Exporters and Integrations

In brief, the following topics will be covered in this chapter:

- ✓ Operating system exporter
- ✓ From logs to metrics
- ✓ Blackbox monitoring
- ✓ Pushing metrics
- ✓ More exporters

Operating system exporter

- When monitoring infrastructure, the most common place to start looking is at the OS level.
- Metrics for resources such as CPU, memory, and storage devices, as well as kernel operating counters and statistics provide valuable insight to assess a system's performance characteristics.
- For a Prometheus server to collect these types of metrics, an OS-level exporter is needed on the target hosts to expose them in an HTTP endpoint.
- The Prometheus project provides such an exporter that supports Unix-like systems called the **Node Exporter**, and the community also maintains an equivalent exporter for Microsoft Windows systems called the **WMI exporter**.

The Node Exporter

- The Node Exporter is the most well-known Prometheus exporter, for good reason.
- It provides over 40 collectors for different areas of the OS, as well as a way of exposing local metrics for cron jobs and static information about the host.
  - ✓ The Node Exporter comes with a sane default configuration and some smarts to identify what can be collected, so it's perfectly reasonable to run it without much tweaking.
- Node Exporter collectors might gather different metrics depending on the system being run, as OS kernels vary in the way they expose internal state and what details they make available.

Node Exporter Configuration

❖ Exporters in the Prometheus ecosystem usually collect a specific set of metrics from a given process.

❖ The Node Exporter differs from most other exporters as machine-level metrics span a wide range of subsystems, and so it is architected to provide individual collectors, which can be turned on and off, depending on the instrumentation needs.

❖ Enabling collectors that are turned off by default can be done with the `--collector.<name>` set of flags; enabled collectors can be disabled by using the `--no-collector.<name>` flag variant.

Node Exporter Configuration

- From all the collectors enabled by default, one needs to be singled out due to its usefulness as well as its need of configuration to properly work.
- The textfile collector enables the exposition of custom metrics by watching a directory for files with the .prom extension that contain metrics in the Prometheus exposition format.
- The `--collector.textfile.directory` flag is empty by default and so needs to be set to a directory path for the collector to do its job. It is expected that only instance-specific metrics be exported through this method, for example:
  - ✓ Local cron jobs can report their exit status through a metric (finish timestamp is not useful to record, as the metrics file modification timestamp is already exported as a metric) Informational metrics (that only exist for the labels they provide), such as VM flavor, size, or assigned role How many package upgrades are pending, if a restart is required Anything else not covered by the built-in collectors

Node Exporter Configuration

- This exporter produces a large number of metrics, depending on which collectors are enabled.
  - ✓ `node_cpu_seconds_total`, which provides the number of seconds cumulatively used per core for all the available CPU modes, is very useful for understanding the CPU utilization
  - ✓ `node_memory_MemTotal_bytes` and `node_memory_MemAvailable_bytes`, which allow calculating the ratio of memory available
  - ✓ `node_filesystem_size_bytes` and `node_filesystem_avail_bytes`, which enable the calculation of filesystem utilization
  - ✓ `node_textfile_scrape_error`, which tells you if the textfile collector couldn't parse any of the metrics files in the textfile directory **(when this collector is enabled)**

Blackbox monitoring

- Introspection is invaluable to gather data about a system, but sometimes we're required to measure from the point of view of a user of that system.
- In such cases, probing is a good option to simulate user interaction.
- As probing is made from the outside and without knowledge regarding the inner workings of the system, this is classified as blackbox monitoring.

