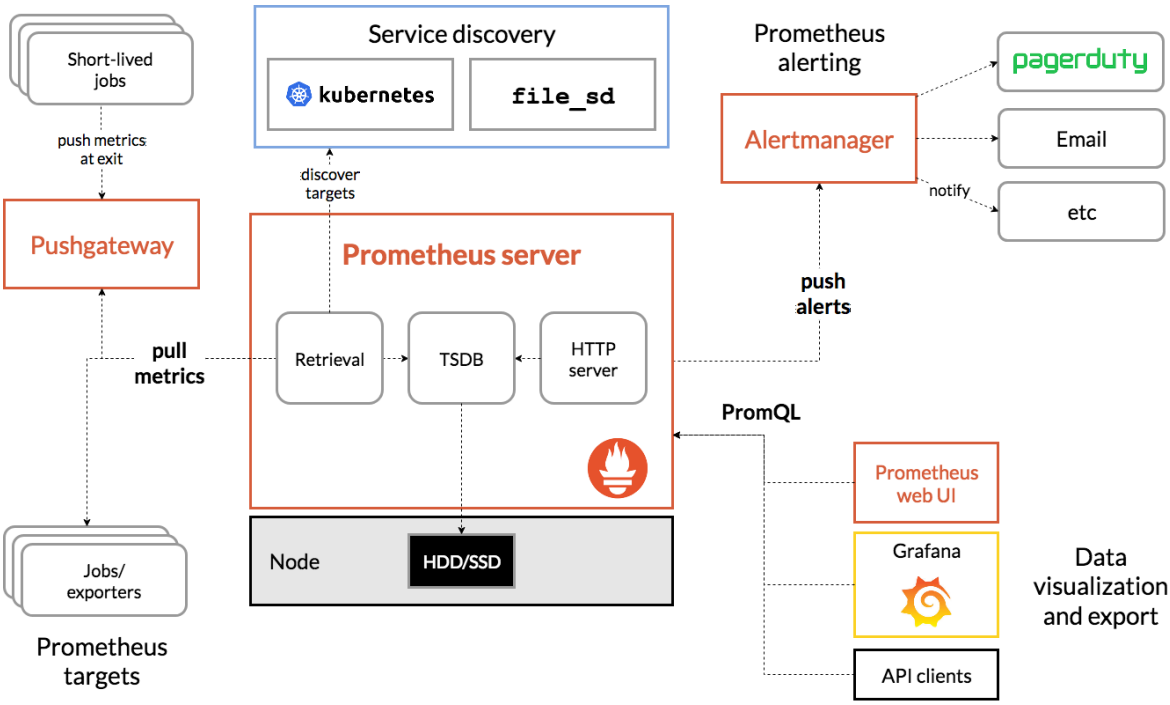
这篇文章介绍了一下 Prometheus 的架构，Prometheus 的整体架构也是一种 server/agent 架构，其中 server 是 Prometheus Server，agent 是数据采集 agent。架构图如下所示。



这里从数据流的链路来简单介绍一下整体的架构：

- 首先指标数据通过 exporter 采集，或者存活时间短的 job 将指标数据发送给 pushgateway。
- Prometheus Server 会定时从 exporter 或者 Pushgateway 中拉取指标数据，并存储到合适的存储介质里面，比如时间序列数据库等。
- 存储下来的指标数据可以外部对接监控展示的 Dashboard，比如 Prometheus 的原生 UI，或者 Grafana，甚至 api client。
- Prometheus Server 对于收集上来的数据还可以通过设置检测规则，比如 CPU 使用率连续 5 个小时超过 95% 则生成告警事件，然后将告警事件发送给用来专门处理告警的 Alertmanager。
- Alertmanager 对接收到的告警做一些处理之后发送到外部的通知系统，比如邮件等。

下面将介绍如下几个主要组件。

- Prometheus Server；
- Exporter；
- Pushgateway；
- AlertManager；
- Grafana。

1. Prometheus Server

Prometheus Server 是整体架构的核心了，负责指标的收集、存储、查询以及告警。

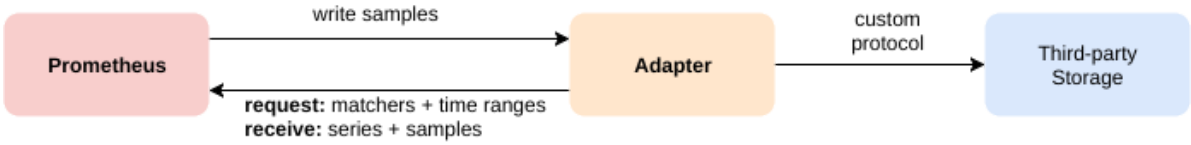
收集

这里说的是指标的收集，没有说采集。准确来说指标的采集是通过 Exporter 去采集的，比如我们要采集机器的指标（CPU、内存等），那么可以将 Exporter 部署到机器上。然后 Prometheus Server 通过 Pull 的方式和 Exporter 进行交互，将数据收集到 Prometheus Server。

