务出现警告状态、未知状态、紧急状态和重新恢复状态时都发送通知给使用者。

host\_notification\_options d,u,r ; 定义主机在什么状态下需要发送通知给使用者，d即down，表示宕机状态;  
 ; u即unreachable，表示不可到达状态，r即recovery，表示重新恢复状态。

service\_notification\_commands notify-service-by-email ; 服务故障时，发送通知的方式，可以是邮件和短信，这里发送的方式是邮件;  
 ; 其中“notify-service-by-email”在commands.cfg文件中定义。

host\_notification\_commands notify-host-by-email ; 主机故障时，发送通知的方式，可以是邮件和短信，这里发送的方式是邮件;  
 ; 其中“notify-host-by-email”在commands.cfg文件中定义。

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL CONTACT, JUST A TEMPLATE!

}

define host{

name generic-host ; 主机名称，这里的主机名，并不是直接对应到真正机器的主机名;  
 ; 乃是对应到在主机配置文件里所设定的主机名。

notifications\_enabled 1 ; Host notifications are enabled

event\_handler\_enabled 1 ; Host event handler is enabled

flap\_detection\_enabled 1 ; Flap detection is enabled

failure\_prediction\_enabled 1 ; Failure prediction is enabled

process\_perf\_data 1 ; 其值可以为0或1，其作用为是否启用Nagios的数据输出功能;  
 ; 如果将此项赋值为1，那么Nagios就会将收集的数据写入某个文件中，以备提取。

retain\_status\_information 1 ; Retain status information across program restarts

retain\_nonstatus\_information 1 ; Retain non-status information across program restarts

notification\_period 24x7 ; 指定“发送通知”的时间段，也就是可以在什么时候发送通知给使用者。

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL HOST, JUST A TEMPLATE!

}

define host{

name linux-server ; 主机名称

use generic-host ; use表示引用，也就是将主机generic-host的所有属性引用到linux-server中来;  
 ; 在nagios配置中，很多情况下会用到引用。

check\_period 24x7 ; 这里的check\_period告诉nagios检查主机的时间段

check\_interval 5 ; nagios对主机的检查时间间隔，这里是5分钟。

retry\_interval 1 ; 重试检查时间间隔，单位是分钟。

max\_check\_attempts 10 ; nagios对主机的最大检查次数，也就是nagios在检查发现某主机异常时，并不马上判断为异常状况;  
 ; 而是多试几次，因为有可能只是一时网络太拥挤，或是一些其他原因，让主机受到了一点影响;  
 ; 这里的10就是最多试10次的意思。

check\_command check-host-alive ; 指定检查主机状态的命令，其中“check-host-alive”在commands.cfg文件中定义。

notification\_period 24x7 ; 主机故障时，发送通知的时间范围，其中“workhours”在timeperiods.cfg中进行了定义;  
 ; 下面会陆续讲到。

notification\_interval 10 ; 在主机出现异常后，故障一直没有解决，nagios再次对使用者发出通知的时间。单位是分钟;  
 ; 如果你觉得，所有的事件只需要一次通知就够了，可以把这里的选项设为0

notification\_options d,u,r ; 定义主机在什么状态下可以发送通知给使用者，d即down，表示宕机状态;  
 ; u即unreachable，表示不可到达状态;  
 ; r即recovery，表示重新恢复状态。

contact\_groups ts ; 指定联系人组，这个“admins”在contacts.cfg文件中定义。

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL HOST, JUST A TEMPLATE!

}

define host{

name windows-server ; The name of this host template

use generic-host ; Inherit default values from the generic-host template

check\_period 24x7 ; By default, Windows servers are monitored round the clock

check\_interval 5 ; Actively check the server every 5 minutes

retry\_interval 1 ; Schedule host check retries at 1 minute intervals

max\_check\_attempts 10 ; Check each server 10 times (max)

check\_command check-host-alive ; Default command to check if servers are "alive"

notification\_period 24x7 ; Send notification out at any time - day or night

notification\_interval 10 ; Resend notifications every 30 minutes

notification\_options d,r ; Only send notifications for specific host states

contact\_groups ts ; Notifications get sent to the admins by default

hostgroups windows-servers ; Host groups that Windows servers should be a member of

register 0 ; DONT REGISTER THIS - ITS JUST A TEMPLATE

}

define service{

name generic-service ; 定义一个服务名称

active\_checks\_enabled 1 ; Active service checks are enabled

passive\_checks\_enabled 1 ; Passive service checks are enabled/accepted

parallelize\_check 1 ; Active service checks should be parallelized;  
 ; (disabling this can lead to major performance problems)

obsess\_over\_service 1 ; We should obsess over this service (if necessary)

check\_freshness 0 ; Default is to NOT check service 'freshness'

notifications\_enabled 1 ; Service notifications are enabled

event\_handler\_enabled 1 ; Service event handler is enabled

flap\_detection\_enabled 1 ; Flap detection is enabled

failure\_prediction\_enabled 1 ; Failure prediction is enabled

process\_perf\_data 1 ; Process performance data

retain\_status\_information 1 ; Retain status information across program restarts

retain\_nonstatus\_information 1 ; Retain non-status information across program restarts

is\_volatile 0 ; The service is not volatile

check\_period 24x7 ; 这里的check\_period告诉nagios检查服务的时间段。

max\_check\_attempts 3 ; nagios对服务的最大检查次数。

normal\_check\_interval 5 ; 此选项是用来设置服务检查时间间隔，也就是说，nagios这一次检查和下一次检查之间所隔的时间;  
 ; 这里是5分钟。

retry\_check\_interval 2 ; 重试检查时间间隔，单位是分钟。

contact\_groups ts ; 指定联系人组

notification\_options w,u,c,r ; 这个定义的是“通知可以被发出的情况”。w即warn，表示警告状态;  
 ; u即unknown，表示不明状态;  
 ; c即criticle，表示紧急状态，r即recover，表示恢复状态;  
 ; 也就是在服务出现警告状态、未知状态、紧急状态和重新恢复后都发送通知给使用者。

notification\_interval 10 ; Re-notify about service problems every hour

notification\_period 24x7 ; 指定“发送通知”的时间段，也就是可以在什么时候发送通知给使用者。

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL SERVICE, JUST A TEMPLATE!

}

define service{

name local-service ; The name of this service template

use generic-service ; Inherit default values from the generic-service definition

max\_check\_attempts 4 ; Re-check the service up to 4 times in order to determine its final (hard) state

normal\_check\_interval 5 ; Check the service every 5 minutes under normal conditions

retry\_check\_interval 1 ; Re-check the service every minute until a hard state can be determined

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL SERVICE, JUST A TEMPLATE!

}

复制代码

b. resource.cfg文件

resource.cfg是nagios的变量定义文件，文件内容只有一行：

$USER1$=/usr/local/nagios/libexec

其中，变量$USER1$指定了安装nagios插件的路径，如果把插件安装在了其它路径，只需在这里进行修改即可。需要注意的是，变量必须先定义，然后才能在其它配置文件中进行引用。

c. commands.cfg文件

此文件默认是存在的，无需修改即可使用，当然如果有新的命令需要加入时，在此文件进行添加即可。

复制代码

#notify-host-by-email命令的定义   
define command{

command\_name notify-host-by-email #命令名称，即定义了一个主机异常时发送邮件的命令。

command\_line /usr/bin/printf "%b" "\*\*\*\*\* Nagios \*\*\*\*\*\n\nNotification Type: $NOTIFICATIONTYPE$\nHost: $HOSTNAME$\nState: $HOSTSTATE$\nAddress: $HOSTADDRESS$\nInfo: $HOSTOUTPUT$\n\nDate/Time: $LONGDATETIME$\n" | /bin/mail -s "\*\* $NOTIFICATIONTYPE$ Host Alert: $HOSTNAME$ is $HOSTSTATE$ \*\*" $CONTACTEMAIL$ #命令具体的执行方式。

