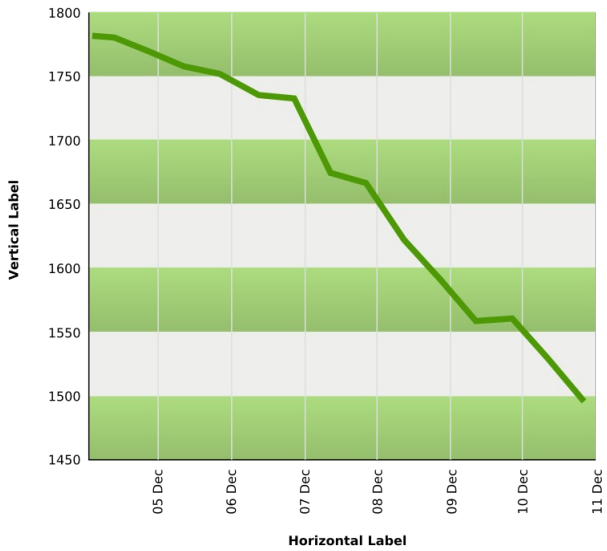
Cloud Monitoring with Prometheus and Grafana

Some things we need from monitoring

- Servers and/or VMs are up
- Services are up
- Services are performing well (good response times)
- Not about to run out of resources
- Problem notification
- Collect data to help debug issues
- Collect data to plan for capacity growth



Components of a monitoring platform

- Collection of measurements
- Data Storage
- Visualization (graphs)
- Analysis / Alerting
- Inventory (what to monitor)

The Old Way

- Data collection: **SNMP**
  - Standardized for network equipment, but difficult to interface to for applications
- Data storage: **RRDtool**
  - Designed to minimize *storage space* by throwing data away
  - But heavy use of disk reads and writes (IOPS)
- Visualization, Alerting, Inventory: mostly monolithic tools
  - e.g. Cacti, LibreNMS, Zabbix, CheckMK… Graphite

The new way: Time Series Database (TSDB)

Optimized for working with time series data

- Efficient disk layout: e.g. adjacent samples from the same time series next to each other
- Compresses extremely well
- *Much* more efficient use of I/O & CPU than RRDtool

Retains data with full fidelity

- Unlike RRD which is designed to throw data away!

API and query language for open data access

Choosing a time series database

There are many!

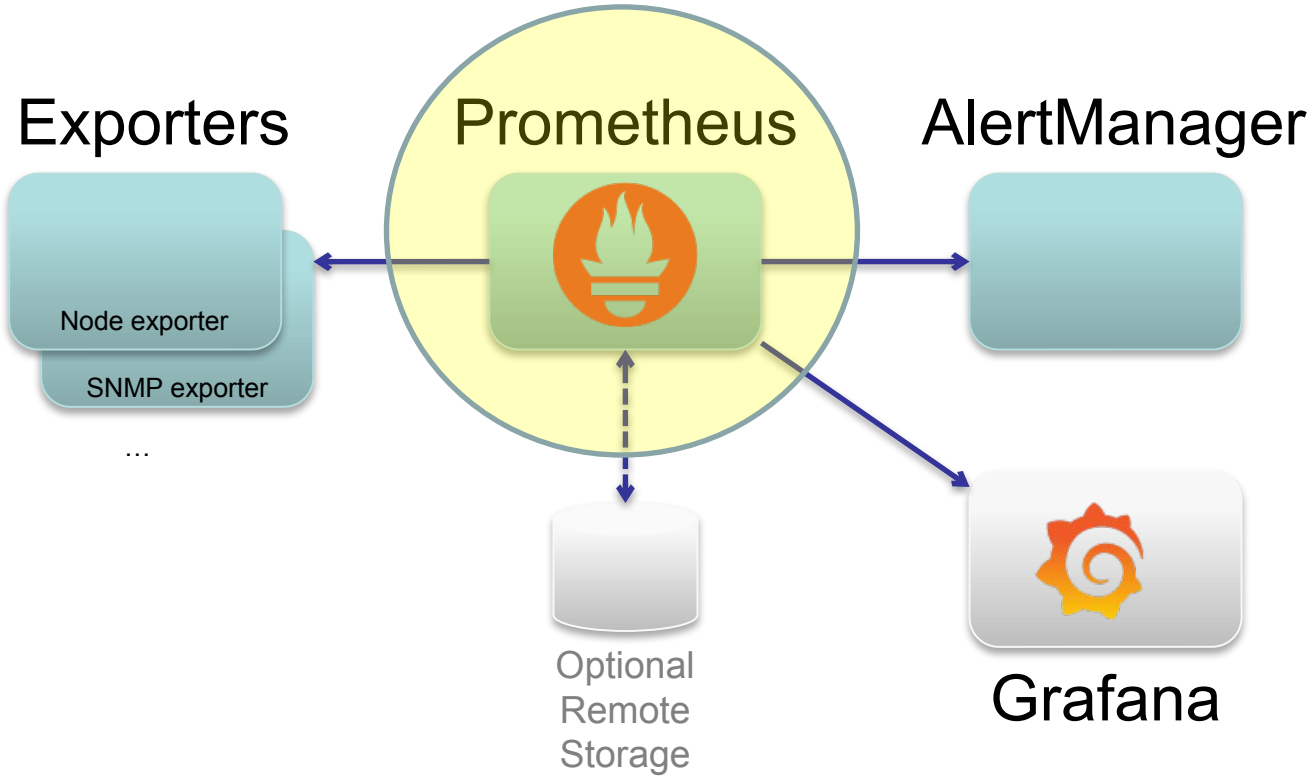
InfluxDB, Cassandra, TimescaleDB, VictoriaMetrics, OpenTSDB, Clickhouse, Yottadb…

