Life of a Label

Brian Brazil
Founder

Who am I?

Engineer passionate about running software reliably in production.

- Prometheus Core developer
- Studied Computer Science in Trinity College Dublin.
- Google SRE for 7 years, working on high-scale reliable systems.
- Contributor to many open source projects, including Prometheus, Ansible, Python, Aurora and Zookeeper.
- Founder of Robust Perception, provider of commercial support and consulting for Prometheus.

Who am I really?

The guy responsible for relabelling.

Instrumentation Labels

Instrumentation labels are to distinguish things happening at the code level inside one metric.

E.g. GET vs POST, Visa vs Mastercard.

Could be from a random library, could be from business logic.

Instrumentation Example

```
from prometheus_client import Counter
c = Counter('my_requests_total', 'HTTP Failures',
            ['method', 'endpoint'])
c.labels('GET', '/').inc()
```

```
my_requests_total{method="GET", endpoint="/"} 1.0
```

So you've got these labels in your app...

You've gone and instrumented your application.

Well done!

Gold Star!

So how do you get those into Prometheus from your live systems?

Where to find targets to monitor

In a push system targets decide what monitoring systems to talk to.

With Prometheus it's the other way around, allowing each team to chose what they want to monitor.

This means though that we need a way to find our targets. Listing them all by hand is not likely to end well.

Enter service discovery.

Service Discovery

There are many supported SD methods: static configs, file, EC2, Consul, Kubernetes, EC2, DNS, Azure, Twitter Serverset and AirBnB Nerve.

File SD reads YML or JSON files off local disk. Inotify used to pick up changes automatically. Intended as hook for when other options don't suit you.

All others work by asking some system for targets over the network.

You've a list of hosts, now what?

So you've got a list of all instances from an EC2 region.

That's fine for monitoring the node exporter that runs everywhere, but how about services that run on only some machines?

Need a way to select which machines to scrape.

And here's the crux

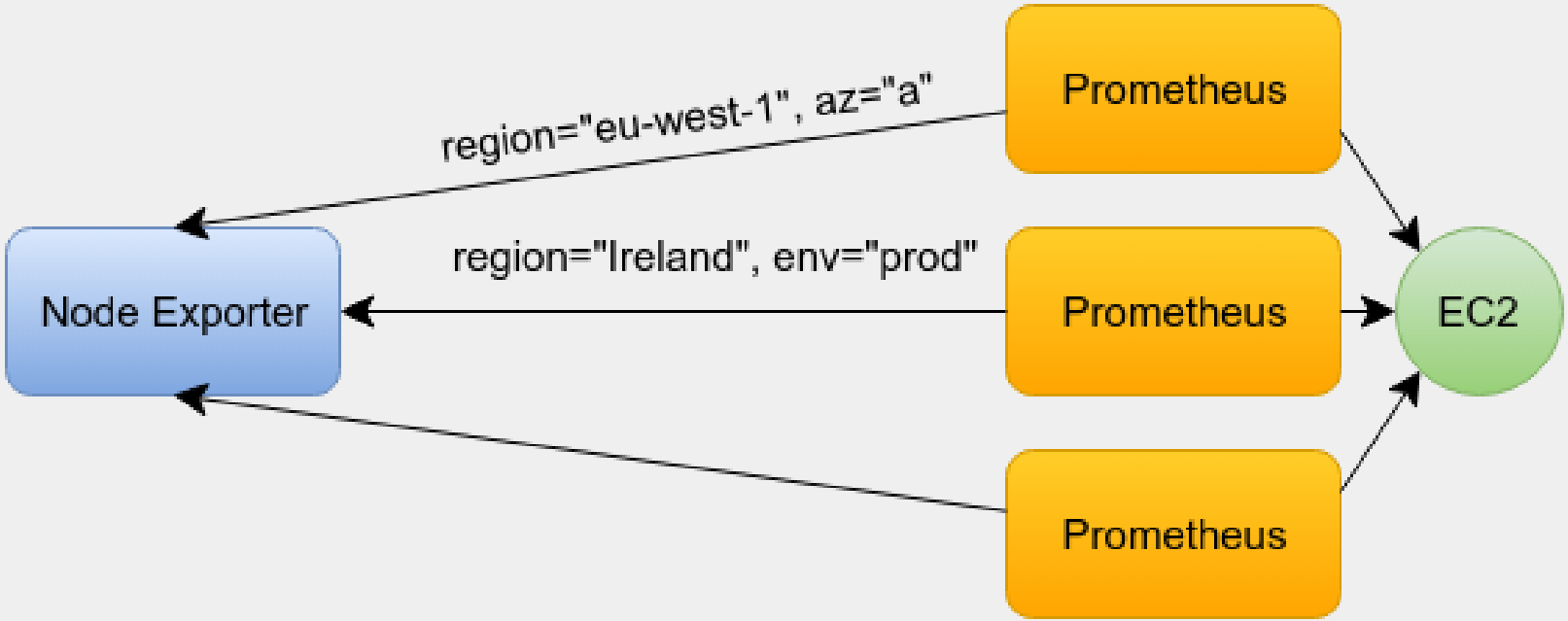
Different organisations do it different ways. Some may use the Name tag, others VPC IDs. It's rarely even consistent within a team, let alone an organisation.