}

#notify-service-by-email命令的定义   
define command{

command\_name notify-service-by-email #命令名称，即定义了一个服务异常时发送邮件的命令

command\_line /usr/bin/printf "%b" "\*\*\*\*\* Nagios \*\*\*\*\*\n\nNotification Type: $NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost: $HOSTALIAS$\nAddress: $HOSTADDRESS$\nState: $SERVICESTATE$\n\nDate/Time: $LONGDATETIME$\n\nAdditional Info:\n\n$SERVICEOUTPUT$\n" | /bin/mail -s "\*\* $NOTIFICATIONTYPE$ Service Alert: $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ \*\*" $CONTACTEMAIL$

}  
#check-host-alive命令的定义

define command{

command\_name check-host-alive #命令名称，用来检测主机状态。

command\_line $USER1$/check\_ping -H $HOSTADDRESS$ -w 3000.0,80% -c 5000.0,100% -p 5   
 # 这里的变量$USER1$在resource.cfg文件中进行定义，即$USER1$=/usr/local/nagios/libexec;  
 # 那么check\_ping的完整路径为/usr/local/nagios/libexec/check\_ping;  
 # “-w 3000.0,80%”中“-w”说明后面的一对值对应的是“WARNING”状态，“80%”是其临界值。  
 # “-c 5000.0,100%”中“-c”说明后面的一对值对应的是“CRITICAL”，“100%”是其临界值。  
 # “-p 1”说明每次探测发送一个包。  
 }

define command{

command\_name check\_local\_disk

command\_line $USER1$/check\_disk -w $ARG1$ -c $ARG2$ -p $ARG3$ #$ARG1$是指在调用这个命令的时候，命令后面的第一个参数。

}

define command{

command\_name check\_local\_load

command\_line $USER1$/check\_load -w $ARG1$ -c $ARG2$

}

define command{

command\_name check\_local\_procs

command\_line $USER1$/check\_procs -w $ARG1$ -c $ARG2$ -s $ARG3$

}

define command{

command\_name check\_local\_users

command\_line $USER1$/check\_users -w $ARG1$ -c $ARG2$

}

define command{

command\_name check\_local\_swap

command\_line $USER1$/check\_swap -w $ARG1$ -c $ARG2$

}

define command{

command\_name check\_ftp

command\_line $USER1$/check\_ftp -H $HOSTADDRESS$ $ARG1$

}

define command{

command\_name check\_http

command\_line $USER1$/check\_http -I $HOSTADDRESS$ $ARG1$

}

define command{

command\_name check\_ssh

command\_line $USER1$/check\_ssh $ARG1$ $HOSTADDRESS$

}

define command{

command\_name check\_ping

command\_line $USER1$/check\_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$ -p 5

}

define command{

command\_name check\_nt

command\_line $USER1$/check\_nt -H $HOSTADDRESS$ -p 12489 -v $ARG1$ $ARG2$

}

复制代码

d. hosts.cfg文件

此文件默认不存在，需要手动创建，hosts.cfg主要用来指定被监控的主机地址以及相关属性信息，根据实验目标配置如下：

复制代码

define host{

use linux-server #引用主机linux-server的属性信息，linux-server主机在templates.cfg文件中进行了定义。

host\_name Nagios-Linux #主机名

alias Nagios-Linux #主机别名

address 192.168.1.111 #被监控的主机地址，这个地址可以是ip，也可以是域名。

}

#定义一个主机组

define hostgroup{

hostgroup\_name bsmart-servers #主机组名称，可以随意指定。

alias bsmart servers #主机组别名

members Nagios-Linux #主机组成员，其中“Nagios-Linux”就是上面定义的主机。   
 }

复制代码

注意：在/usr/local/nagios/etc/objects 下默认有localhost.cfg 和windows.cfg 这两个配置文件，localhost.cfg 文件是定义监控主机本身的，windows.cfg 文件是定义windows 主机的，其中包括了对host 和相关services 的定义。所以在本次实验中，将直接在localhost.cfg 中定义监控主机（Nagios-Server），在windows.cfg中定义windows 主机（Nagios-Windows）。根据自己的需要修改其中的相关配置，详细如下：

localhost.cfg

复制代码

define host{

use linux-server ; Name of host template to use

; This host definition will inherit all variables that are defined

; in (or inherited by) the linux-server host template definition.

host\_name Nagios-Server

alias Nagios-Server

address 127.0.0.1

}

define hostgroup{

hostgroup\_name linux-servers ; The name of the hostgroup

alias Linux Servers ; Long name of the group

members Nagios-Server ; Comma separated list of hosts that belong to this group

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description PING

check\_command check\_ping!100.0,20%!500.0,60%

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description Root Partition

check\_command check\_local\_disk!20%!10%!/

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description Current Users

check\_command check\_local\_users!20!50

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description Total Processes

check\_command check\_local\_procs!250!400!RSZDT

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description Current Load

check\_command check\_local\_load!5.0,4.0,3.0!10.0,6.0,4.0

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description Swap Usage

check\_command check\_local\_swap!20!10

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description SSH

check\_command check\_ssh

notifications\_enabled 0

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description HTTP

check\_command check\_http

notifications\_enabled 0

}

复制代码

windows.cfg

复制代码

define host{

use windows-server ; Inherit default values from a template

host\_name Nagios-Windows ; The name we're giving to this host

alias My Windows Server ; A longer name associated with the host

address 192.168.1.113 ; IP address of the host

}

define hostgroup{

hostgroup\_name windows-servers ; The name of the hostgroup

alias Windows Servers ; Long name of the group

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description NSClient++ Version

check\_command check\_nt!CLIENTVERSION

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description Uptime

check\_command check\_nt!UPTIME

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description CPU Load

check\_command check\_nt!CPULOAD!-l 5,80,90

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description Memory Usage

check\_command check\_nt!MEMUSE!-w 80 -c 90

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description C:\ Drive Space

check\_command check\_nt!USEDDISKSPACE!-l c -w 80 -c 90

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description W3SVC

check\_command check\_nt!SERVICESTATE!-d SHOWALL -l W3SVC

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description Explorer

check\_command check\_nt!PROCSTATE!-d SHOWALL -l Explorer.exe

}

复制代码

e. services.cfg文件

此文件默认也不存在，需要手动创建，services.cfg文件主要用于定义监控的服务和主机资源，例如监控http服务、ftp服务、主机磁盘空间、主机系统负载等等。Nagios-Server 和Nagios-Windows 相关服务已在相应的配置文件中定义，所以这里只需要定义Nagios-Linux 相关服务即可，这里只定义一个检测是否存活的服务来验证配置文件的正确性，其他服务的定义将在后面讲到。

define service{

use local-service #引用local-service服务的属性值，local-service在templates.cfg文件中进行了定义。

host\_name Nagios-Linux #指定要监控哪个主机上的服务，“Nagios-Server”在hosts.cfg文件中进行了定义。

service\_description check-host-alive #对监控服务内容的描述，以供维护人员参考。

check\_command check-host-alive #指定检查的命令。  
 }

f. contacts.cfg文件

contacts.cfg是一个定义联系人和联系人组的配置文件，当监控的主机或者服务出现故障，nagios会通过指定的通知方式（邮件或者短信）将信息发给这里指定的联系人或者使用者。