We needed to pick one for this course

… so we picked **Prometheus**

Vibrant community, widely used in both cloud and on-prem

Prometheus architecture



Lightweight, efficient, and the origin of OpenMetrics

Prometheus terminology

Metrics are generated by "*exporters*"

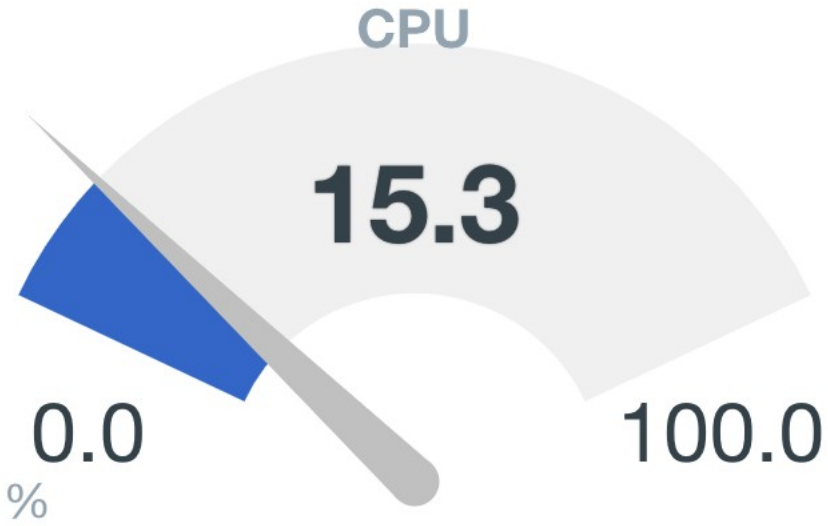
Central prometheus server collects data by making periodic HTTP requests to exporters – called "*scraping*"

Prometheus needs to know which targets to scrape – this is called "*service discovery*"

In the simplest case, you just give it a static list of targets

(But there are many other ways, e.g. query AWS API to get a list of EC2 instances; query Kubernetes API to get a list of pods)

Metrics



Something that you *measure*

Metrics are always *numeric values*

Types of metric:

- Gauges *(e.g. available disk space, temperature)*
- Counters *(e.g. bytes received, total time spent working)*
  - *counters only ever INCREASE*
- Histograms *(e.g. number of requests grouped by latency)*
  - *counters divided into buckets*

How do you gather metric data?

Passive measurement, e.g.

- Counters which increment when packets pass through
- Count events or logs matching a pattern

Active measurement, e.g.

