Counter, Gauge, Upper 90 - Oh my!

let’s learn enough to ~~worry~~ think about metrics

Amit Saha
@echorand

My monitoring journey - Stage 1

“When an ostrich is afraid,

it will bury its head in the ground, assuming that

because it cannot see, it cannot be seen”

http://drpaulose.com/spirituality/ostrich-mentality

Why should I monitor?

Your business needs to stay running

http://techbusinessintelligence.blogspot.com/2016/02/upgrade-of-production-bi-server.html

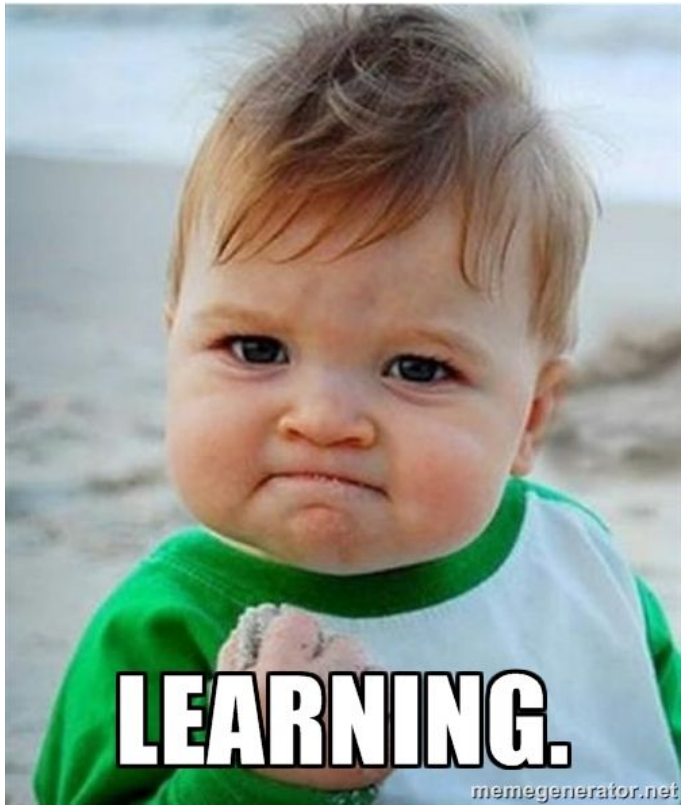
Why should I monitor?

Understand system/application behavior

Why should I monitor?

Capacity planning, autoscaling, hardware configuration, performance troubleshooting

