BUSINESS DASHBOARDS

using Bonobo, Airflow and Grafana

makersquad.fr

Romain Dorgueil

romain@makersquad.fr

Building software from Zero to Market

rdorgueil

makersquad.fr

Content

DISCLAIMERS

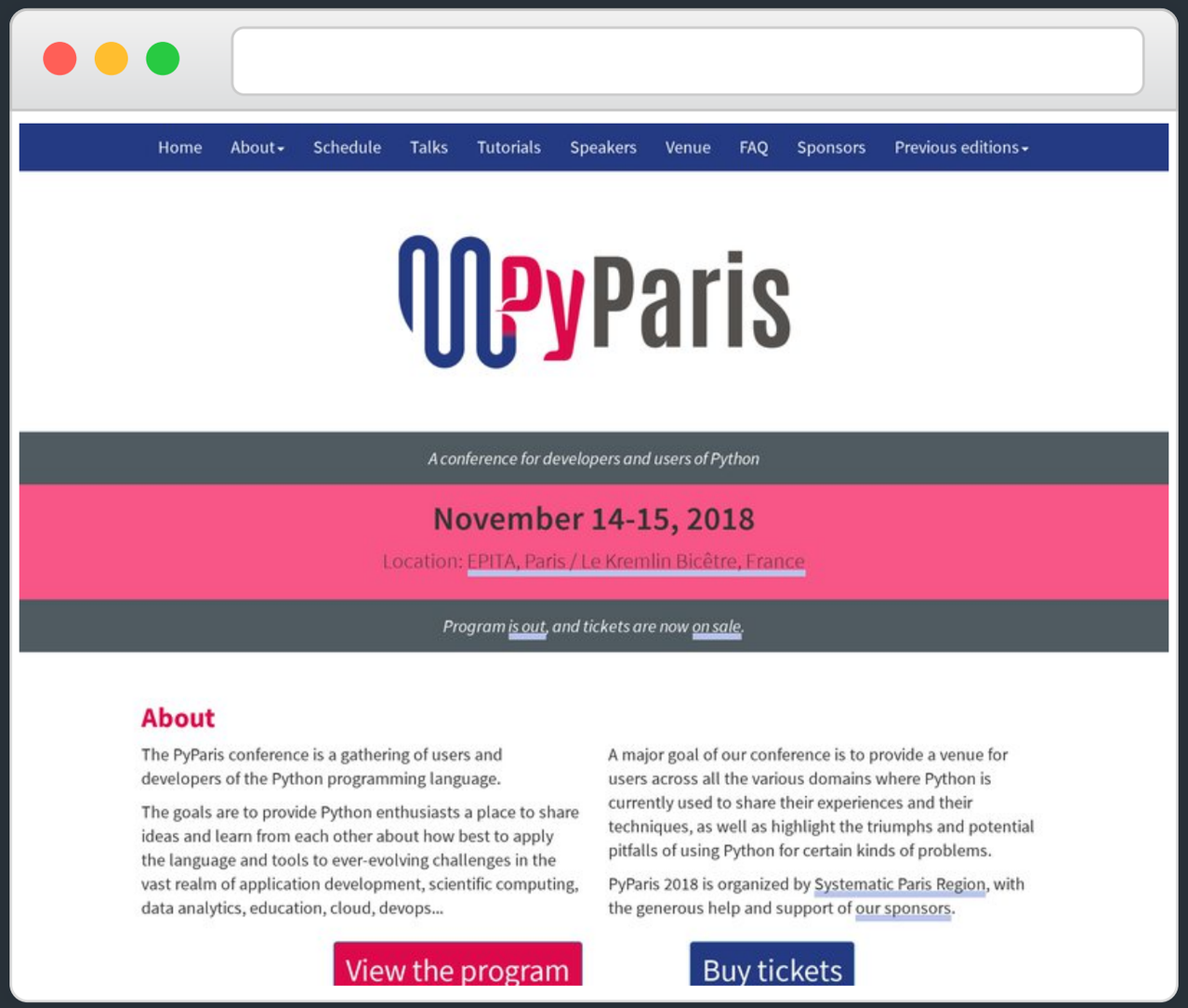
If you build a product, **do your own research**.
Take time to learn and understand tools you consider.