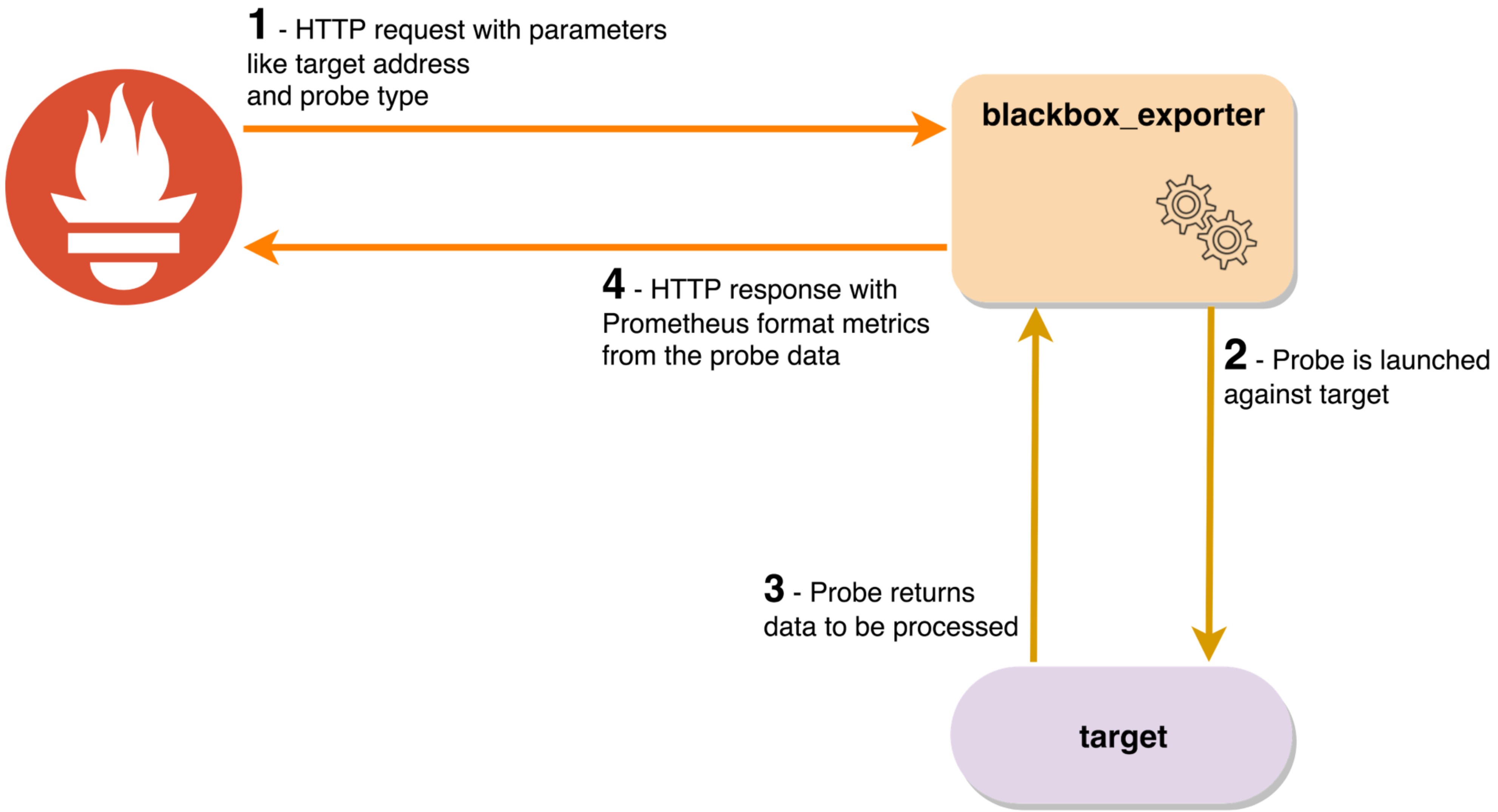
Blackbox exporter

- `blackbox_exporter` is one of the most peculiar of all the currently available exporters in the Prometheus ecosystem.
- Its usage pattern is ingenious and usually, newcomers are puzzled by it.
- The `blackbox_exporter` service exposes two main endpoints:
  - ✓ `/metrics`: Where its own metrics are exposed
  - ✓ `/probe`: It is the query endpoint that enables blackbox probes, returning their results in Prometheus exposition format
- Besides the two previous endpoints, the / of the service also provides valuable information, including logs for the probes performed.
- This endpoint is available in the static infrastructure test environment at port 9115 on `blackbox_exporter` endpoint.