- Make a test HTTP request, measure latency
- Generate test traffic and measure throughput (speedtest)

Add instrumentation to applications

Common exporters

- *node_exporter:* Linux/BSD server metrics
- *windows_exporter:* Windows server metrics
- *snmp_exporter:* make SNMP queries, return metrics
- *blackbox_exporter:* active ICMP/HTTP/DNS tests
- many more available
- many applications have built-in prometheus exporters
- easy to write your own

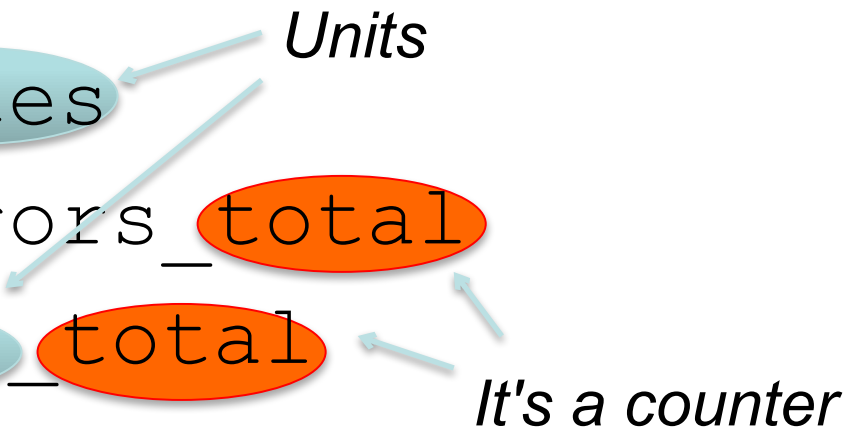
Metric name

Identifies the type of thing you are measuring

- e.g. "`node_filesystem_avail_bytes`"

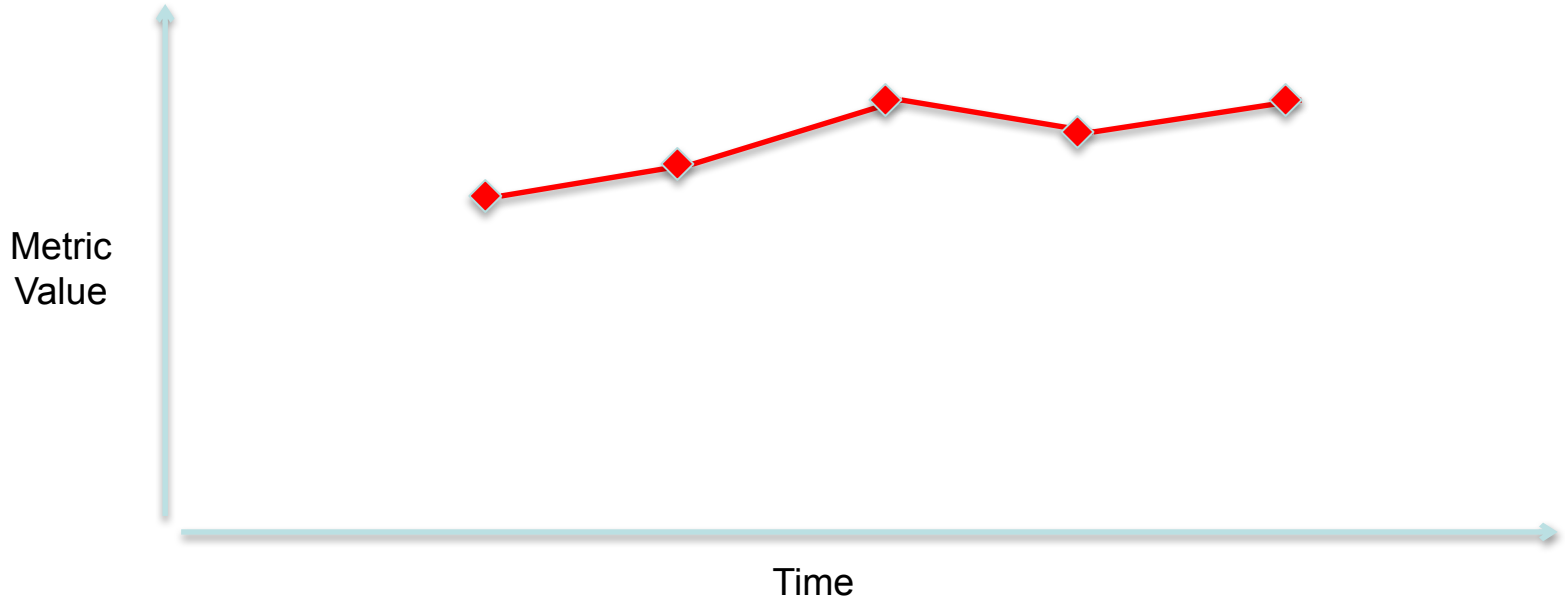
Good metric names show the units and whether it's a gauge or a counter