I don't know nothing, and **I recommend nothing**.
I assume things and change my mind when I hit a wall.

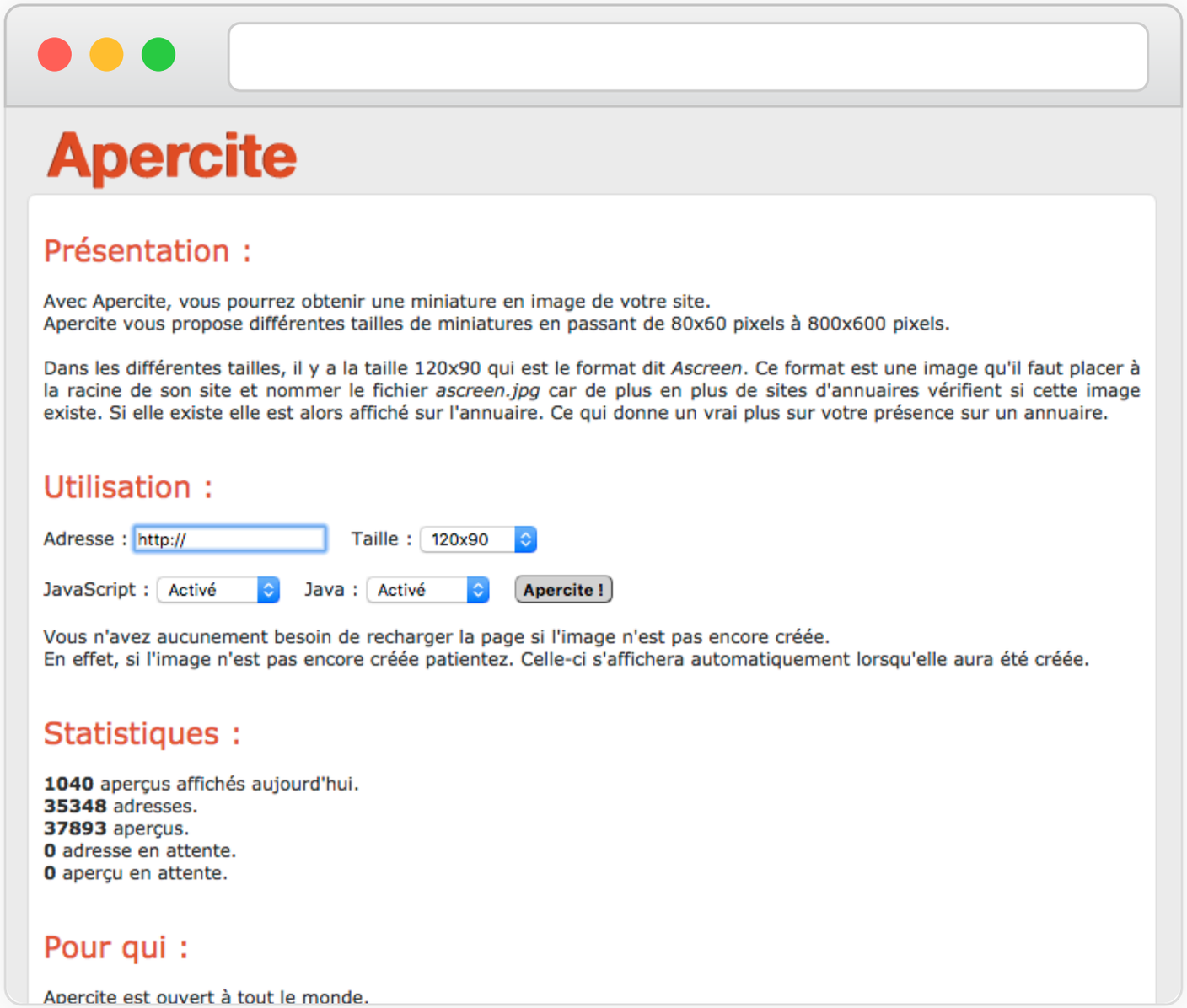
I'm the creator and main developper of Bonobo ETL.
I'll try to be objective, but **there is a bias here**.