复制代码

define contact{

contact\_name David #联系人的名称,这个地方不要有空格

use generic-contact #引用generic-contact的属性信息，其中“generic-contact”在templates.cfg文件中进行定义

alias Nagios Admin

email david.tang@bsmart.cn

}

define contactgroup{

contactgroup\_name ts #联系人组的名称,同样不能空格

alias Technical Support #联系人组描述

members David #联系人组成员，其中“david”就是上面定义的联系人，如果有多个联系人则以逗号相隔

}

复制代码

g. timeperiods.cfg文件

此文件只要用于定义监控的时间段，下面是一个配置好的实例：

复制代码

#下面是定义一个名为24x7的时间段，即监控所有时间段

define timeperiod{

timeperiod\_name 24x7 #时间段的名称,这个地方不要有空格

alias 24 Hours A Day, 7 Days A Week

sunday 00:00-24:00

monday 00:00-24:00

tuesday 00:00-24:00

wednesday 00:00-24:00

thursday 00:00-24:00

friday 00:00-24:00

saturday 00:00-24:00

}

#下面是定义一个名为workhours的时间段，即工作时间段。

define timeperiod{

timeperiod\_name workhours

alias Normal Work Hours

monday 09:00-17:00

tuesday 09:00-17:00

wednesday 09:00-17:00

thursday 09:00-17:00

friday 09:00-17:00

}

复制代码

h. cgi.cfg文件

此文件用来控制相关cgi脚本，如果想在nagios的web监控界面执行cgi脚本，例如重启nagios进程、关闭nagios通知、停止nagios主机检测等，这时就需要配置cgi.cfg文件了。  
由于nagios的web监控界面验证用户为david，所以只需在cgi.cfg文件中添加此用户的执行权限就可以了，需要修改的配置信息如下：

复制代码

default\_user\_name=david

authorized\_for\_system\_information=nagiosadmin,david

authorized\_for\_configuration\_information=nagiosadmin,david

authorized\_for\_system\_commands=david

authorized\_for\_all\_services=nagiosadmin,david

authorized\_for\_all\_hosts=nagiosadmin,david

authorized\_for\_all\_service\_commands=nagiosadmin,david

authorized\_for\_all\_host\_commands=nagiosadmin,david

复制代码

i. nagios.cfg文件

nagios.cfg默认的路径为/usr/local/nagios/etc/nagios.cfg，是nagios的核心配置文件，所有的对象配置文件都必须在这个文件中进行定义才能发挥其作用，这里只需将对象配置文件在Nagios.cfg文件中进行引用即可。

复制代码

log\_file=/usr/local/nagios/var/nagios.log # 定义nagios日志文件的路径

cfg\_file=/usr/local/nagios/etc/objects/commands.cfg # “cfg\_file”变量用来引用对象配置文件，如果有更多的对象配置文件，在这里依次添加即可。  
cfg\_file=/usr/local/nagios/etc/objects/contacts.cfg  
cfg\_file=/usr/local/nagios/etc/objects/hosts.cfg  
cfg\_file=/usr/local/nagios/etc/objects/services.cfg  
cfg\_file=/usr/local/nagios/etc/objects/timeperiods.cfg

cfg\_file=/usr/local/nagios/etc/objects/templates.cfg  
cfg\_file=/usr/local/nagios/etc/objects/localhost.cfg # 本机配置文件  
cfg\_file=/usr/local/nagios/etc/objects/windows.cfg # windows 主机配置文件

object\_cache\_file=/usr/local/nagios/var/objects.cache # 该变量用于指定一个“所有对象配置文件”的副本文件，或者叫对象缓冲文件

precached\_object\_file=/usr/local/nagios/var/objects.precache  
resource\_file=/usr/local/nagios/etc/resource.cfg # 该变量用于指定nagios资源文件的路径，可以在nagios.cfg中定义多个资源文件。

status\_file=/usr/local/nagios/var/status.dat # 该变量用于定义一个状态文件，此文件用于保存nagios的当前状态、注释和宕机信息等。

status\_update\_interval=10 # 该变量用于定义状态文件（即status.dat）的更新时间间隔，单位是秒，最小更新间隔是1秒。

nagios\_user=nagios # 该变量指定了Nagios进程使用哪个用户运行。  
nagios\_group=nagios # 该变量用于指定Nagios使用哪个用户组运行。

check\_external\_commands=1 # 该变量用于设置是否允许nagios在web监控界面运行cgi命令;  
 # 也就是是否允许nagios在web界面下执行重启nagios、停止主机/服务检查等操作;  
 # “1”为运行，“0”为不允许。

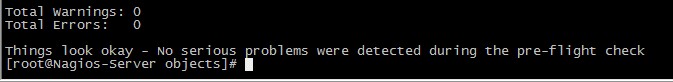
command\_check\_interval=10s # 该变量用于设置nagios对外部命令检测的时间间隔，如果指定了一个数字加一个"s"(如10s);  
 # 那么外部检测命令的间隔是这个数值以秒为单位的时间间隔;  
 # 如果没有用"s"，那么外部检测命令的间隔是以这个数值的“时间单位”的时间间隔。  
interval\_length=60 # 该变量指定了nagios的时间单位，默认值是60秒，也就是1分钟;  
 # 即在nagios配置中所有的时间单位都是分钟。

复制代码

6.4 验证Nagios 配置文件的正确性

Nagios 在验证配置文件方面做的非常到位，只需通过一个命令即可完成：

# /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg



Nagios提供的这个验证功能非常有用，在错误信息中通常会打印出错误的配置文件以及文件中的哪一行，这使得nagios的配置变得非常容易，报警信息通常是可以忽略的，因为一般那些只是建议性的。