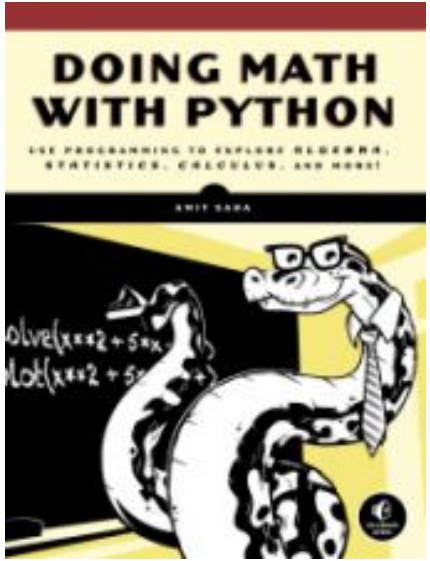
My monitoring journey - Stage 2



About me

Currently DevOps Engineer at RateSetter Australia

Author of “Doing Math with Python” and various technical articles



Fedora Scientific creator/maintainer

https://bit.ly/python-monitoring

Metric

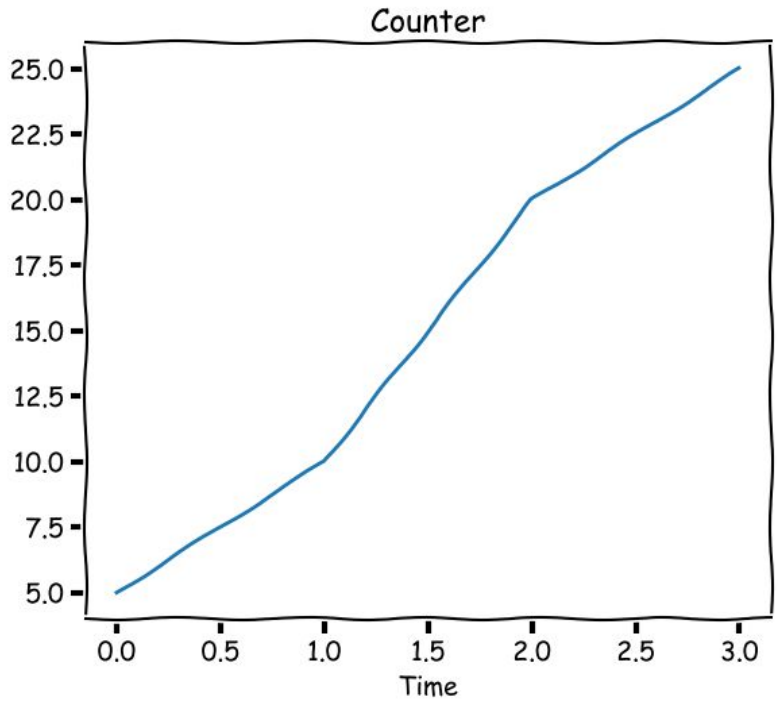
The measure/value of a quantity at a given point of time

Source: [matplotlib examples showcase]