- `process_resident_memory_bytes`
- `promhttp_metric_handler_errors_total`
- `node_disk_read_time_seconds_total`



Time series

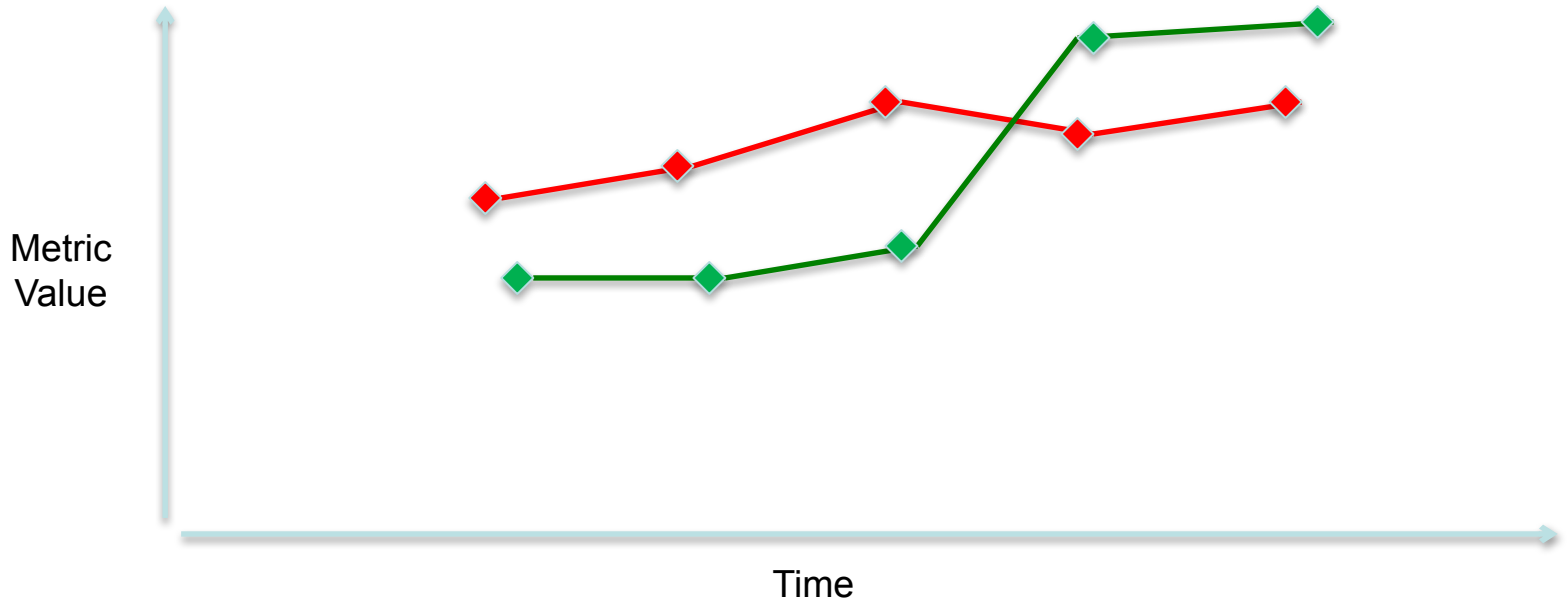
Repeated measurements of the same variable over time



node_filesystem_avail_bytes

Time series

There can be distinct time series for the *same metric*



node_filesystem_avail_bytes{instance="nuc1",mountpoint="/"}
node_filesystem_avail_bytes{instance="nuc1",mountpoint="/home"}

Labels

Used to identify a particular time series

- `ifHCInOctets{instance="rtr1",ifName="gi0"}`
- `ifHCInOctets{instance="rtr1",ifName="gi1"}`
- `ifHCInOctets{instance="rtr2",ifName="gi0"}`
- `ifHCOutOctets{instance="rtr1",ifName="gi0"}`

Every distinct combination of (metric name + labels) defines a unique time series

Change, add or remove any label => different time series!