看到上面这些信息就说明没问题了，然后启动Nagios 服务。

七、Nagios的启动与停止

7.1 启动Nagios

a. 通过初始化脚本启动nagios

# /etc/init.d/nagios start  
or  
# service nagios start

b. 手工方式启动nagios

通过nagios命令的“-d”参数来启动nagios守护进程：

# /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg

7.2 重启Nagios

当修改了配置文件让其生效时，需要重启/重载Nagios服务。

a. 通过初始化脚本来重启nagios

# /etc/init.d/nagios reload  
or  
# /etc/init.d/nagios restart  
or  
# service nagios restart

b. 通过web监控页重启nagios

可以通过web监控页的 "Process Info" -> "Restart the Nagios process"来重启nagios



c. 手工方式平滑重启

# kill -HUP <nagios\_pid>

7.3 停止Nagios

a. 通过初始化脚本关闭nagios服务

# /etc/init.d/nagios stop  
or  
# service nagios stop

b. 通过web监控页停止nagios

可以通过web监控页的 "Process Info" -> "Shutdown the Nagios process"来停止nagios



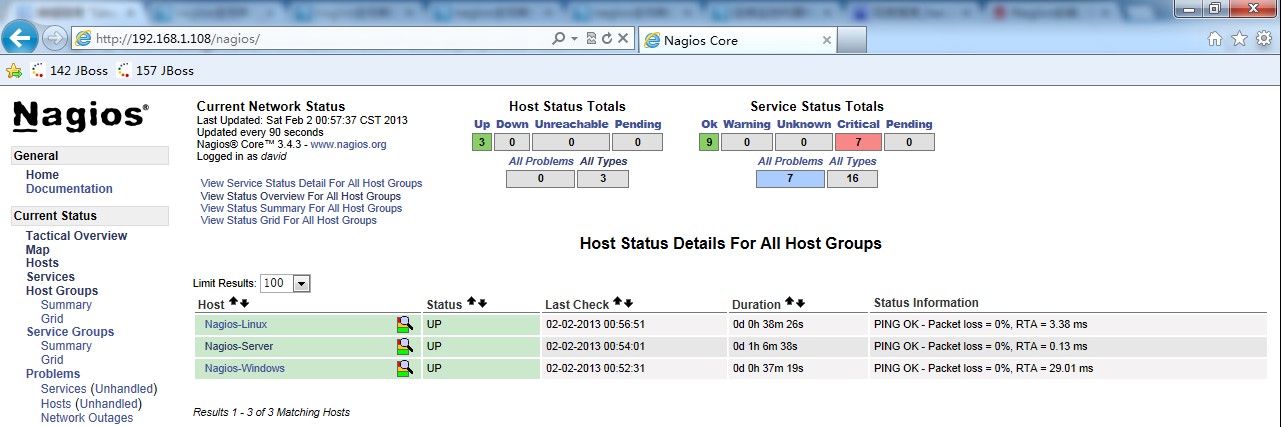
c. 手工方式停止Nagios

# kill <nagios\_pid>

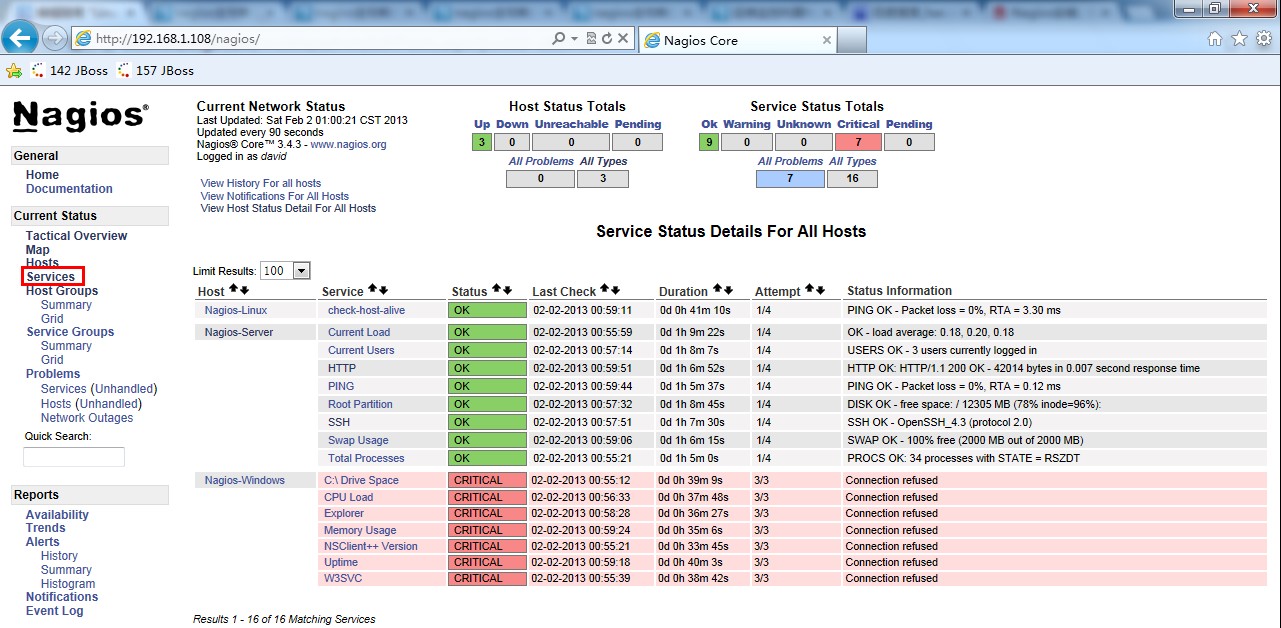
八、查看初步配置情况

8.1 启动完成之后，登录Nagios Web监控页[http://192.168.1.108/nagios/](http://172.16.1.124/nagios/) 查看相关信息。

8.2 点击左面的Current Status -> Hosts 可以看到所定义的三台主机已经全部UP了。



8.3 点击Current Status -> Services 查看服务监控情况。

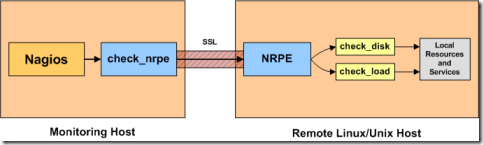


看到Nagios-Linux和Nagios-Server的服务状态已经OK了，但是Nagios-Windows的服务状态为CRITICAL，Status Information 提示Connection refused。因为Nagios-Windows上还未安装插件，内部服务还无法查看，所以出现这种情况。将在下面具体讲解。

九、利用NRPE监控远程Linux上的“本地信息”

上面已经对远程Linux 主机是否存活做了监控，而判断远程机器是否存活，我们可以使用ping 工具对其监测。还有一些远程主机服务，例如ftp、ssh、http，都是对外开放的服务，即使不用Nagios，我们也可以试的出来，随便找一台机器看能不能访问这些服务就行了。但是对于像磁盘容量，cpu负载这样的“本地信息”，Nagios只能监测自己所在的主机，而对其他的机器则显得有点无能为力。毕竟没得到被控主机的适当权限是不可能得到这些信息的。为了解决这个问题，nagios有这样一个附加组件--“NRPE”，用它就可以完成对Linux 类型主机"本地信息”的监控。

9.1 NRPE 工作原理

[](http://images.cnitblog.com/blog/370046/201301/30172018-ee257c664e12472590da4a3b503be33e.png)

NRPE 总共由两部分组成：

* check\_nrpe 插件，位于监控主机上
* NRPE daemon，运行在远程的Linux主机上(通常就是被监控机)

按照上图，整个的监控过程如下：

当Nagios 需要监控某个远程Linux 主机的服务或者资源情况时：

1. Nagios 会运行check\_nrpe 这个插件，告诉它要检查什么；
2. check\_nrpe 插件会连接到远程的NRPE daemon，所用的方式是SSL；
3. NRPE daemon 会运行相应的Nagios 插件来执行检查；
4. NRPE daemon 将检查的结果返回给check\_nrpe 插件，插件将其递交给nagios做处理。

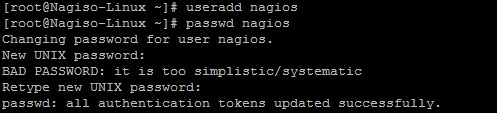
注意：NRPE daemon 需要Nagios 插件安装在远程的Linux主机上，否则，daemon不能做任何的监控。

9.2 在被监控机（Nagios-Linux）上

a. 增加用户&设定密码

# useradd nagios

# passwd nagios



b. 安装Nagios 插件

# tar zxvf nagios-plugins-1.4.16.tar.gz

# cd nagios-plugins-1.4.16

# ./configure --prefix=/usr/local/nagios

# make && make install

这一步完成后会在/usr/local/nagios/下生成三个目录include、libexec和share。

http://images.cnitblog.com/blog/370046/201302/02152814-8ad7b743e94d47e4a273d26f17fc039b.jpg

修改目录权限

# chown nagios.nagios /usr/local/nagios

# chown -R nagios.nagios /usr/local/nagios/libexec

http://images.cnitblog.com/blog/370046/201302/02152955-bba448f8c550421199e95f5abcd2a367.jpg

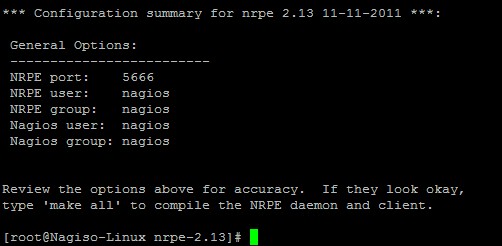
c. 安装NRPE

# wget http://prdownloads.sourceforge.net/sourceforge/nagios/nrpe-2.13.tar.gz

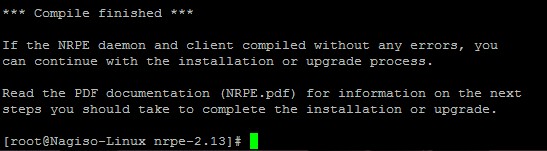
# tar zxvf nrpe-2.13.tar.gz

# cd nrpe-2.13

# ./configure



# make all



接下来安装NPRE插件，daemon和示例配置文件。

c.1 安装check\_nrpe 这个插件

# make install-plugin

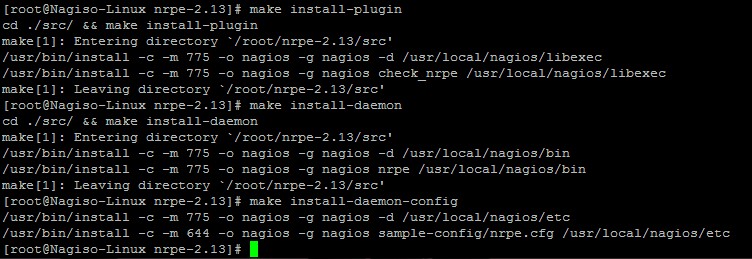
监控机需要安装check\_nrpe 这个插件，被监控机并不需要，我们在这里安装它只是为了[**测试**](http://lib.csdn.net/base/softwaretest)目的。