PRODUCT

GET https://aprc.it/api/800x600/http://pyparis.org/



January 2009



Apercite

Présentation :

Avec Apercite, vous pourrez obtenir une miniature en image de votre site.
Apercite vous propose différentes tailles de miniatures en passant de 80x60 pixels à 800x600 pixels.

Dans les différentes tailles, il y a la taille 120x90 qui est le format dit *Ascreen*. Ce format est une image qu'il faut placer à la racine de son site et nommer le fichier *ascreen.jpg* car de plus en plus de sites d'annuaires vérifient si cette image existe. Si elle existe elle est alors affiché sur l'annuaire. Ce qui donne un vrai plus sur votre présence sur un annuaire.

Utilisation :

Adresse : http:// Taille : 120x90

JavaScript : Activé Java : Activé Apercite !

Vous n'avez aucunement besoin de recharger la page si l'image n'est pas encore créée.
En effet, si l'image n'est pas encore créée patientez. Celle-ci s'affichera automatiquement lorsqu'elle aura été créée.

Statistiques :

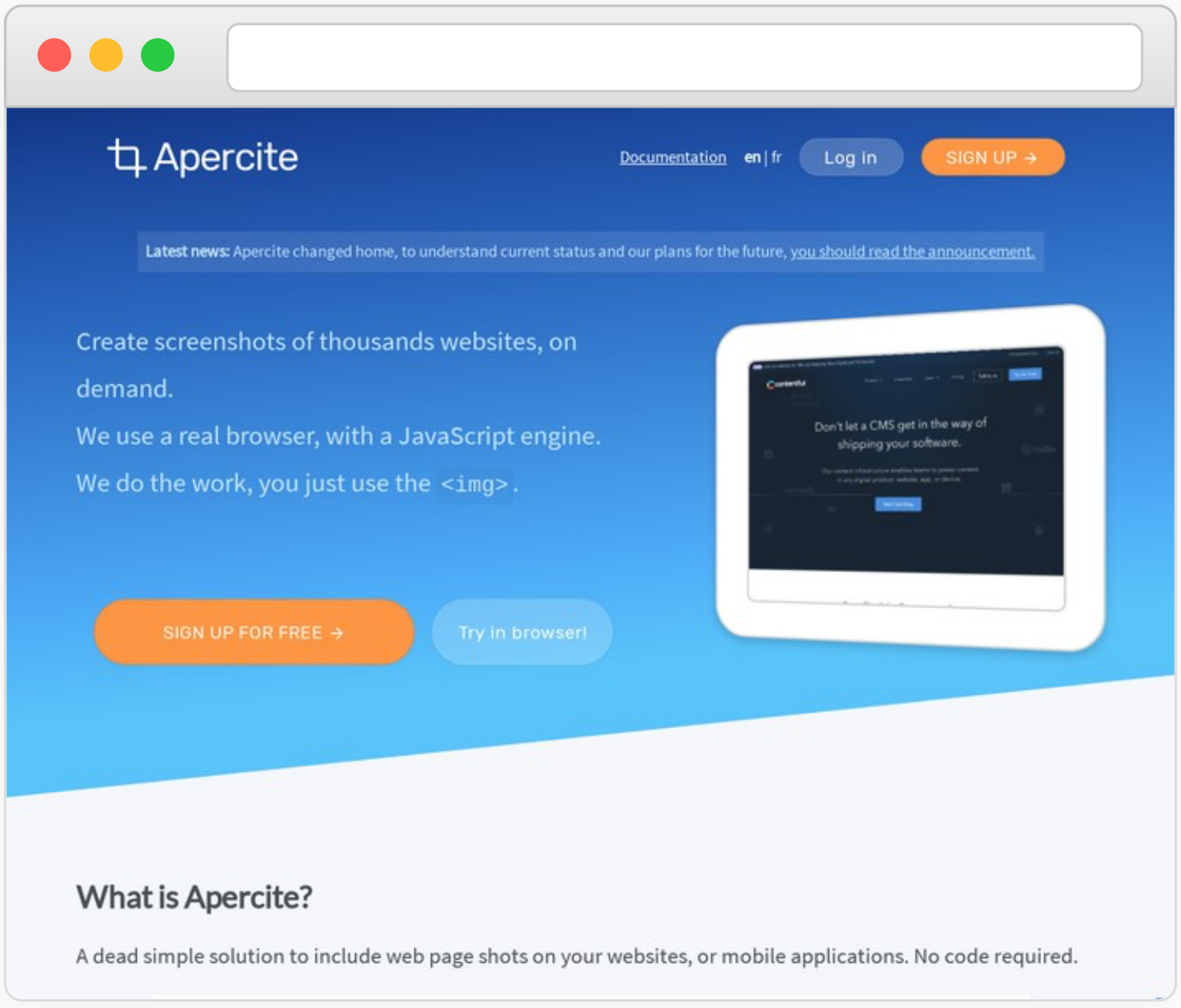
1040 aperçus affichés aujourd'hui.
35348 adresses.
37893 aperçus.
0 adresse en attente.
0 aperçu en attente.