** But order of labels doesn't matter*

OpenMetrics data format

```
ifHCInOctets{instance="rtr1",ifName="gi1"} 123456.0
```

Metric Name | Labels | Value | *(Optional Timestamp)*

That's it!

Other metrics formats exist, e.g. InfluxDB line protocol, Graphite protocol, CSV

Scraping in action

- HTTP request returns a list of OpenMetrics lines
- This is almost ludicrously simple (and easy to debug)

```
$ curl http://nuc1:9100/metrics
…
# HELP node_arp_entries ARP entries by device
# TYPE node_arp_entries gauge
node_arp_entries{device="br255"} 12
node_arp_entries{device="isolate"} 1
node_arp_entries{device="lxdbr0"} 2
…
```

Prometheus time series database

Prometheus has a built-in TSDB, just needs a directory to write to

`--storage.tsdb.path=/var/lib/prometheus/data/`

Highly optimized (chunking, write-ahead log etc)

By default, keeps 15 days of data; tune with global flag

`--storage.tsdb.retention.time=31d`

Also options for long-term archival and remote storage

Querying Prometheus

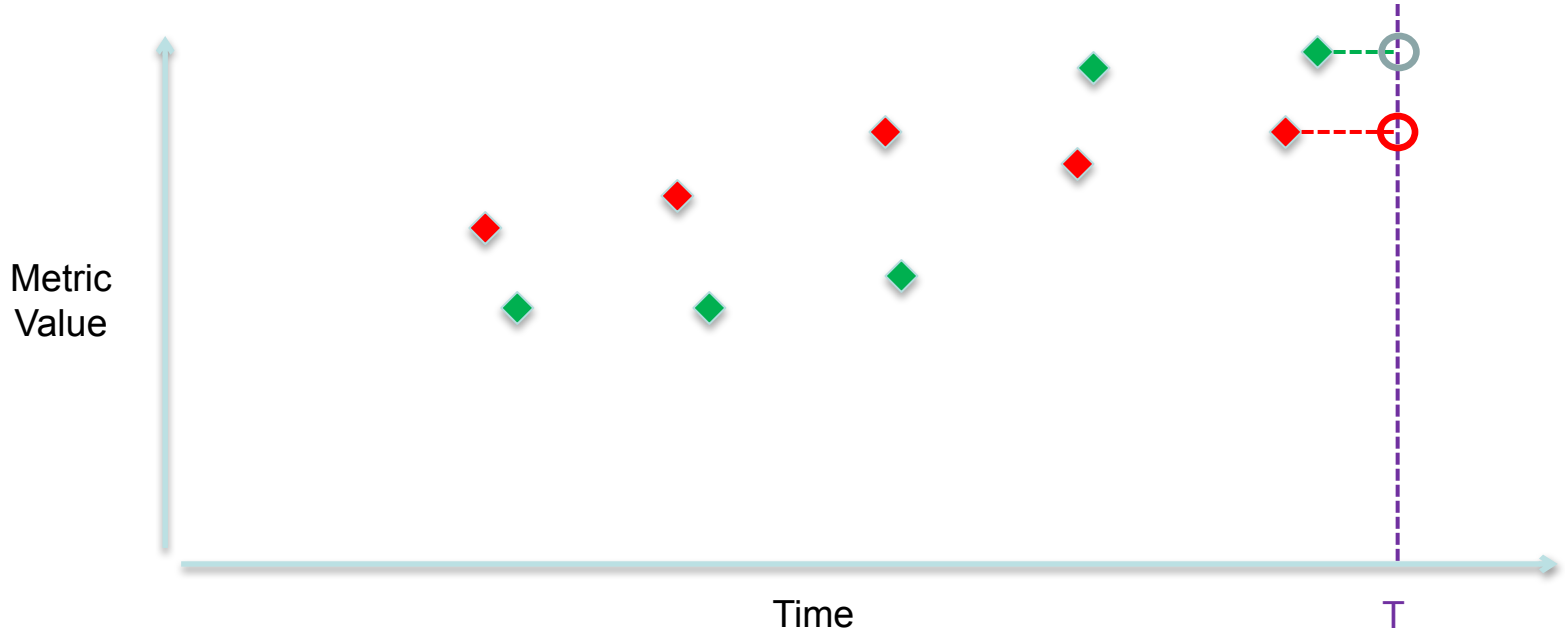
- Prometheus has its own query language – *PromQL*
- The simplest query is just a metric name

```
node_filesystem_avail_bytes
```