c.2 安装deamon

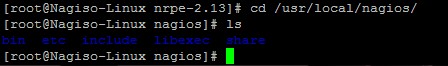
# make install-daemon

c.3 安装配置文件

# make install-daemon-config



现在再查看nagios 目录就会发现有5个目录了



按照安装文档的说明，是将NRPE deamon作为xinetd下的一个服务运行的。在这样的情况下xinetd就必须要先安装好，不过一般系统已经默认安装了。

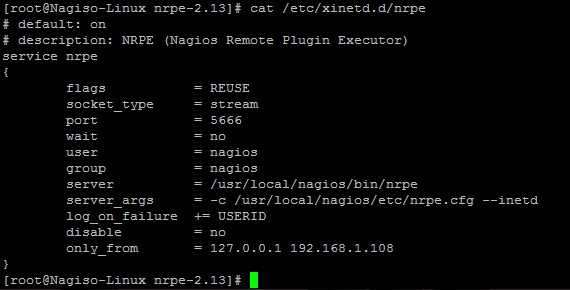
d. 安装xinted 脚本

# make install-xinetd

http://images.cnitblog.com/blog/370046/201302/02154307-bee4a49f8d87420a87899cd9c840c6d9.jpg

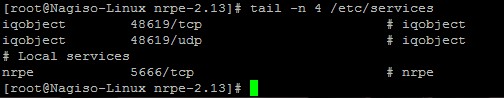
可以看到创建了这个文件/etc/xinetd.d/nrpe。

编辑这个脚本：



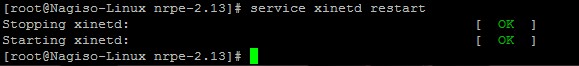
在only\_from 后增加监控主机的IP地址。

编辑/etc/services 文件，增加NRPE服务

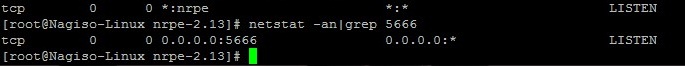


重启xinted 服务

# service xinetd restart



查看NRPE 是否已经启动



可以看到5666端口已经在监听了。

e. 测试NRPE是否则正常工作

使用上面在被监控机上安装的check\_nrpe 这个插件测试NRPE 是否工作正常。

# /usr/local/nagios/libexec/check\_nrpe -H localhost

会返回当前NRPE的版本

http://images.cnitblog.com/blog/370046/201302/02160042-6ae7c0101c264ff8b8076192ace36a23.jpg

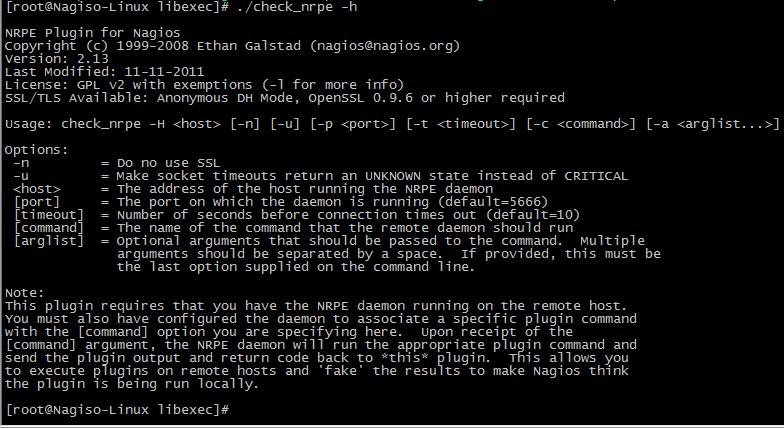
也就是在本地用check\_nrpe连接nrpe daemon是正常的。

注：为了后面工作的顺利进行，注意本地防火墙要打开5666能让外部的监控机访问。

f. check\_nrpe 命令用法

查看check\_nrpe 命令用法

# /usr/local/nagios/libexec/check\_nrpe –h



可以看到用法是：

check\_nrpe –H 被监控的主机 -c 要执行的监控命令

注意：-c 后面接的监控命令必须是nrpe.cfg 文件中定义的。也就是NRPE daemon只运行nrpe.cfg中所定义的命令。

g. 查看NRPE的监控命令

# cd /usr/local/nagios/etc

# cat nrpe.cfg |grep -v "^#"|grep -v "^$"

复制代码

[root@Nagiso-Linux etc]# cat nrpe.cfg |grep -v "^#"|grep -v "^$"

log\_facility=daemon

pid\_file=/var/run/nrpe.pid

server\_port=5666

nrpe\_user=nagios

nrpe\_group=nagios

allowed\_hosts=127.0.0.1

dont\_blame\_nrpe=0

debug=0

command\_timeout=60

connection\_timeout=300

command[check\_users]=/usr/local/nagios/libexec/check\_users -w 5 -c 10

command[check\_load]=/usr/local/nagios/libexec/check\_load -w 15,10,5 -c 30,25,20

command[check\_sda1]=/usr/local/nagios/libexec/check\_disk -w 20% -c 10% -p /dev/sda1

command[check\_zombie\_procs]=/usr/local/nagios/libexec/check\_procs -w 5 -c 10 -s Z

command[check\_total\_procs]=/usr/local/nagios/libexec/check\_procs -w 150 -c 200

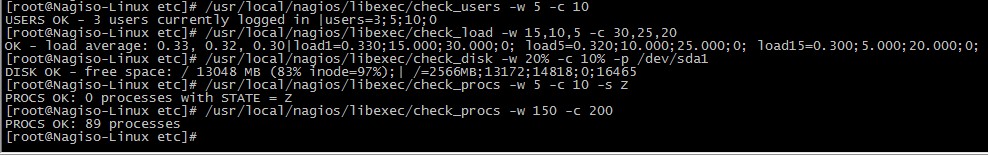
[root@Nagiso-Linux etc]#

复制代码

红色部分是命令名，也就是check\_nrpe 的-c 参数可以接的内容，等号 “=” 后面是实际执行的插件程序（这与commands.cfg 中定义命令的形式十分相似，只不过是写在了一行）。也就是说check\_users 就是等号后面/usr/local/nagios/libexec/check\_users -w 5 -c 10 的简称。

我们可以很容易知道上面这5行定义的命令分别是检测登陆用户数，cpu负载，sda1的容量，僵尸进程，总进程数。各条命令具体的含义见插件用法（执行“插件程序名 –h”）。

由于-c 后面只能接nrpe.cfg 中定义的命令，也就是说现在我们只能用上面定义的这五条命令。我们可以在本机实验一下。



9.3 在监控主机（Nagios-Server）上

之前已经将Nagios运行起来了，现在要做的事情是：

* 安装check\_nrpe 插件；
* 在commands.cfg 中创建check\_nrpe 的命令定义，因为只有在commands.cfg 中定义过的命令才能在services.cfg 中使用；
* 创建对被监控主机的监控项目；

9.3.1 安装check\_nrpe 插件

# tar zxvf nrpe-2.13.tar.gz

# cd nrpe-2.13

# ./configure

# make all

# make install-plugin

只运行这一步就行了，因为只需要check\_nrpe插件。

在Nagios-Linux 上我们已经装好了nrpe，现在我们测试一下监控机使用check\_nrpe 与被监控机运行的nrpe daemon之间的通信。

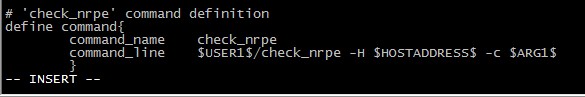
http://images.cnitblog.com/blog/370046/201302/02165718-6aa53e921dc2432a9738a5638aaa17e3.jpg

看到已经正确返回了NRPE的版本信息，说明一切正常。

9.3.2 在commands.cfg中增加对check\_nrpe的定义

# vi /usr/local/nagios/etc/objects/commands.cfg

在最后面增加如下内容：



意义如下：

# 'check\_nrpe' command definition

define command{

command\_name check\_nrpe # 定义命令名称为check\_nrpe,在services.cfg中要使用这个名称.

command\_line $USER1$/check\_nrpe -H $HOSTADDRESS$ -c $ARG1$ #这是定义实际运行的插件程序.  
 # 这个命令行的书写要完全按照check\_nrpe这个命令的用法,不知道用法的就用check\_nrpe –h查看.