存储

Prometheus Server 将数据收集来之后会进行存储，可以使用多种存储引擎来进行数据存储。比如本地磁盘、时间序列数据库等。

值得一提的是，Prometheus 还支持远端数据存储。支持是以一种协议的形式，包括 remote write 和 remote read，分别对应写和读。



Prometheus 支持常见的存储引擎的集成，比如 InfluxDB，OpenTSDB 等。全量列表可以参考 链接。

查询

Prometheus 提供了一种 DSL 叫 PromSQL 用来对存储的数据进行查询。

2. Exporter

---

exporter 简单来说就是指标采集 agent 程序。exporter 一般是部署在目标机器上，或者和监控目标可以网络连通的地方，这样就可以采集目标对象的指标数据。然后我们将 exporter 的信息暴露给 Prometheus Server，Prometheus Server 会根据配置中的固定时间间隔去 exporter 拉取数据即可。

目前在 Prometheus 的官网的 Download 的 Tab 页面 里面提供了多种 exporter，包括：

- blackbox_exporter；
- consul_exporter；
- graphite_exporter；
- haproxy_exporter；
- memcached_exporter；
- mysqld_exporter；
- node_exporter。

下面我们以 node_exporter 为例简单来一下 exporter 的工作机制。首先下载自己机器对应的 node_exporter，darwin 对应 Mac OS 系统，linux 对应 Linux 系统，我们这里以 Linux 版本的 node_exporter 为例。

node_exporter

Exporter for machine metrics prometheus/node_exporter

1.0.0 / 2020-05-25 Release notes

File name	OS	Arch	Size	SHA256 Checksum
node_exporter-1.0.0.darwin-amd64.tar.gz	darwin	amd64	5.06 MiB	68eec397b0b88767508aab9ec5214070b5877daef33fb94b13
node_exporter-1.0.0.linux-amd64.tar.gz	linux	amd64	9.04 MiB	fae88be0aa33c8ae22340723f9b4a4e519e67d2969da7a2775

下载之后解压完成会发现三个文件

```
[root@xxx node_exporter-1.0.0.linux-amd64]
LICENSE  node_exporter  NOTICE
```

其中 node_exporter 就是可以运行的二进制文件。node_exporter 提供了若干启动参数，我们这里使用最简单的启动方式。

```
[root@xxx node_exporter-1.0.0.linux-amd64]
level=info ts=2020-06-10T15:32:22.465Z caller=node_exporter.go:177 msg="Starting
node_exporter" version="(version=1.0.0, branch=HEAD,
revision=b9c96706a7425383902b6143d097cf6d7cfd1960)"
level=info ts=2020-06-10T15:32:22.465Z caller=node_exporter.go:178 msg="Build
context" build_context="(go=go1.14.3, user=root@3e55cc20ccc0, date=20200526-
06:01:48)"
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:105 msg="Enabled
collectors"
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:112 collector=arp
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:112
collector=bcache
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:112
collector=bonding
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:112
collector=btrfs
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:112
collector=conntrack
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:112 collector=cpu
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:112
collector=cpufreq
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:112
collector=diskstats
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:112
collector=edac
...
level=info ts=2020-06-10T15:32:22.466Z caller=node_exporter.go:191 msg="Listening
on" address=:9100
level=info ts=2020-06-10T15:32:22.466Z caller=tls_config.go:170 msg="TLS is
disabled and it cannot be enabled on the fly." http2=false
```

当看到最下面的日志 “Listening on” 即表示启动成功了。指标数据对应的请求 path 为 `/metrics`

```
[root@emr-header-1 node_exporter-1.0.0.linux-amd64]


go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0


go_goroutines 8


go_info{version="go1.14.3"} 1
```

```
go_memstats_alloc_bytes 2.574184e+06


go_memstats_alloc_bytes_total 2.574184e+06
```

3. Pushgateway

上一小节介绍了我们可以通过各种各样的 exporter 来采集指标数据，但是 exporter 和 Prometheus Server 的交互方式是一种拉的方式，这样就会有一个问题，因为 Prometheus Server 拉数据是按固定时间间隔来拉的，有可能我们的监控目标运行时间不超过一个时间间隔就结束了。这种情况在大数据场景中比较常见，比如很多批处理作业可能只会运行很多的时间，比如 10s，而 Prometheus 的定时采集间隔设置为 15s 就会导致作业的指标数据没有被采集到。