Metric Types

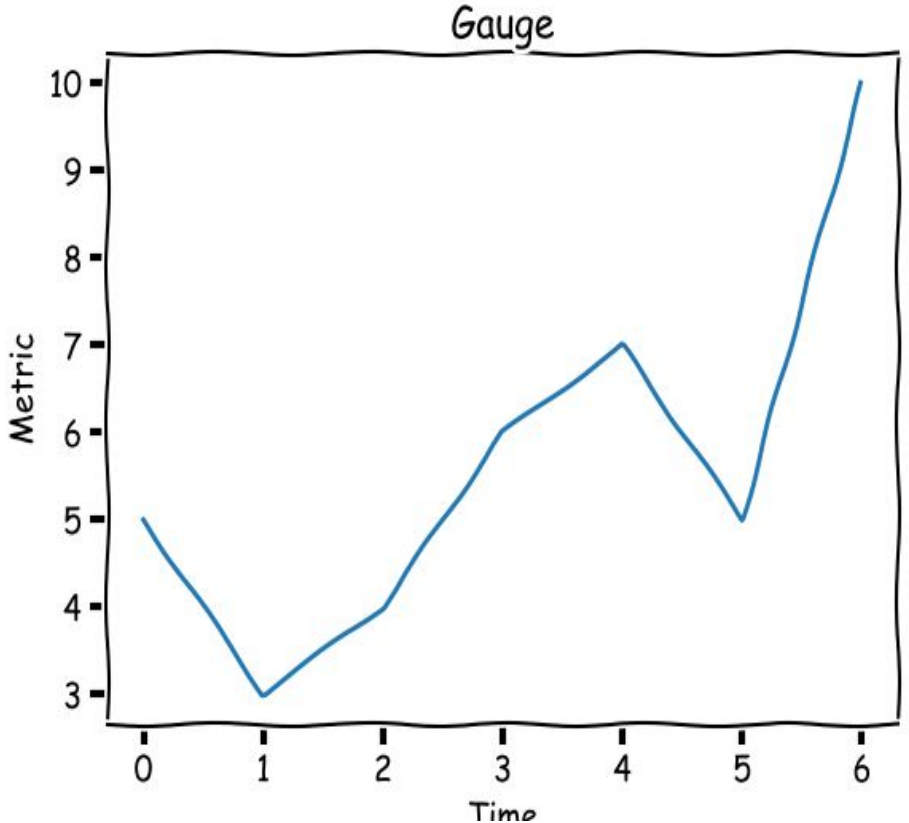
Counter

A metric whose value increases during the lifetime of a process/system



Gauge

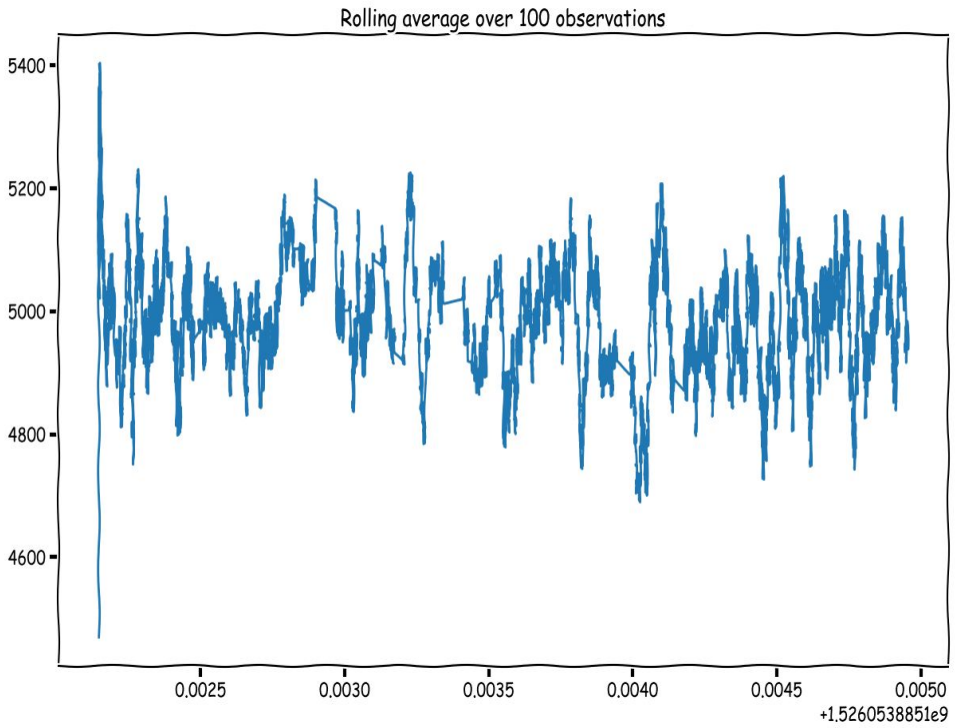
A metric whose value can go up or down arbitrarily - usually with a floor and ceiling



Histogram/Timer

A metric to track

observations



Source Code Walkthrough (Demo 1)

Demo 1: What did we see?

Using flask *middleware* to calculate/report metrics

Demo 1: What did we see?

Lots of metrics generated, hence we need to *summarize* the data

Demo 1: What did we see?

No characteristics in the metrics - which endpoint? What response status?

Statistics

Mean and Median

Mean

Mean of 5, 8, 3 = (5+8+3)/3 = 5.33....

Median: a better average

Median of 5, 8, 3 is 5

Percentile and Upper X

The *percentile* is a measure which gives us a measure below which a certain, k percentage of the numbers lie.

Most monitoring systems refer to it as *upper_X* where *X* is the percentile.

Quantile

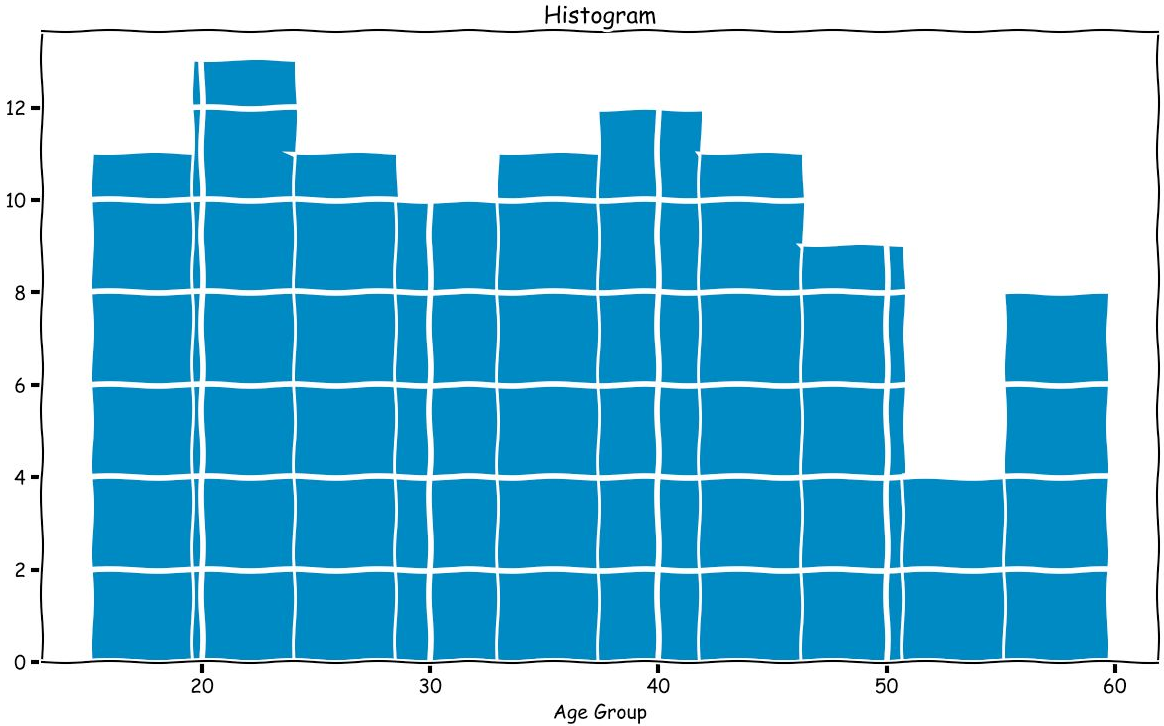
A *quantile* gives us another way to find a number at a specific