}

-c 后面带的$ARG1$ 参数是传给nrpe daemon 执行的检测命令，之前说过了它必须是nrpe.cfg 中所定义的那5条命令中的其中一条。在services.cfg 中使用check\_nrpe 的时候要用 “!” 带上这个参数。

9.3.3 定义对Nagios-Linux 主机的监控

下面就可以在services.cfg 中定义对Nagios-Linux 主机的监控了。

复制代码

define service{

use local-service

host\_name Nagios-Linux

service\_description Current Load

check\_command check\_nrpe!check\_load

}

define service{

use local-service

host\_name Nagios-Linux

service\_description Check Disk sda1

check\_command check\_nrpe!check\_sda1

}

define service{

use local-service

host\_name Nagios-Linux

service\_description Total Processes

check\_command check\_nrpe!check\_total\_procs

}

define service{

use local-service

host\_name Nagios-Linux

service\_description Current Users

check\_command check\_nrpe!check\_users

}

define service{

use local-service

host\_name Nagios-Linux

service\_description Check Zombie Procs

check\_command check\_nrpe!check\_zombie\_procs

}

复制代码

还有一个任务是要监控Nagios-Linux 的swap 使用情况。但是在nrpe.cfg 中默认没有定义这个监控功能的命令。怎么办？手动在nrpe.cfg 中添加，也就是自定义NRPE命令。

现在我们要监控swap 分区，如果空闲空间小于20%则为警告状态 -> warning；如果小于10%则为严重状态 -> critical。我们可以查得需要使用check\_swap插件，完整的命令行应该是下面这样。

# /usr/local/nagios/libexec/check\_swap -w 20% -c 10%  
  
在被监控机（Nagios-Linux）上增加check\_swap 命令的定义

# vi /usr/local/nagios/etc/nrpe.cfg

增加下面这一行

command[check\_swap]=/usr/local/nagios/libexec/check\_swap -w 20% -c 10%

我们知道check\_swap 现在就可以作为check\_nrpe 的-c 的参数使用了

修改了配置文件，当然要重启。

如果你是以独立的daemon运行的nrpe，那么需要手动重启；如果你是在xinetd 下面运行的，则不需要。

由于本实验中nrpe 是xinetd 下运行的，所以不需要重启服务。  
  
在监控机（Nagios-Server）上增加这个check\_swap 监控项目

define service{

use local-service

host\_name Nagios-Linux

service\_description Check Swap

check\_command check\_nrpe!check\_swap

}

同理，Nagios-Linux 上我还开启了http 服务，需要监控一下，按照上面的做法，在被监控机（Nagios-Linux）上增加check\_http 命令的定义

# vi /usr/local/nagios/etc/nrpe.cfg

增加下面这一行

command[check\_http]=/usr/local/nagios/libexec/check\_http -I 127.0.0.1

在监控机（Nagios-Server）上增加check\_http 监控项目

define service{

use local-service

host\_name Nagios-Linux

service\_description HTTP

check\_command check\_nrpe!check\_http

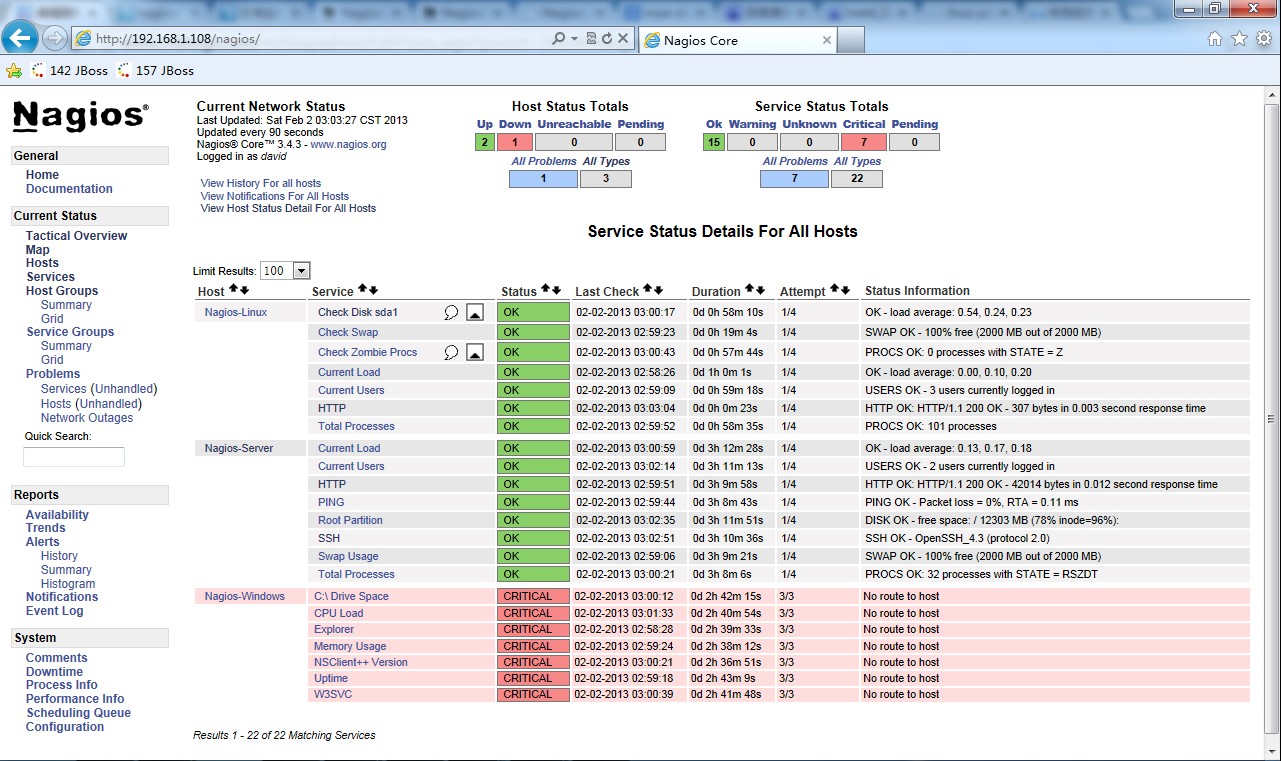
}

所有的配置文件已经修改好了，现在重启Nagios。

# service nagios restart

9.3.4 查看配置情况

登录Nagios Web监控页[http://192.168.1.108/nagios/](http://172.16.1.124/nagios/) 查看相关信息。

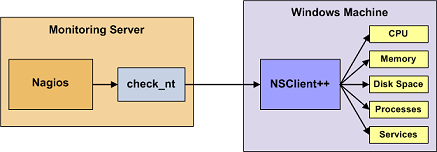


可以看到，对于Nagios-Server 和Nagios-Linux 上的相关服务的监控已经成功了，还有Nagios-Windows 上的服务还没有定义，下面讲到。

十、利用NSClient++监控远程Windows上的“本地信息”

