# 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.

A single Prometheus server can easily handle millions of time series. That's enough for a thousand servers with a thousand time series each scraped every 10 seconds. As your systems scale beyond that, Prometheus can scale too.

Initial Deployment

A single Prometheus server per datacenter or similar failure domain (e.g. EC2 region) can typically handle a thousand servers, so should last us for a good while. Running one per datacenter avoids having the internet or WAN links on the critical path of your monitoring.

If we have more than one datacenter, we may wish to have global aggregates of some time series. This is done with a "global Prometheus" server, which federates from the datacenter Prometheus servers.

It's suggested to run two global Prometheus in different datacenters. This keeps our global monitoring working even if one datacenter has an outage.

**- scrape\_config:**

**- job\_name: dc\_prometheus**

**honor\_labels: true**

**metrics\_path: /federate**

**params:**

**match[]:**

**- '{\_\_name\_\_=~"^job:.\*"}' # Request all job-level time series**

**static\_configs:**

**- targets:**

**- dc1-prometheus:9090**

**- dc2-prometheus:9090**

Splitting By Use

As we grow we'll come to a point where a single Prometheus isn't quite enough. The next step is to run multiple Prometheus servers per datacenter. Each one will own monitoring for some team or slice of the stack. A first pass may result in fronted, backend and machines (node exporter) for example.

As we continue to grow, this process can be repeated. MySQL and Cassandra monitoring may end up with their own Prometheus, or each Cassandra cluster may have a Prometheus server dedicated to it.

We may also wish to start splitting by use before there are performance issues, as teams may not want to share Prometheus or to improve isolation.

## Prometheus Hierarchy

When we can't subdivide Prometheus servers any longer, the final step in scaling is to scale out. This usually requires that a single job has thousands of instances, a scale that most users never reach. This is more complex setup and is much more involved to manage than a normal Prometheus deployment, so should be avoided for as long as we can.

The architecture is to have multiple slave Prometheus, each scraping a subset of the targets and aggregating them up within the slave. A master federates the aggregates produced by the slaves, and then the master aggregates them up to the job level.

On the slaves we can use a hash of the address to select only some targets to scrape:

**global:**

**external\_labels:**

**slave: 1 # This is the 2nd slave. This prevents clashes between slaves.**

**scrape\_configs:**

**- job\_name: some\_job**

**# Add usual service discovery here, such as static\_configs**

**relabel\_configs:**

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

**modulus: 4 # 4 slaves**

**target\_label: \_\_tmp\_hash**

**action: hashmod**

**- source\_labels: [\_\_tmp\_hash]**

**regex: ^1$ # This is the 2nd slave**

**action: keep**

And the master federates from the slaves:

**- scrape\_config:**

**- job\_name: slaves**

**honor\_labels: true**

**metrics\_path: /federate**

**params:**

**match[]:**

**- '{\_\_name\_\_=~"^slave:.\*"}' # Request all slave-level time series**

**static\_configs:**

**- targets:**

**- slave0:9090**

**- slave1:9090**

**- slave3:9090**

**- slave4:9090**

Information for dashboards is usually taken from the master. If we wanted to drill down to a particular target, we'd do so via its slave.

### Example:

For example say we wanted to aggregate up the total amount of memory on all our machines at a global level.

First there's part of the config file for each datacenter Prometheus:

**global:**

**external\_labels:**

**datacenter: eu-west-1**

**rule\_files:**

**- node.rules**

**scrape\_configs:**

**etc.**

What we need to do here is to specify unique ***external\_labels*** on each of the datacenter Prometheus servers. This will cause them to add those labels on the ***/federate*** endpoint, and prevent the clashing time series we're running into.

Then in the **node.rules** rules file we aggregate up to the datacenter level:

**job:node\_memory\_MemTotal:sum = sum without(instance)(node\_memory\_MemTotal{job="node"})**

As only the job label will be left on the time series it gets a **job:** prefix, and we're summing so it's a **:sum** suffix.

In the global Prometheus config we pull in this timeseries:

**global:**

**external\_labels:**

**datacenter: global**

**scrape\_configs:**

**- job\_name: datacenter\_federation**

**honor\_labels: true**

**metrics\_path: /federate**

**params:**

**match[]:**

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

**static\_configs:**

**- targets:**

**- eu-west-1-prometheus:9090**

The **match[]** here requests all job-level time series, so by following this naming convention we don't have to adjust the config every time there's a new aggregating rule.

Now we can use the below expression to get the memory available across our entire global fleet!

**sum without(datacenter)(job:node\_memory\_MemTotal:sum)**

## Where Federation Doesn't Fit

In the above case, federation is being used to pull in a limited and aggregated set of time series from another Prometheus. That Prometheus is otherwise continuing to do it's duty, firing alerts and serving graph queries.

Where federation isn't suitable is if we use it to pull large swatches of time series - or even all time series - from another Prometheus, and then do alerting and graphing only from there. There's three broad reasons for this.

The first is performance and scaling. As the limiting factor of Prometheus performance is how much can be handled by a single machine, routing all your data to one global Prometheus limits your monitoring to what one machine can handle. By instead pulling only aggregated time series we're only limited to what one datacenter Prometheus can handle, allowing we to add new datacenters without having to worry about scaling the global Prometheus. The federation request itself can also be heavy to serve for the receiving Prometheus.

The second is reliability. If the data we need to do alerting is moved from one Prometheus to another then we've added an additional point of failure. This is particularly risky when WAN links such as the internet are involved. As far as is possible, we should try and push alerting as deep down the federation hierarchy as possible. For example an alert about a target being down should be setup on the Prometheus scraping that target, not a global Prometheus which could be several steps removed.

The third is correctness. Due to how it works, federation will pull in data some time after it was scraped and may also miss some data due to races. While the artifacts and oddness this causes in a global Prometheus are generally tolerable in a setup where your datacenter Prometheus servers are the primary workhorses for graphing and alerting, this is much more likely to cause issues when we're federating everything.

These issues don't just apply between datacenter and global Prometheus servers. Some have attempted to use a Prometheus as a type of proxy server, using federation to pull all data out of the scraping Prometheus into another Prometheus where all the real work is done. There are issues with this, such as the above mentioned correctness problems. If we find yourself in this situation either make the scraping Prometheus handle alerting and graphing, or do the scrapes via an actual proxy server using proxy\_url.