Pour qui :

Apercite est ouvert à tout le monde.


timedelta(years=9)

April 2018


Apercite
Documentation
en | fr
Log in
SIGN UP →
Latest news: Apercite changed home, to understand current status and our plans for the future, you should read the announcement.
Create screenshots of thousands websites, on demand.
We use a real browser, with a JavaScript engine.
We do the work, you just use the <img>.
SIGN UP FOR FREE →
Try in browser!
Don't let a CMS get in the way of shipping your software.
What is Apercite?
A dead simple solution to include web page shots on your websites, or mobile applications. No code required.

under the hood ...

AMQP/RabbitMQ

HTTP/HTTPS

SQL/Storage

Load Balancer
TCP / L4

Reverse Proxy
HTTP2 / L7

HTTP

Website
(django)

APIServer
(tornado)

Local
Cache

"MISS"

Redis

Database
(postgres)

Object
Storage

"Events" Message Queue

Janitor
(asyncio)

"CRAWL"

"CREATED"

Spider
(asyncio)

"Orders" Message Queue

AMQP

HTTP/HTTPS

SQL/Storage

Load Balancer
TCP / L4

Reverse Proxy
HTTP2 / L7

HTTP

MANAGEMENT SERVICES

- Weblate
- Grafana
- AlertManager
- Prometheus

EXTERNAL SERVICES

- Google Analytics
- Mailgun
- Stripe
- Drift
- Slack
- MixMax
- Sentry
- ...

Database
(postgres)

+

PROMETHEUS EXPORTERS

- Prometheus
- PostgreSQL
- Kubernetes
- NGINX + VTS
- RabbitMQ
- Apercite
- Redis
- ...

Grafana - Apercite :: Pods

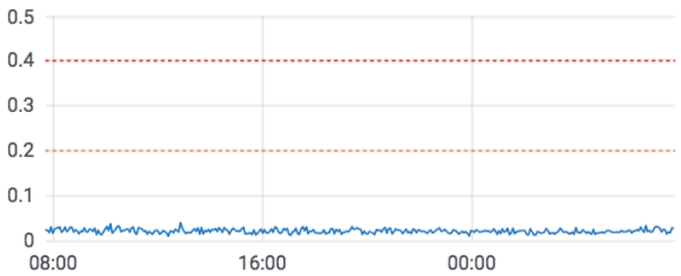
Romain

Sécurisé

Apercite :: Pods

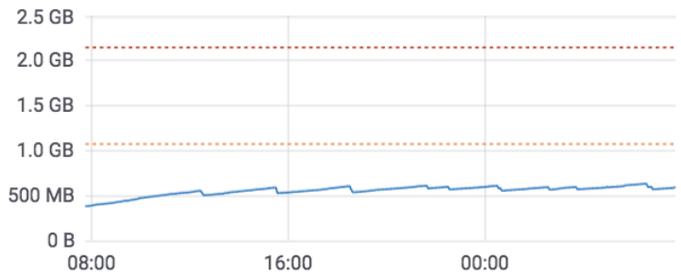
Last 24 hours

Gateway (CPU)