- This returns the values for zero or more time series at a particular point in time (by default: “now”)
- This is called an “instant vector”
  - “vector” because it can return multiple values
  - the value of a metric at time T is the most recent sample *at or before time T*

PromQL query: “node_filesystem_avail_bytes”

Instant vector at time T



(by default, lookback maximum 5 minutes)

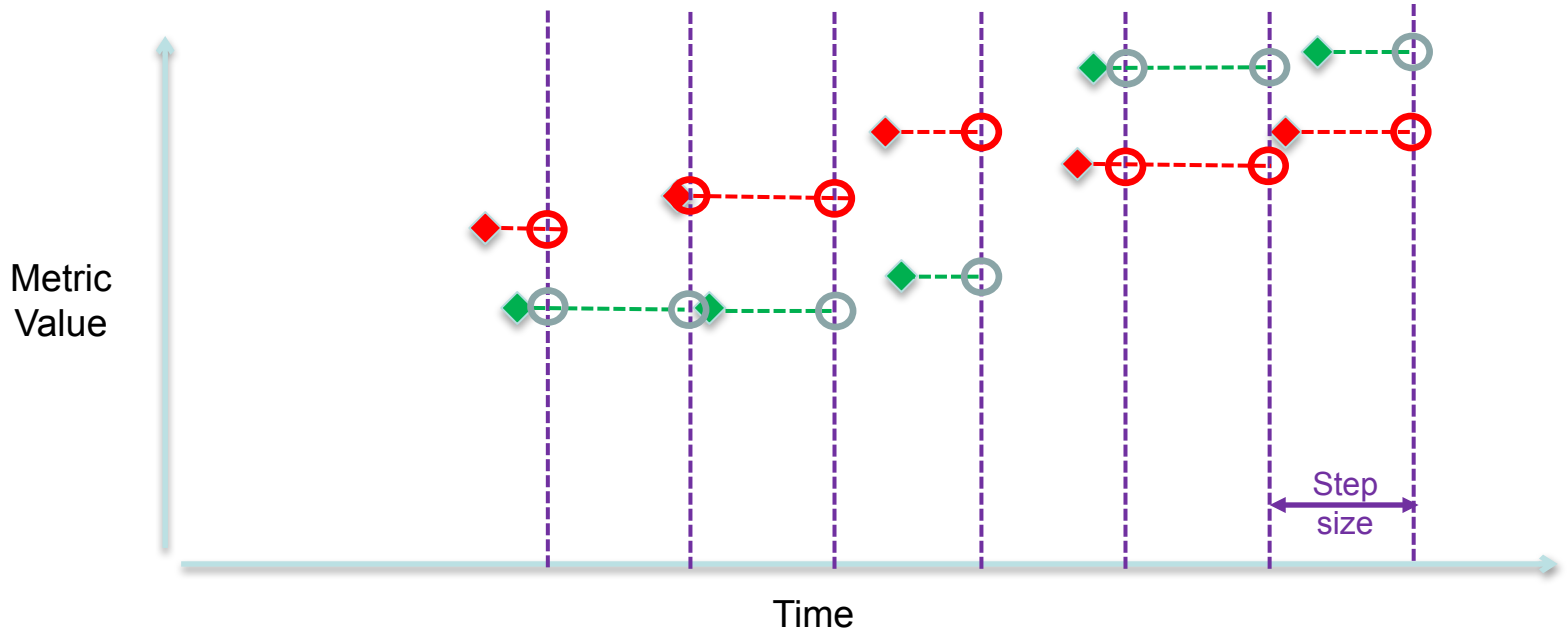
node_filesystem_avail_bytes{instance="nuc1",mountpoint="/"} 52428800.0
node_filesystem_avail_bytes{instance="nuc1",mountpoint="/home"} 62914560.0

Graphing

- To create a graph, the instant query is repeated at multiple instants in time
- The graphing client chooses the start and end time and the size of the steps

PromQL graph: "node_filesystem_avail_bytes"