(There's hardcoded assumptions almost every time SD mechanisms are added)

Could representing it as a set of config options, but with so many variants it'd quickly become unwieldy with likely hundreds of interacting config options.

Instead we have relabelling.

What do we want to allow?



Robust Perception

What is relabelling?

Relabelling is a way to take in metadata about a target, and based on that select which targets to scrape.

You can also use it to choose your target labels, as by default you'll only get an `__address__/instance` label.

Choosing Targets

How do you allow for any potential way of going from arbitrary metadata?

When there's little to no structure, regexes are a good choice.

```
relabel_configs:
- source_labels: ["__meta_consul_tags"]
  regex: ".*,production,.*"
  action: keep
```

Keep and drop

The simplest relabel actions are `keep` and `drop`.

If the given regex matches with `keep`, the target continues on.

If it doesn't match, processing halts and we try the next target.

`drop` is the other way round, halting processing if the regex matches.

What if you want to match against two labels?

`source_labels` is a list so you can specify as many labels are you like.

Results will be concatenated, separated by a semicolon.

Can change separator via `separator`

Missing labels will have an empty string.

For more complex rules, `relabel_configs` is a list so you can add as many actions as you like! All actions are applied until a `keep` or `drop` halts it.

Label handling

This is relatively simple so far. We have an SD, we use regexes to pick which targets to scrape.

The real power of Prometheus is labels combined with the query language.

Wouldn't it be nice to be able to make some targets have labels like `env="prod"` or `env="dev"` and aggregate across those?

Munging labels

The core of relabelling is the `replace` action.

It applies a `regex` to the `source_labels`, if it matches interpolates the regex match groups into `replacement` and write the result to the `target_label`.

An empty label means the label is removed. `__meta` labels are discarded after relabelling.

This is all simple in theory.

It gets complicated when you try to map your view of the world into Prometheus labels working off whatever metadata you have.

Example: job name in EC2 Name tag

```
relabel_configs:
- source_labels: ["__meta_ec2_tag_Name"]
  regex: "(.*)"
  action: replace
  replacement: "${1}"
  target_label: "job"
```

Defaults to make things simpler

A label copy is very common, so the defaults reduce this to two lines:

```
relabel_configs:
- source_labels: ["__meta_ec2_tag_Name"]
  target_label: "job"
```

Instance label

The label that SD returns with the host:port is `__address__`.

If no `instance` label is present by the end of relabelling, it defaults to the value of `__address__`.

This means that you can have the `instance` label be something more meaningful than a host:port - such as an EC2 instance id or Zookeeper path.

Avoid adding other labels as readable instance names, it'll break sharing `without` based expressions.

Other labels

Many other settings are also configurable via relabelling.

`scheme`, `metrics_path` and `params` are just defaults, so whether to use http or https could come from service discovery.

For `params` only the first value of each URL parameter is available for relabelling.

This is how the blackbox and SNMP exporters work, changing what would normally be an `__address__` label into a URL parameter.

Other relabel actions

There's two more relabel actions for advanced use cases.

`labelmap` copies labels based on regex substitution.

It's different in that `regex` and `replacement` apply to the label names, not the label values. Useful if you've a set of key/value tags you want to copy wholesale without listing every individual label in the relabel config.

`hashmod` is used with `keep` for sharding. It takes a modulus of a hash of the source labels and puts it in the target label as an integer.

Other notes on labels: Dealing with label clashes

If there's a clash with instrumentation labels, the target label takes precedence.

The scraped label will be prefixed with `exported_`

This behaviour can be changed with `honor_labels: true`, which makes the scraped label win and discards the target label.

In addition, an empty scraped label will remove labels including `instance`. Use this for the Pushgateway and other places where you don't want an `instance` label.

Metric relabel configs

Sometimes you need to temporarily change the scraped metrics while waiting for instrumentation to be fixed.

`metric_relabel_configs` apply to all scraped samples just before they're added to the database. Could use it to drop expensive metrics, or fix a label value.

Beware using expensive or extensive rules as it's applied to every sample.

As up isn't a scraped metric, only `relabel_configs` apply.

From alerts...

```
ALERT MyExampleAlert
  IF rate(my_requests_total[5m]) < 10
  FOR 5m
  LABELS { severity = "page" }
```

The alert will have `method` and `endpoint` labels from the alert expression and a new `severity` label.

External labels and alert relabeling applied too.

...to the Alertmanager

Just as with the rest of the Prometheus stack, labels are core to the Alertmanager.

A tree uses labels to route alerts into groups. Each team can have their own route!

Each group can choose which labels to fan-out notifications by, reducing spam.

Silences are specified using labels, suppressing precisely the alerts you want.

Summary

Instrumentation labels come from the application.

Service discovery creates targets.

Relabelling filters targets and adds target labels to make them meaningful for you.

Metric relabel configs apply to scraped time series.

Alerts can add labels before sending to the alertmanager.

The alertmanager uses labels for routing, grouping, deduplication and silencing.

Resources

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

Label flow: http://www.robustperception.io/life-of-a-label/

Target label best practices:
http://www.robustperception.io/target-labels-are-for-life-not-just-for-christmas/

Robust Perception Blog: www.robustperception.io/blog

Queries: prometheus@robustperception.io

Robust Perception