Blackbox exporter

- The blackbox exporter supports probing endpoints through a wide variety of protocols natively, such as TCP, ICMP, DNS, HTTP (versions 1 and 2), as well as TLS on most probes.
- Additionally, it also supports scripting text based protocols such as IRC, IMAP, or SMTP by connecting through TCP and configuring what messages should be sent and what responses are expected; even plain HTTP would be possible to script but, as HTTP probing is such a common use case, it's already built in.
- This exporter doesn't cover all the blackbox-style monitoring needs.
  - ✓ For those cases, writing your own exporter might be needed.

Blackbox exporter


1 - HTTP request with parameters like target address and probe type
blackbox_exporter
4 - HTTP response with Prometheus format metrics from the probe data
2 - Probe is launched against target
3 - Probe returns data to be processed
target

Installing Blackbox Exporter

```
$ wget \
https://github.com/prometheus/blackbox_exporter/releases/download/v0.18.0
/blackbox_exporter-0.18.0.linux-amd64.tar.gz

$ tar -xzf blackbox_exporter-0.18.0.linux-amd64.tar.gz

$ cd blackbox_exporter-0.18.0.linux-amd64/

$ sudo ./blackbox_exporter
```

Monitoring ICMP Check

- Visit http://localhost:9115/ in your browser to see status page

```
http://localhost:9115/probe?module=icmp&target=localhost

http://localhost:9115/probe?module=icmp&target=www.google.com

http://localhost:9115/probe?module=icmp&target=www.google.com&debug=true
```

ICMP Check with Custom Settings

- To prefer IPv4 instead, you can add a new module with the preferred_ip_protocol.
- ipv4 option to blackbox.yml:

```
icmp_ipv4:
  prober: icmp
  icmp:
    preferred_ip_protocol: ip4
```

```
http://localhost:9115/probe?module=icmp_ipv4&target=www.google.com
```

TCP

- The Transmission Control Protocol is the TCP in TCP/IP.
  - ✓Many standard protocols use it, including websites (HTTP), email (SMTP), remote login (Telnet and SSH), and chat (IRC).
- The `tcp` probe of the Blackbox exporter allows you to check TCP services, and perform simple conversations for those that use line-based text protocols.
- To check if your local SSH server is listening on port 22 :

```
http://localhost:9115/probe?module=tcp_connect&target=localhost:22
```

ssh_banner

- The ssh_banner module goes further, checking for a particular response from the remote server:

```
ssh_banner:
  prober: tcp
  tcp:
    query_response:
    - expect: "^SSH-2.0-"
```

- As the very start of an SSH session is in plain text, you can check for this part of the protocol with the tcp probe.
- This is better than tcp_connect as you are not only checking that the TCP port is open, but that the server on the other end is responding with an SSH banner.
  - ✓ If your server returned something different, the expect regex will not match, and probe_success will be 0.
  - ✓ In addition, probe_failed_due_to_regex would be 1.

DNS

- The `dns` probe is primarily for testing DNS servers.
- Test that your DNS servers were responding over TCP, in your blackbox.yml:

```
dns_tcp:
  prober: dns
  dns:
    transport_protocol: "tcp"
    query_name: "www.prometheus.io"
```

```
http://localhost:9115/probe?module=dns_tcp&target=8.8.8.8
```

- Note that the target URL parameter is the DNS server that is talked to, and the `query_name` is the DNS request sent to the DNS server.
  - ✓ This is the same as if you ran the command dig -tcp @8.8.8.8 www.prometheus.io.