position in a set of numbers

0.xy quantile => xy percentile

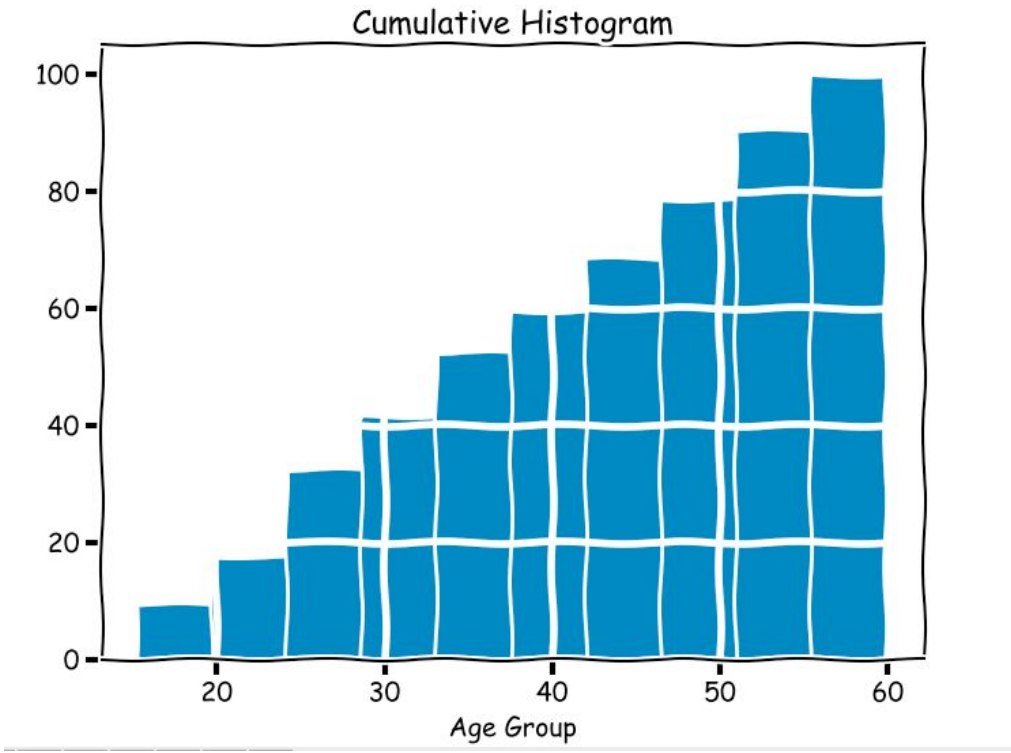
The (real) Histogram

Groups data into *buckets*



Cumulative Histogram

Groups data into buckets, but each bucket also contains the previous bucket members



Adding characteristics to metrics

Why do we need characteristics?

What was the latency of a specific HTTP endpoint?