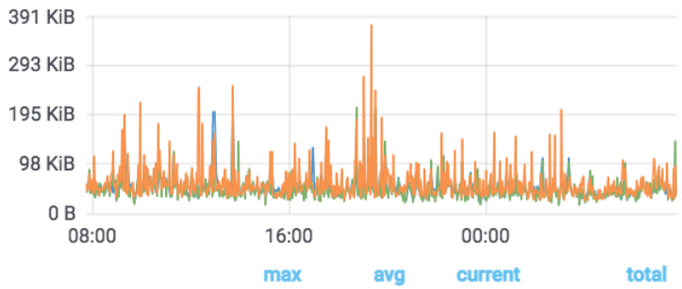
	min	max	avg	current
nginx	0.0098	0.0396	0.0210	0.0259

Gateway (Memory)



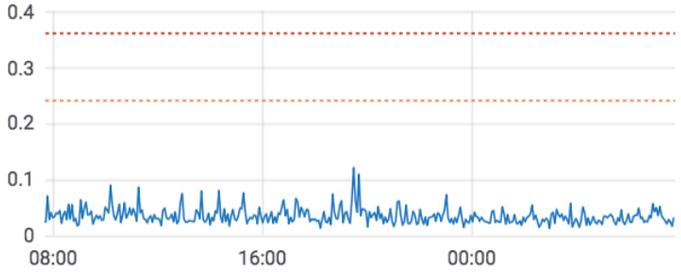
	min	max	avg	current
nginx	377 MB	633 MB	553 MB	593 MB

Network I/O



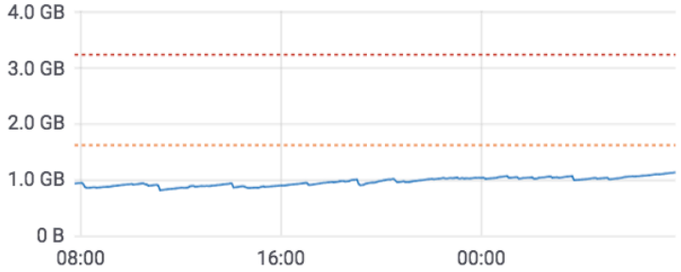
	max	avg	current	total
in	213 KiB	52 KiB	120 KiB	18.339 MiB
out	213 KiB	49 KiB	35 KiB	34.758 MiB

APIServer (CPU)



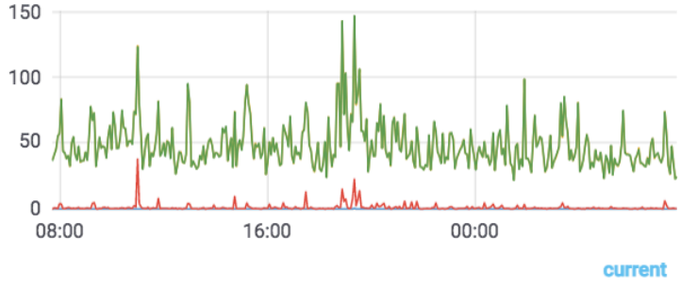
	min	max	avg	current
apiserver	0.0132	0.1210	0.0349	0.0322

APIServer (Memory)



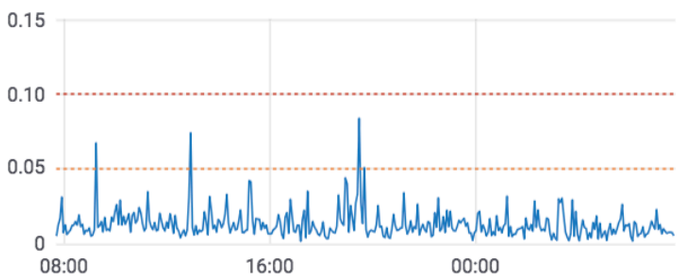
	min	max	avg	current
apiserver	808 MB	1.127 GB	962 MB	1.127 GB

AMQP Hits / Minute



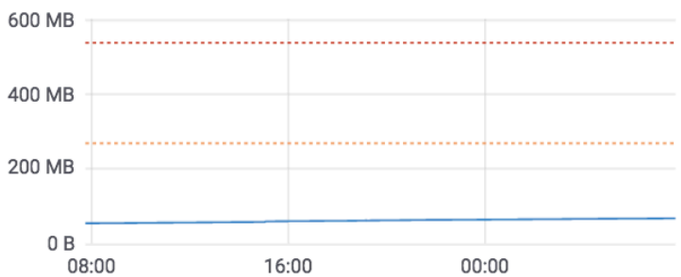
	current
hit:ready	0
hit:published	25

Janitor (CPU)