在Nagios的libexec下有check\_nt这个插件，它就是用来检查windows机器的服务的。其功能类似于check\_nrpe。不过还需要搭配另外一个软件NSClient++，它则类似于NRPE。

NSClient++的原理如下图



可以看到NSClient与nrpe最大的区别就是：

* 被监控机上安装有nrpe，并且还有插件，最终的监控是由这些插件来进行的。当监控主机将监控请求发给nrpe后，nrpe调用插件来完成监控。
* NSClient++则不同，被监控机上只安装NSClient，没有任何的插件。当监控主机将监控请求发给NSClient++后，NSClient直接完成监控，所有的监控是由NSClient完成的。

这也说明了NSClient++的一个很大的问题：不灵活、没有可扩展性。它只能完成自己本身包含的监控操作，不能由一些插件来扩展。好在NSClient++已经做的不错了，基本上可以完全满足我们的监控需求。

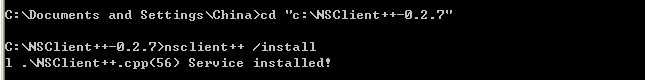
10.1 安装NSClient++

从http://www.nsclient.org/nscp/downloads 下载NSClient++-0.2.7.zip

解压到C盘根目录。

打开cmd 切换到c:\NSClient++-0.2.7

执行nsclient++ /install 进行安装



执行nsclient++ SysTray （注意大小写），这一步是安装系统托盘，时间稍微有点长。

http://images.cnitblog.com/blog/370046/201302/02222739-b0076ad25fbe40d99ff7094f43418657.jpg

在运行里面输入services.msc 打开“服务”



看到下图就说明NSClient服务已经安装上了

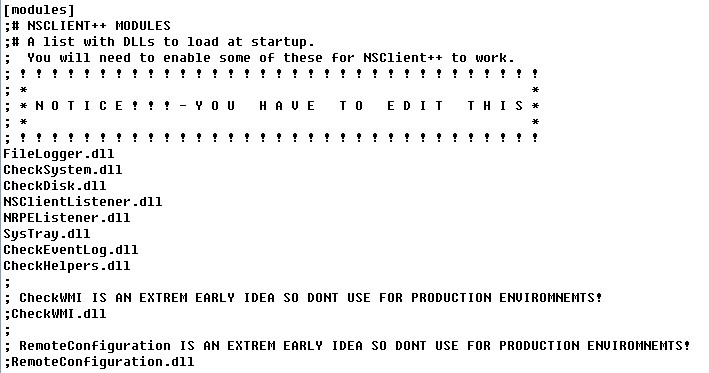
http://images.cnitblog.com/blog/370046/201302/02222842-718e9a7851a348809b4d00244e369544.jpg

双击打开，点“登录”标签，在“允许服务与桌面交互”前打勾。



编辑c:\NSClient++-0.2.7下的NSC.ini文件。

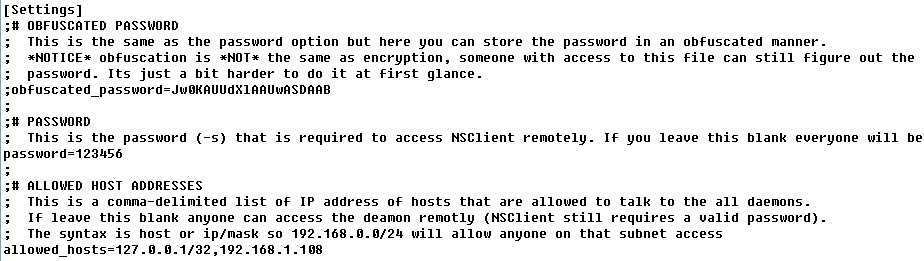
将 [modules]部分的所有模块前面的注释都去掉，除了CheckWMI.dll 和 RemoteConfiguration.dll 这两个。



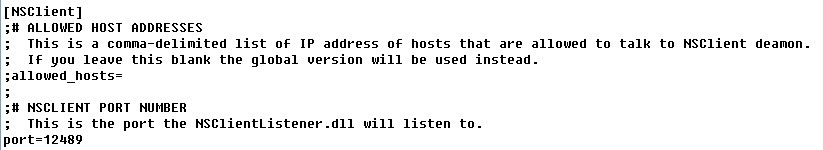
在[Settings]部分设置'password'选项来设置密码，作用是在nagios连接过来时要求提供密码。这一步是可选的，我这里设置为'123456'。

将[Settings]部分'allowed\_hosts'选项的注释去掉，并且加上运行nagios的监控主机的IP。各IP之间以逗号相隔。这个地方是支持子网的，如果写成192.168.1.0/24则表示该子网内的所有机器都可以访问。如果这个地方是空白则表示所有的主机都可以连接上来。

注意是[Settings]部分的，因为[NSClient]部分也有这个选项。



必须保证[NSClient]的'port'选项并没有被注释，并且它的值是'12489'，这是NSClient的默认监听端口。



在cmd 中执行nsclient++ /start启动服务，注意所在目录是c:\NSClient++-0.2.7

http://images.cnitblog.com/blog/370046/201302/02223713-a55666da26bb47338db68d06d04db0ff.jpg

这时在桌面右下角的系统托盘处会出现一个黄色的M字样的图标

http://images.cnitblog.com/blog/370046/201302/02223741-eea5809e0c0a4a75920308e0530a679e.jpg

查看服务

http://images.cnitblog.com/blog/370046/201302/02223800-aa5a5cf0f907417e804e1bf42e3ad95c.jpg

已经正常启动了。

注意服务默认设的是“自动”，也就是说是开机自动启动的。

在cmd 里面执行netstat –an 可以看到已经开始监听tcp的12489端口了。

http://images.cnitblog.com/blog/370046/201302/02223852-f70c360e03204553a28086040839174d.jpg

这样外部就可以访问了吗？

错！

防火墙也要打开tcp的12489端口，否则nagios 检查此服务的时候会报错。

这样被监控机的配置就搞定了，它就等待nagios 发出某个监控请求，然后它执行请求将监控的结果发回到nagios监控主机上。

之前已经在监控主机（Nagios-Server）上对Windows 主机的监控做了配置，但是commands.cfg 中默认没有设置密码项，所以要修改一下，增加"-s 123456"，如下：

# 'check\_nt' command definition

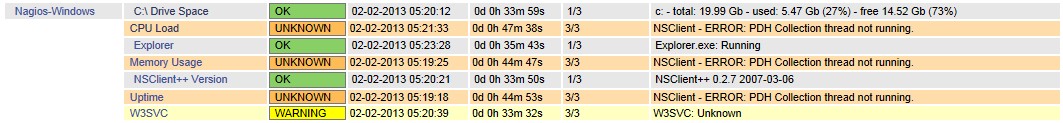
define command{

command\_name check\_nt

command\_line $USER1$/check\_nt -H $HOSTADDRESS$ -p 12489 -s 123456 -v $ARG1$ $ARG2$

}

现在打开Nagios Web监控页便可查看到相关信息了。



可以看到有错误：NSClient - ERROR: PDH Collection thread not running.

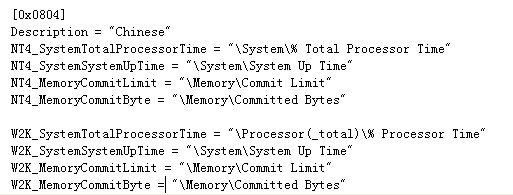
Google 一下，是由于[**操作系统**](http://lib.csdn.net/base/operatingsystem)语言的问题，好像NSClient 默认支持的语言并不多，具体可以百度一下。

查看NSClient的日志C:\NSClient++-0.2.7\nsclient.log，信息如下：

2013-02-02 22:05:30: error:.\PDHCollector.cpp:98: You need to manually configure performance counters!