Why do we need characteristics?

What was the latency for a specific instance of the application?

Why do we need characteristics?

What were the number of HTTP 500s for a specific endpoint?

Examples of metric characteristics

System identifier (IP address, Container ID, AWS instance ID..)

HTTP Endpoint name

HTTP Method

HTTP response status

RPC Method Name

..

Source Code Walkthrough (Demo 2)

Demo 2: What did we see?

We saw how we can add characteristics to metrics

Demo 2: What did we see?

We have a multi-column CSV file - what does it look similar to?


gettyimages
MOHD RASFAN
594359398

Grouping, Aggregation using Pandas (Demo 2)

Read the CSV file

```
import pandas as pd
metrics = pd.read_csv('./src/metrics.csv', index_col=0)
```

Metrics as *pandas* DataFrame

The *timestamp* is the *index*

timestamp	app_prefix	node_id	http_endpoint	http_method	http_status	latency
1522219178	webapp1	10.1.3.4	test	GET	200	0.109911
1522219178	webapp1	10.1.3.4	test	GET	200	0.054598
1522219178	webapp1	10.1.3.4	test	GET	200	0.051498
1522219178	webapp1	10.0.1.1	test	GET	400	0.059128
1522219178	webapp1	10.1.3.4	test	GET	200	0.053644

Each metric characteristic is a *column*

Grouping

Metric

```
In [8]: metrics.groupby(['node_id', 'http_status']).latency.aggregate(np.percentile, 99.999)
```

```
Out[8]: node_id    http_status
        10.0.1.1   200            0.068661
                   400            0.057935
                   500            1.330025
        10.1.3.4   200            0.220268
                   400            0.162116
                   500            1.419507
```

Aggregation

Summary: Monitoring your applications

1. Your application calculates the metrics (*Middleware)*

2. A monitoring system stores these (*CSV files*)

3. Human/machine queries the monitoring system (*Pandas*)

Integrating monitoring in your applications (for real)

What application metrics should I calculate?

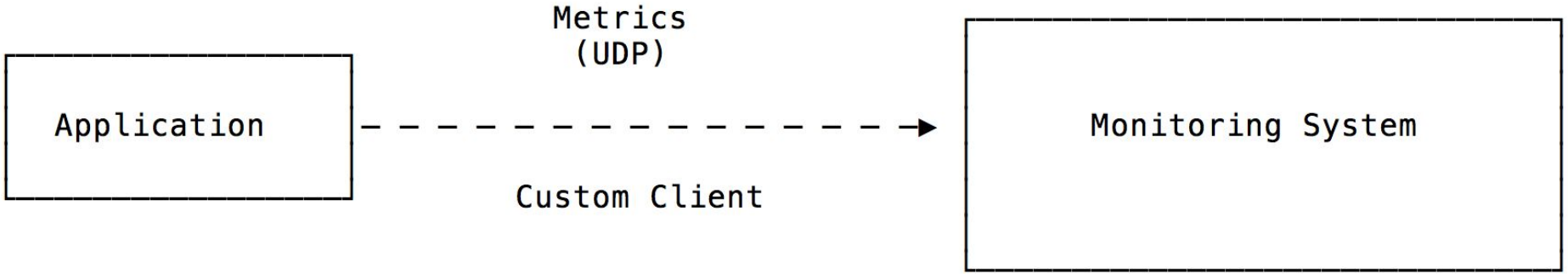
Network servers: Request latency, Queue size (if any), Exceptions, Waiting time, Worker usage