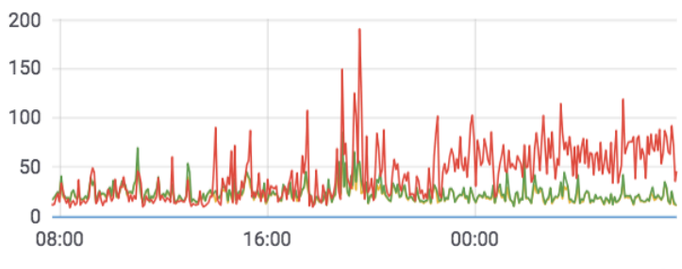
	min	max	avg	current
janitor	0.0018	0.0837	0.0131	0.0053

Janitor (Memory)



	min	max	avg	current
janitor	55 MB	68 MB	62 MB	68 MB

AMQP Crawls / Minute



	current
crawl:ready	0
crawl:published	11

Grafana - Apercite :: Pods

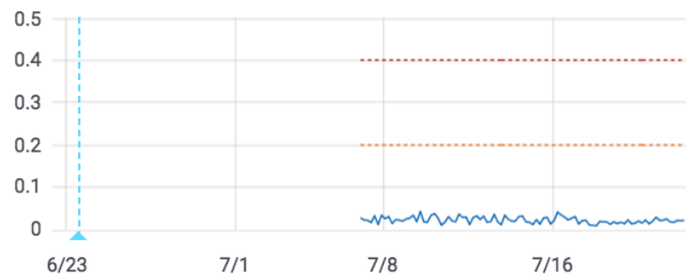
Romain

Sécurisé

Apercite :: Pods

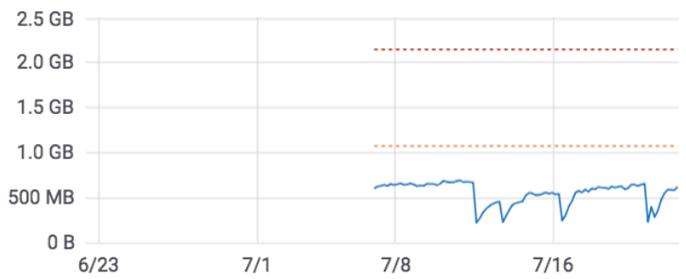
Last 30 days

Gateway (CPU)



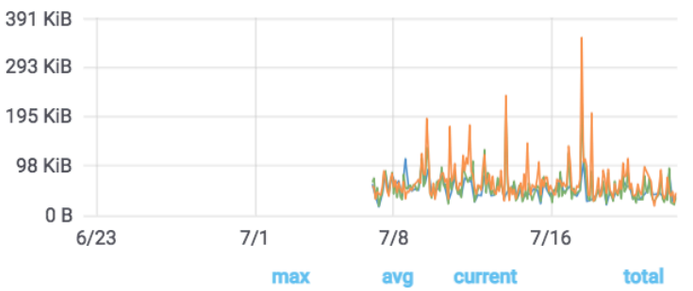
	min	max	avg	current
nginx	0.009	0.043	0.023	0.023

Gateway (Memory)



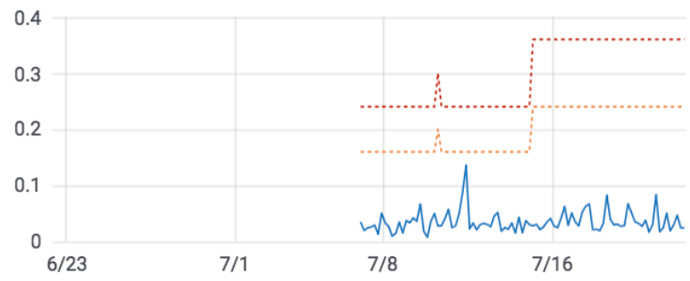
	min	max	avg	current
nginx	221 MB	691 MB	551 MB	621 MB

