# Workshop Part 1 - Introduction

## Part 1 - Section A - Prometheus self-monitoring

- Go to https://prometheus.io/download/ and download the latest stable release of Prometheus for your architecture
- Untar the tarball
- Cd into the directory
- Setup Prometheus to monitor itself by putting the following in Prometheus.yml:

```
global:
    scrape_interval: 10s
scrape_configs:
  - job_name: 'prometheus'
    target_groups:
      - targets: ['localhost:9090']
```

- Run it: ./prometheus
- Prometheus should now be accessible on http://localhost:9090/

## Part 1 - Section B - Adding a job

- Add the following to the end of your prometheus.yml, and reload the config. Watch out for indentation!

```
  - job_name: node
    target_groups:
      - targets: ['localhost:9100']
```

- Go to https://prometheus.io/download/ and download the latest stable release of node_exporter for your architecture
- Untar the tarball
- Cd into the directory
- Run it: ./node_exporter
- The node exporter should now be accessible on http://localhost:9100/

Note: If you're not on Linux, you can use demo.robustperception.io:9100 instead. The node exporter currently has good metrics primarily for Linux and FreeBSD.

# Workshop Part 2 - Data Model

## Part 2 - Section A  - Notes

Gauges

- Snapshot of state
- Can go up and down
- You care about absolute value
- Export raw data

Types

- All Prometheus data points are 64bit floating point
- Timestamps have millisecond resolution

Counters

- Count events, you care about rate of increase
- Start at 0, can only be incremented
- Resets automatically handled by Prometheus counter functions
- End in _total by convention

Labels

- Key-value pairs
- Any utf-8 value
- __ is reserved
- __name__ is the metric name

Summary & Histogram

- Compound types to make common case easier
- Can be used to track distributions
- Summary without quantiles is cheap, use it freely

# Workshop Part 3 - PromQL

## Part 3 - Section A  - Notes

Types

- PromQL is strongly typed
    - Scalar
    - Vector (or Instant Vector)
    - Range Vector
    - String

Matchers

- =, !=, =~, !~
- Can mix and match
- Regexes are anchored

Functions

- Full list in official docs
- Use rate, increase, irate, and resets only on counters.
- Use rate/irate before histogram_quantile
- Absent is for alerting on time series not existing
- Don't use drop_common_labels or count_scalar

Aggregators

- sum, count, min, max, avg, stddev and stdvar
- Use without to select output labels
- Can also use by, but it makes sharing expressions harder
- Don't use keep_common
- Always rate and then sum, never sum then rate

Operators

- Arithmetic, Comparison and Logical
- Using ignoring to choose which labels not to bucket on. Can also use on.
- group_left/group_right for many-to-one matches
- Bool modifier for comparisons to return 0/1, otherwise filters
- Logical operators always many-to-many

HTTP API

- Query range, each step is independently calculated

Recording Rules

- Run regularly by Prometheus
- For precomputation, when you need a range vector, and for aggregates for federation
- See official docs for naming rules best practices. aggregation:metric:operations.

# Part 3 - Section B - Recording Rules

In prometheus.yml add:

```
rules:
 - prometheus.rules
```

And create prometheus.rules with:

```
job:up:sum = sum without(instance)(up{job="prometheus"})
```

Then HUP or restart prometheus.

# Workshop Part 4 - Instrumentation

## Part 4 - Section B - Notes

Client libraries

- Do more than marshalling data
- Take care of bookkeeping, concurrency etc.
- Designed for ease of use

Key metrics

- Online serving: requests, errors, latency
- Offline processing: items in, in progress, and out. Heartbeat for latency.
- Batch: Last time it succeeded. Use with Pushgateway

Best Practices

- See instrumentation best practices and writing an exporter guidelines on prometheus.io
- Make metric names useful to someone who has no knowledge of the system
- Put units in metric names
- Avoid clashes with metric type prefix, and common target label names
- Failure should be a separate metric. Don't break latency out by failure/success
- Beware cardinality, and over-using labels
- Never have a "total" label, use aggregation operators

## Part 4 - Section B - Python Exercise

Download code from https://github.com/brian-brazil/prometheus_workshop_python

Add basic monitoring for both the client and the server.

Hook both into your Prometheus server.

Graph the three key metrics.

# Workshop Part 5 - Configuring Prometheus and Exporters

## Part 5 - Section A - Notes

Prometheus.yml

- Can be reloaded with HUP or /-/reload
- No templating, use your configuration management's templating
- External_labels is to distinguish different Prometheus servers
- Rules can have a file glob

Scrape_configs

- Service discovery to find targets, file SD as fallback
- SD mechanisms provide metadata
- Metadata can be used with relabelling to select which targets to scrape via keep and drop actions
- Replace relabel action can be used to set, copy and munge labels from metadata. Defaults chosen to make common cases simpler.
- Regexes are anchored
- Various options for HTTP settings and auth. Some HTTP options relableable
- __address__ is default value for instance label
- Scraped labels that clash with target labels get prefixed with exported_
- Metric_relabel_configs applies to all scraped samples
- hashmod and labelmap actions for advanced use cases
- Think about how you could pick up new services without having to add a new scrape config

Exporters

- 1-to-1 with the process they monitor, as close as we can get to direct instrumentation
- Scheduling and service discovery always stays with Prometheus
- https://prometheus.io/docs/instrumenting/writing_exporters/
- Node exporter is for machines/VMs. Use textfile collector for machine-level batch jobs.
- Blackbox and SNMP exporter are different run beside Prometheus and pass target as a URL parameter.