需要手动配置performance counters。

打开C:\NSClient++-0.2.7\counters.defs文件，复制文件里面"English US"那部分内容，粘贴到counters.defs 文件的最后，修改Description = "Chinese"。



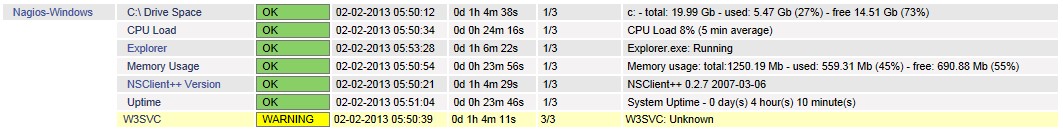
修改完之后，在mmc中重启NSClient 服务。

然后查看日志，内容如下：

http://images.cnitblog.com/blog/370046/201302/03004023-10d2dacf5c6a4d57bbb1c723972a67bd.jpg

在正常执行了。

打开Nagios Web监控页查看。

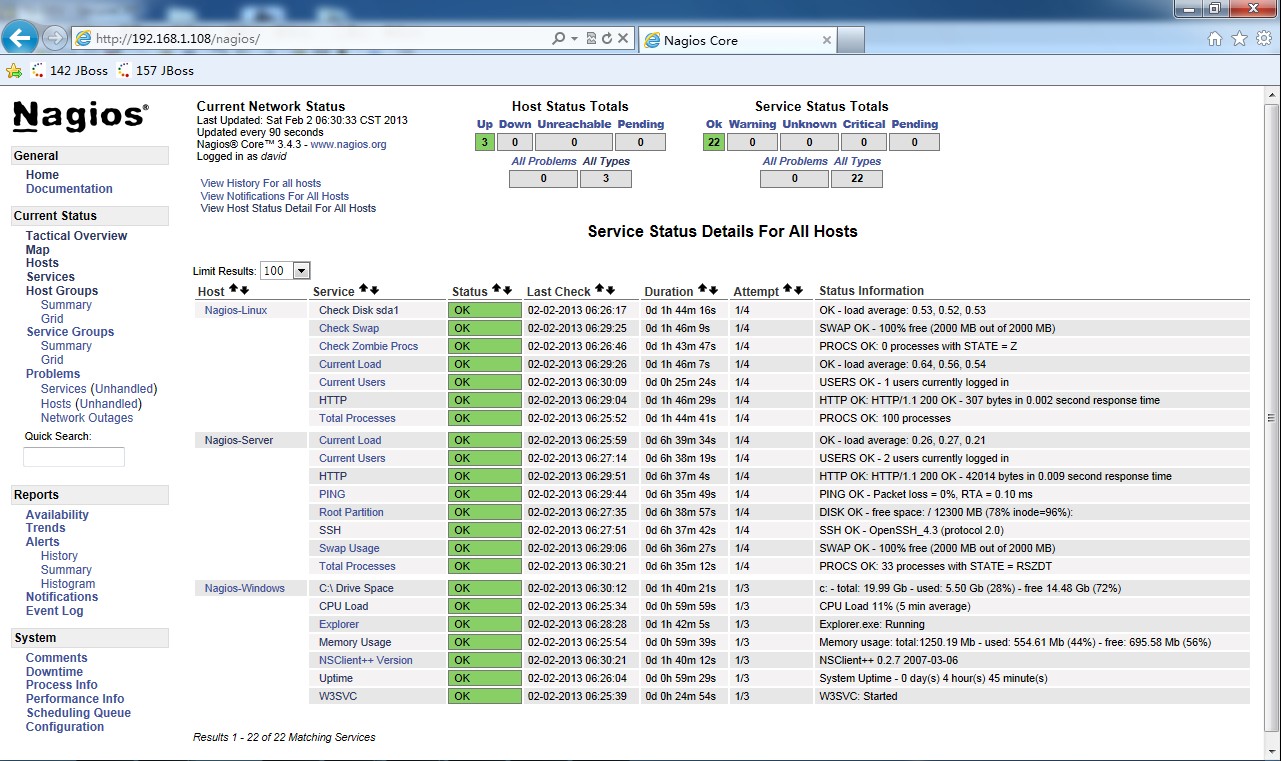


执行成功，但是W3SVC服务为Unknown 状态。查资料，需要开启Windows 的IIS服务。

打开“控制面板”进行安装。



安装完毕后，再到Nagios Web监控页查看，全部监控正常。



十一、Nagios邮件报警的配置

11.1 安装sendmail 组件

首先要确保sendmail 相关组件的完整安装，我们可以使用如下的命令来完成sendmail 的安装：

# yum install -y sendmail\*

然后重新启动sendmail服务：

# service sendmail restart

然后发送测试邮件，验证sendmail的可用性：

# echo "Hello World" | mail david.tang@bsmart.cn

11.2 邮件报警的配置

在上面我们已经简单配置过了/usr/local/nagios/etc/objects/contacts.cfg 文件，Nagios 会将报警邮件发送到配置文件里的E-mail 地址。

11.3 Nagios 通知

PROBLEM



RECOVERY



Linux下Nagios安装配置完毕。

参考资料

* Nagios官方网站：<http://www.nagios.org/>
* yahoon的小屋 《nagios全攻略》：http://yahoon.blog.51cto.com/
* 技术成就梦想 《运维监控利器Nagios》：http://ixdba.blog.51cto.com/

David Camp

* 技术交流，请加QQ群：

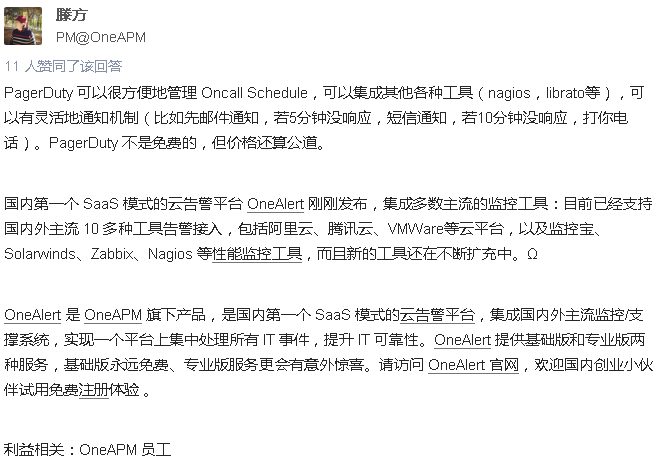
　　　　系统运维技术分享：296513821

* 业务合作，请联系作者QQ：562866602
* 我的微信号：mchina\_tang
* 给我写信：mchina\_tang@qq.com
* 我的地址：江苏·苏州

我们永远相信，分享是一种美德 | We Believe, Great People Share Knowledge...

## Pagerduty有详细的中文介绍吗？ 国内有类似Pagerduty的通告服务吗？

- 知乎 https://www.zhihu.com/question/32084832#answer-20485955



## 群聊IM工具HipChat开始向五人及以下的小团队免费提供服务\_36氪 http://36kr.com/p/202091.html

HipChat是一款非常赞的应用，专为团队内部群聊IM设计。你可以为单个项目或者小组搭建自有的聊天室，也可以很方便的发起一对一聊天。这套IM系统还整合了团队文件管理和分享，拖拽就能完成保存操作。HipChat基本支持所有平台，桌面端到移动端的都有。

[HipChat](https://www.hipchat.com/)是一款非常赞的应用，专为团队内部群聊IM设计。你可以为单个项目或者小组搭建自有的聊天室，也可以很方便的发起一对一聊天。这套IM系统还整合了团队文件管理和分享，拖拽就能完成保存操作。HipChat基本支持所有平台，桌面端到移动端的都有。

今天它在官网上专门写了篇文章，告诉大家五人及以下的小团队可以免费使用HipChat了。如果你之前就已经在用了而且团队人数不超过五个，那么不用再付费了；如果还没有，等什么呢孩子，去试试吧，你不会失望了。对了，HipChat刚刚推出了全新的Mac客户端，Android版也有一些更新。