Network I/O



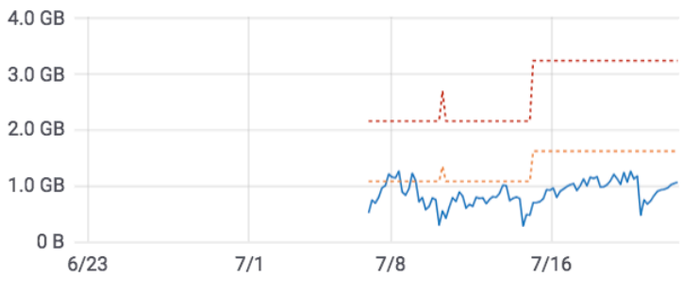
	max	avg	current	total
in	112 KiB	52 KiB	40 KiB	4.713 MiB
out	224 KiB	57 KiB	36 KiB	10.214 MiB

APIServer (CPU)



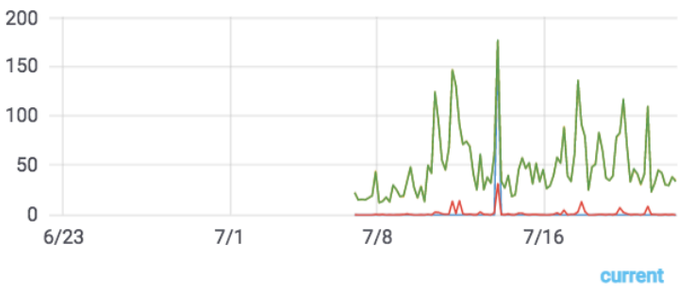
	min	max	avg	current
apiserver	0.0079	0.1360	0.0355	0.0243

APIServer (Memory)



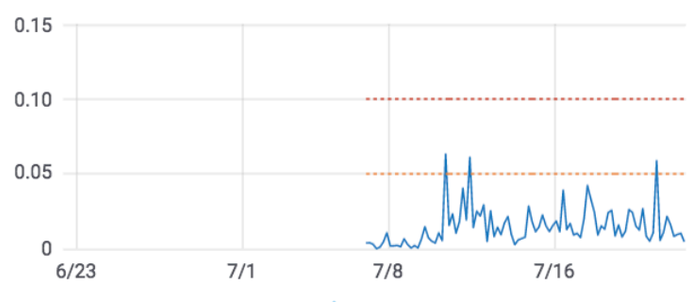
	min	max	avg	current
apiserver	280 MB	1.255 GB	863 MB	1.056 GB

AMQP Hits / Minute



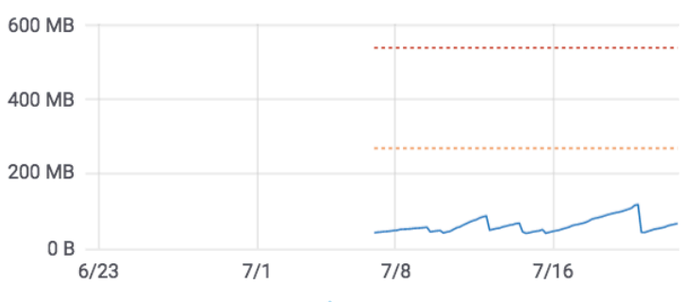
	current
hit:ready	0
hit:published	35

Janitor (CPU)



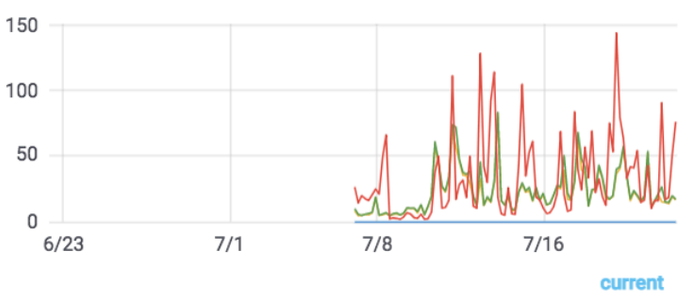
	min	max	avg	current
janitor	0.0001	0.0631	0.0147	0.0047

Janitor (Memory)



	min	max	avg	current
janitor	41 MB	118 MB	62 MB	67 MB

AMQP Crawls / Minute



	current
crawl:ready	0
crawl:published	17

PLAN

« What gets measured gets improved. »

— *Peter Drucker*

Planning

- Take your time to **choose metrics** wisely.
- **Cause** vs **Effect**.
- Less is More.
- **One at a time.** Or one by team.
- There is **not one answer** to this question.