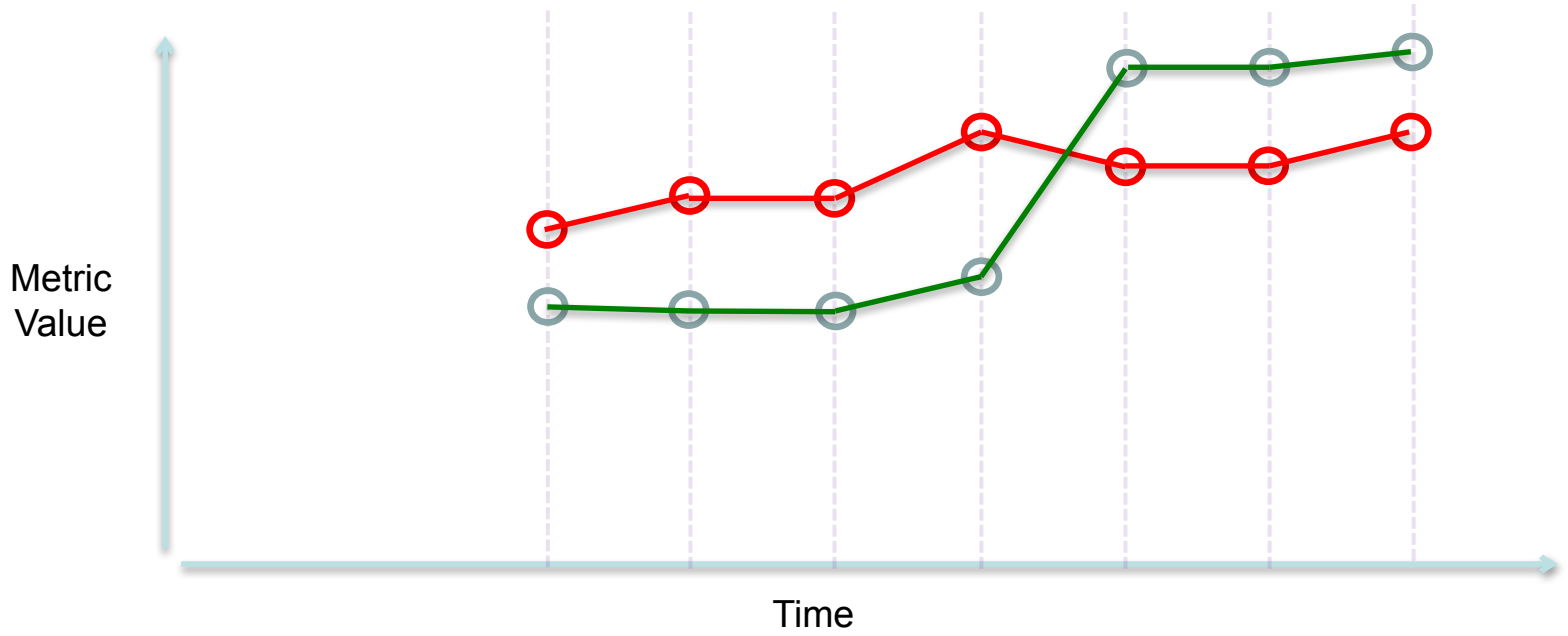
Samples at time intervals



node_filesystem_avail_bytes{instance="nuc1",mountpoint="/"}
node_filesystem_avail_bytes{instance="nuc1",mountpoint="/home"}

PromQL graph: “node_filesystem_avail_bytes”

Final graph of samples



node_filesystem_avail_bytes{instance="nuc1",mountpoint="/"}
node_filesystem_avail_bytes{instance="nuc1",mountpoint="/home"}

PromQL filtering: select a subset

- Filter by label values or patterns

```
node_filesystem_avail_bytes{mountpoint="/home"}
ifHCInOctets{instance=~"rtr[1-2]"}
```

- Filter by time series value

```
node_filesystem_avail_bytes < 100000000
```

  - Commonly used for alerting expressions
  - Note that this is not a boolean (true/false) result: it filters out values which don't meet the criteria, and passes those that do
  - If *any* value is present in the result set, an alert is fired

More PromQL features

- Arithmetic

```
node_filesystem_avail_bytes / 1024
node_filesystem_avail_bytes / node_filesystem_size_bytes
```