Batch jobs: Last run, latency

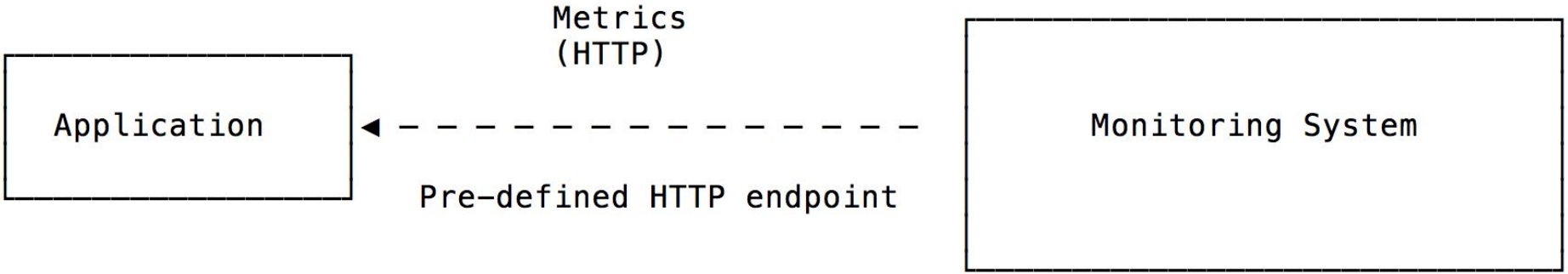
Consumers: Latency

Recommended: The four golden signals

Application metrics -> Monitoring System



Application metrics <- Monitoring System



Monitoring Systems

Self hosted/maintained - statsd, prometheus

Third party SaaS

- https://www.outlyer.com/features/
- https://docs.datadoghq.com/developers/dogstatsd/
- https://honeycomb.io/docs/

Please!
Say *NO* to DIY monitoring system

statsd

Key statsd concepts

Application push metrics to statsd server (usually over UDP)

A metric *key* is of the form
webapp1.<key1>.<key2>...<keyN>.latency

Each *dot* separated part of the key is a *metric characteristic/dimension*

Key statsd concepts: example keys

```
# request.<path>.<method>.http_<status_code>.latency
REQUEST_LATENCY_METRIC_KEY_PATTERN = 'instance1.{0}.{1}.http_{2}.latency'

# request.<path>.<method>.http_<status_code>
REQUEST_COUNT_METRIC_KEY_PATTERN = 'instance1.request.{0}.{1}.http_{2}.count'
```

ip-10-11-12-54.*webapp1.test_endpoint.get.http_200.latency* is a valid statsd metric name

Key statsd concepts: Pushing metrics

Key statsd concepts: Grouping and Aggregation

```
scaleToSeconds(sumSeries(stats.timers.webapp1.*.*.*.http_200.latency.count),60))
```

```
groupByNode(stats.timers.webapp1.*.*.*.upper_90, 3, 'maxSeries')
```

Prometheus

Key prometheus concepts

Application exposes a HTTP endpoint for prometheus to scrape - usually, /metrics

```
@app.route('/metrics')
def metrics():
    return Response(prometheus_client.generate_latest(), mimetype=CONTENT_TYPE_LATEST)
```

Key prometheus concepts

Each metric can be associated with multiple *labels* which are the characteristics of the metric

Internally, each *metric* and *label* combination is a separate metric

Key prometheus concepts: Metric definition

```
REQUEST_COUNT = Counter(
    'request_count_total', 'App Request Count',
    ['app_name', 'method', 'endpoint', 'http_status']
)
REQUEST_LATENCY = Histogram('request_latency_seconds', 'Request latency',
    ['app_name', 'endpoint']
)
```

Key prometheus concepts: Metric updates

```
REQUEST_LATENCY.labels('webapp', request.path).observe(resp_time)

REQUEST_COUNT.labels('webapp', request.method, request.path, response.status_code).inc()
```

Key prometheus concepts: Grouping and Aggregation

```
max(request_latency_ms{label1="value1", label2="value2", quantile="0.99"}) by  (label2)
```

Statsd or Prometheus?

Native prometheus exporting in Python has certain *gotchas*

I recommend using the statsd exporter

I have written about this topic elsewhere

Summary

http://techbusinessintelligence.blogspot.com/2016/02/upgrade-of-production-bi-server.html

We should talk about them, learn as we go - may be from *first principles*

And once we have learned enough ...

My monitoring journey - Stage 3

Learn to do it right!

https://bit.ly/python-monitoring

Feedback? Questions?

@echorand
https://echorand.me

amitsaha.in@gmail.com

Thanks

You for choosing my talk!

PyCon committee for the opportunity!

My previous employer and team at Freelancer.com

My employer - RateSetter Australia for funding my conference visit!

Sydney Python Meetup group for the opportunity to deliver a version of this talk

Nick Coghlan for feedback and lending a travel adapter :)