DNS probe

- Aside from testing DNS servers, you could also use a `dns probe` to confirm that specific results are being returned by DNS resolution.
- But usually you want to go further and communicate to the returned service via HTTP, TCP, or ICMP, in which case one of those probes makes more sense as you get the DNS check for free.
- An example of using the dns probe to check for specific results would be to check that your MX records have not disappeared.

```
dns_mx_present_rp_io:
  prober: dns
  dns:
    query_name: "robustperception.io"
    query_type: "MX"
    validate_answer_rrs:
      fail_if_not_matches_regexp:
        - ".+"
```

DNS probe

- After restarting the Blackbox exporter, you can visit http://localhost:9115/probe?module=dns_mx_present_rp_io&target=8.8.8.8 to check that robustperception.io has MX records.
- Note that as the `query_name` is specified per module, you will need a module for every domain that you want to check.
- The `dns` probe has more features intended to help check for aspects of DNS responses, such as authority and additional records.

HTTP

```
http://192.168.42.11:9115/probe?module=http_2xx&target=google.com
```

HTTP

- For example, I may want to test that users to http://www.robustperception.io end up redirected to a HTTPS website, with a 200 status code, and that the word “Prometheus” is in the body. To do so you could create a module like:

```
http_200_ssl_prometheus:
  prober: http
  http:
    valid_status_codes: [200]
    fail_if_not_ssl: true
    fail_if_not_matches_regexp:
      - Prometheus
```

```
http://localhost:9115/probe?module=http_200_ssl_prometheus&target=http://www.robustperce
ption.io
http://localhost:9115/probe?module=http_200_ssl_prometheus&target=http://prometheus.io
```

BlackBox Configuration

- The scrape job configuration for blackbox probes is unusual, in the sense that both the prober module and the list of targets, whether static or discovered, need to be relayed to the exporter as HTTP GET parameters to the /probe endpoint.
- To make this work, a bit of relabel_configs magic is required.

BlackBox Configuration

- Using the following Prometheus configuration snippet as an example, we're setting up an ICMP probe against the Prometheus instance, while blackbox_exporter is running:

```
- job_name: 'blackbox-icmp'
  metrics_path: /probe
  params:
  module: [icmp]
  static_configs:
    - targets:
        - prometheus.prom.inet
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
  replacement: target01:9115
```

Prometheus Configuration

- The Blackbox exporter takes a module and target URL parameter on the /probe endpoint.
- Using the params and metrics_path, you can provide these in a scrape config, but that would mean having a scrape config per target, which would be unwieldy as you could not take advantage of Prometheus's ability to do service discovery.
- The good news is that you can take advantage of service discovery, as the `__param_<name>` label can be used to provide URL parameters in relabelling.

Prometheus Configuration

```
scrape_configs:
  - job_name: blackbox
metrics_path: /probe
params:
module: [http_2xx]
static_configs:
- targets:
- http://www.prometheus.io
- http://www.robustperception.io
- http://demo.robustperception.io
relabel_configs:
- source_labels: [__address__]
  target_label: __param_target
- source_labels: [__param_target]
  target_label: instance
- target_label: __address__
  replacement: 127.0.0.1:9115
```

Prometheus Configuration

- A default job label, custom path, and one URL parameter are specified:

```
- job_name: 'blackbox'
  metrics_path: /probe
  params:
    module: [http_2xx]
```

Prometheus Configuration

❖There are three websites that you will be probing:

```
static_configs:
  - targets:
    - http://www.prometheus.io
    - http://www.robustperception.io
    - http://demo.robustperception.io
```

Prometheus Configuration

- The relabel_configs is where the magic happens.
- First, the __address__ label becomes the target URL parameter and secondly also the instance label.
- At this point the instance label and target URL parameter have the value you want, but the __address__ is still a URL rather than the Blackbox exporter.
- The final relabeling action sets the __address__ to the host and port of the local Blackbox exporter.

```
relabel_configs:
  - source_labels: [__address__]
    target_label: __param_target
  - source_labels: [__param_target]
    target_label: instance
  - target_label: __address__
    replacement: 127.0.0.1:9115
```