- Functions across time series

```
sum(node_filesystem_avail_bytes)
```

Returns a 1-element instant vector with the *total available size*

```
min(node_filesystem_avail_bytes)
```

Returns a 1-element instant vector with the *lowest available size*

Range vector queries

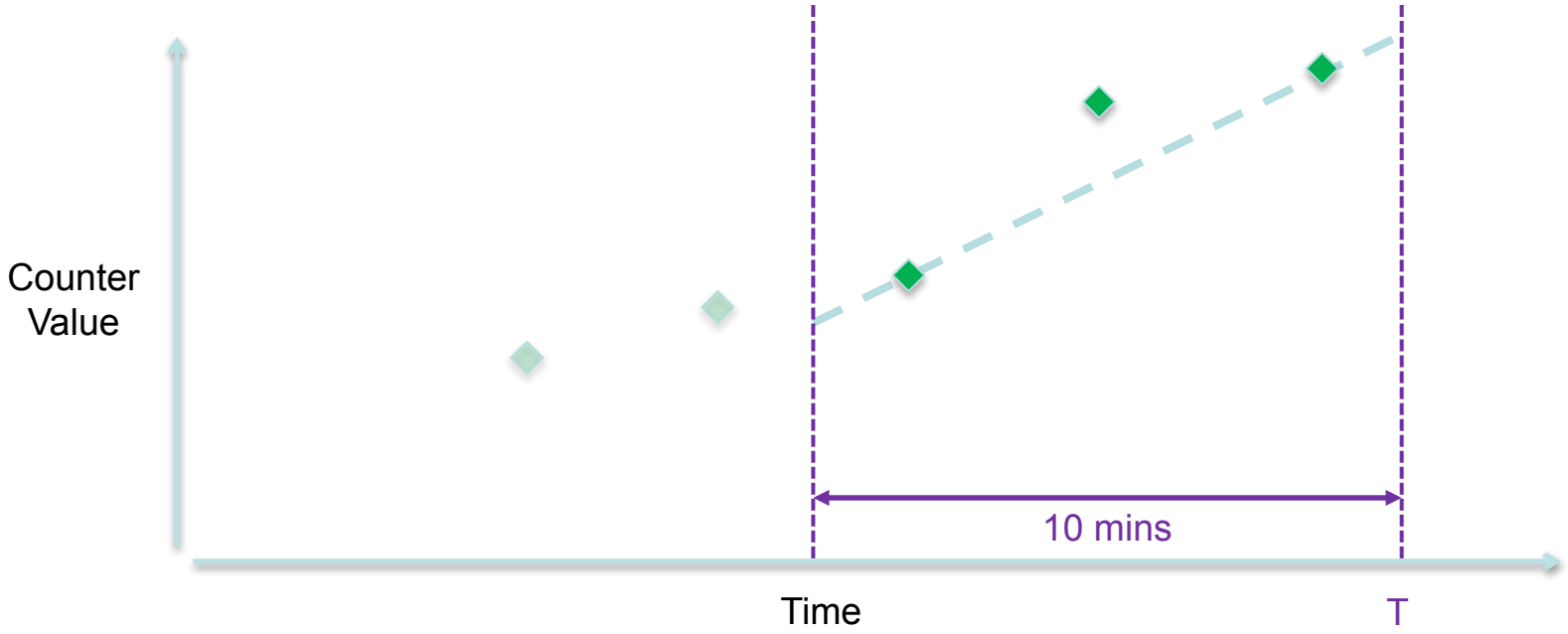
- Returns all of the data points within a time window up to the selected instant, e.g.

  `ifHCInOctets[10m]`
- This allows you to apply functions across a time range
- Converting counters into rates is the most important, e.g.

  `rate(ifHCInOctets[10m])`
- Can also calculate average, max, min etc over time

rate(ifHCInOctets[10m])

Input is a range vector, result is an instant vector



Average rate, e.g. bytes per second, is calculated between the *first* and *last* points in the range vector. It's the slope of the dashed line, except when there are counter resets

Roadmap from here

- Exporters and Service Discovery
  - in the lab you will connect node_exporter, proxmox-pve-exporter and ceph to Prometheus and Grafana
- Alerting
  - alerting rules with PromQL: you can test your expressions in the web interface
  - alertmanager for delivery of notifications
- API consumers (e.g. Grafana)

That's the basics covered!

- Any questions?