# Part 5 - Section B - Relabelling Exercise

In this example we're going to learn about using File SD and Consul for discovery.

We're not actually going to run Consul, instead we're going to emulate it with File SD.

First put this file into file_sd, and get the targets showing up on the Prometheus status page:

```
[
  {
    "targets": [ "myslave1:9100"],
    "labels": {
      "__meta_consul_tags": ",database,prod,"
    }
  },
 {
    "targets": [ "myhost1:9100"],
    "labels": {
      "__meta_consul_tags": ",prod,host"
    }
  },
 {
    "targets": [ "myhost2:9100"],
    "labels": {
      "__meta_consul_tags": ",host,dev,"
    }
  }
]
```

Now use target relabelling to have an `env` tag that has the value either `prod` or `dev`.

Can you make it ignore databases?

# Workshop Part 6 - Alerting

## Part 6 - Section A - Notes

Architecture

- Many Prometheus servers, one Alertmanager
- Prometheus sends alerts based on PromQL, Alertmanager route, groups, de-duplicates and sends notifications
- Separate Alertmanager so alerts from multiple Prometheus servers can be grouped into one notification

Alerts in Prometheus

- Alerts are another form of recording rule
- Use FOR to prevent brief spikes and other issues from causing pages
- LABELS is to help routing alerts, e.g. severity="page"
- ANNOTATIONS is to add additional information to put inside notifications
- Up tells you if the scrape of a target failed
- Targets can be removed by service discovery
- Staleness means old time series hang around for 5 minutes
- Alert on symptoms, not causes
- Beware missing labels, avoid them at the instrumentation stage
- Use rate for alerts, not irate
- Combine conditions using the and operator
- Batch jobs should push last success time to pushgateway (service-level) or node exporter textfile collector (machine-level). Compare to time()

Alertmanager

- alertmanager.yml holds the config
- Many receivers, each with its own config.
- Templates to configure how notifications look.
- Can get notifications when alerts are resolved, avoid for sanity.
- Tree of routes to find settings to apply to alerts.
- Post-order transversal.
- Continue matches, and continues transversal to match.
- Groups send no more often than every group interval
- First notification of a group is delayed by group wait
- Notifications repeated after repeat interval
- Inhibits stop alerts based on other alerts
- Silences stop alerts for a while
- Have a way to deal with non-critical alerts

# Part 6 - Section B - Exercise

Create a webhook receiver that prints out the message it receives (hint: a Python HTTP server that prints the payload it gets is one option).

Create an alertmanager configuration that routes severity=page and severity=chat alerts to different places.

Send alerts based on the Prometheus you have running. How long after an alert starts firing do you get a notification? What happens when an alert stops firing?

# Workshop Part 7 - Grafana

## Part 7 - Section A - Notes

Grafana

- Dashboard system, supports many different backends
- SQLite persistence by default
- Datasources are where data come from
- Dashboards have Rows which have Panels
- Graph is the main type of panel you'll use

## Part 7 - Section B - Exercise

Install Grafana. Default credentials are admin/admin.

Add your Prometheus as datasource.

Create a dashboard with graphs of errors and latency of the Python instrumentation example.

Use templates to create a node exporter dashboard that's not specific to one machine.

Can you use repeated templates to show a graph per filesystem?

Can you use a table to show predicted time until each filesystem fills?

# Workshop Part 8 - Practical Deployment

- 800K samples/s is record for Prometheus.
- Plan for 3-4kB of RAM per active time series, and 1.3bytes/sample of disk with varbit encoding
- SSD recommended.
- Ultra-granular monitoring isn't free, tradeoff against time and cost.
- 60s resolution is a good starting point.
- Run at least one Prometheus per datacenter
- One Prometheus per team/service works well, and lets you scale.
- Global Prometheus servers can pull in aggregated stats via federation.
- For HA, run two identical Prometheus servers.
- Alertmanager currently requires manual handling on failure.
- Prometheus focuses on reliability over perfection.
- Use cross, meta and 3rd part monitoring to detect monitoring system failure.
- Prometheus storage is considered ephemeral. Long term storage will handle historical data.
- For migration start with exporters, low risk with immediate value.
- Use reverse proxy like nginx for auth.
- Prometheus supports credentials when scraping.
- Mtail is useful for logs.