对于上面这种场景，一种可行的解决方案是批处理作业自己将指标数据推出来，而不是被动地等 Prometheus Server 来拉。那么数据应该推给谁呢？这个时候 Pushgateway 就出场了，Pushgateway 最核心的功能就是提供一个 POST 接口用来接收指标数据。下面简单演示一下。

首先下载解压并启动。

```
[root@xxx pushgateway-1.1.0]
LICENSE  NOTICE  pushgateway
[root@xxx pushgateway-1.1.0]
level=info ts=2020-06-10T16:12:06.011Z caller=main.go:82 msg="starting
pushgateway" version="(version=1.1.0, branch=HEAD,
revision=83950fb69e17ebf21041f7e8dbb118f3d130c9cd)"
level=info ts=2020-06-10T16:12:06.011Z caller=main.go:83 build_context="
(go=go1.13.6, user=root@ec0ab3e0fbb8, date=20200127-18:10:26)"
level=info ts=2020-06-10T16:12:06.013Z caller=main.go:143 listen_address=:9091
```

下面使用 POST 方法向 Pushgateway 发送指标数据，下面的 url 表示我们发送的这个指标具有 label 值：job=some_job。

查询指标数据。

```
[root@xxx ]


go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
...

some_metric{instance="",job="some_job"} 3.14
```

我们可以看到最后一行就是我们刚才方式的指标，除了我们 url 中指定的 label 还自带了一个 instance label。这个肯定有同学会问了，除了 instance 和 job 这两个 label，我们还可以指定其他的 label 吗？当然是可以的，我们可以通过如下两种方式来指定，一种是 url 中拼接，一种是在指标中说明。

这个时候我们在进行查询，我们可以看到上面发送的两条指标数据都可以查询到。

```
[root@xxx]

some_metric{instance="",job="some_job",} 3.14
some_metric{id="123",instance="",job="some_job"} 3.14
```

4. Alertmanager

作为一个合格的监控系统，怎么能缺少告警组件呢？Alertmanager 就是 Prometheus 版图中的告警组件，它会接收来自 Prometheus Server 或者其他应用程序发过来的告警，然后做一些处理，比如去重、分组、抑制，然后将这些告警信息发送到外部接收系统，比如 email 等。我们下面简单看看 Alertmanager 提供的这些处理操作，对于相关从业者还是很有启发意义的。

分组（Grouping）

分组是将具有相同性质的告警进行合并。可能这么听起来比较抽象，我们举一个具体的例子。

比如我们有一个几百台机器规模的集群，每台机器上运行一个进程，这些进程会去访问数据库。这个时候，集群发生了网络故障导致一半的机器连不上数据。同时我们对每一台机器上面的服务配置了告警规则：如果连不上数据库就告警。这个时候如果不做处理就会有几百条告警信息发送到用户或者管理员，作为告警的接收者这绝对不是一个好的体验。这几百条告警信息都是说一件事情，我们更希望接收到的告警是一条类似这样的告警信息：这些主机 {1,2,3...100} 连不上数据库了。这就是分组，这个示例中的主机就是分组的维度。

抑制（Inhibition）

抑制也是告警系统中一个常见的功能，简单来说就是当两个有关联关系的告警产生的时候，有时候我们可以抑制其中一个告警。还是拿上面的集群为例。

现在有一种告警是集群的机器宕机了，当这种告警产生的时候，那么其他的告警，比如服务不可用，则都可以抑制掉。

静默（Silences）

静默也是告警系统中的功能，比如不是很重要的告警如果持续发送，接收人有时候会比较难受，最重要的是可能会导致遗漏重要的信息。通过 Alertmanager 我们可以将特定模式的告警静默掉。

外部告警通知系统对接

Alertmanager 不支持原生的通知的机制，比如短信等，但是可以通过 webhook 的方式对接到外部系统。支持列表如下，参考链接。

5. Grafana

对于监控系统来说，指标展示 Dashboard 必然是不可或缺的一环，尽管 Prometheus 提供了一个页面用来查看指标，但是实在是太简陋了。现在社区使用最多的指标展示 Dashboard 是 Grafana，虽然严格来说 Grafana 并不算 Prometheus 系统里面的，但是现在基本所有人都将这两个关联在一起介绍，所以我们这里也来简单介绍了一下 Grafana。

Grafana 是一款使用 Go 语言开发的开源数据可视化工具，不仅可以用来说监控指标展示，还可以用来做数据统计，甚至报表。Grafana 也是具有告警功能的，但是一般直接使用 Grafana 的告警功能的公司并不多。Grafana 提供了一个在线的 Play Ground 可以供大家来直观的体验一下 Grafana 的酷炫。

6. 总结

本篇文章带领大家一起看了一下 Prometheus 的几个核心组件，希望大家在后面的使用或者公司技术选型的时候可以做到有的放矢。

}