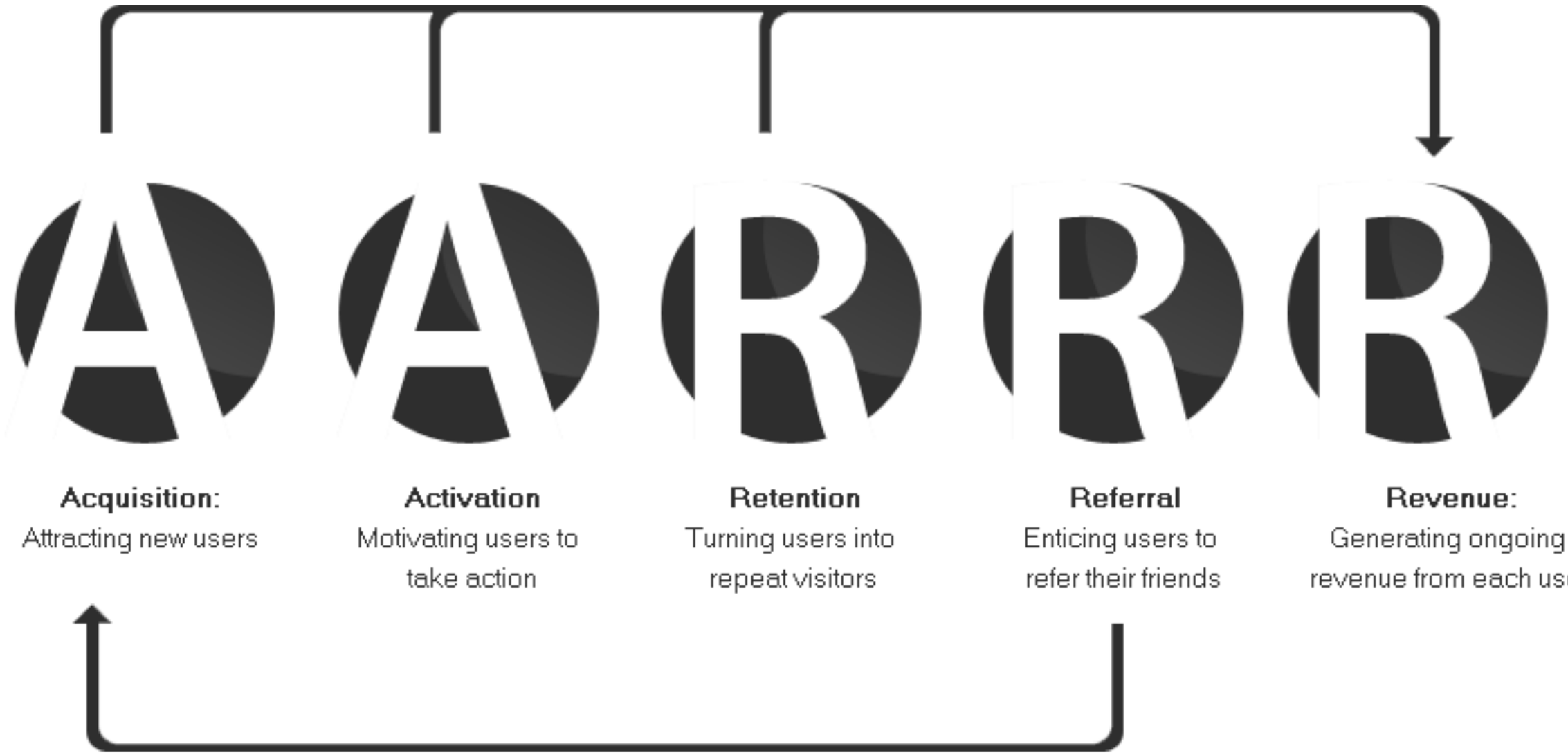
Vanity metrics will

waste your time

Planning

- Start with a **framework**.
- You may build your own, later.

Pirate Metrics



Acquisition:
Attracting new users

Activation
Motivating users to take action

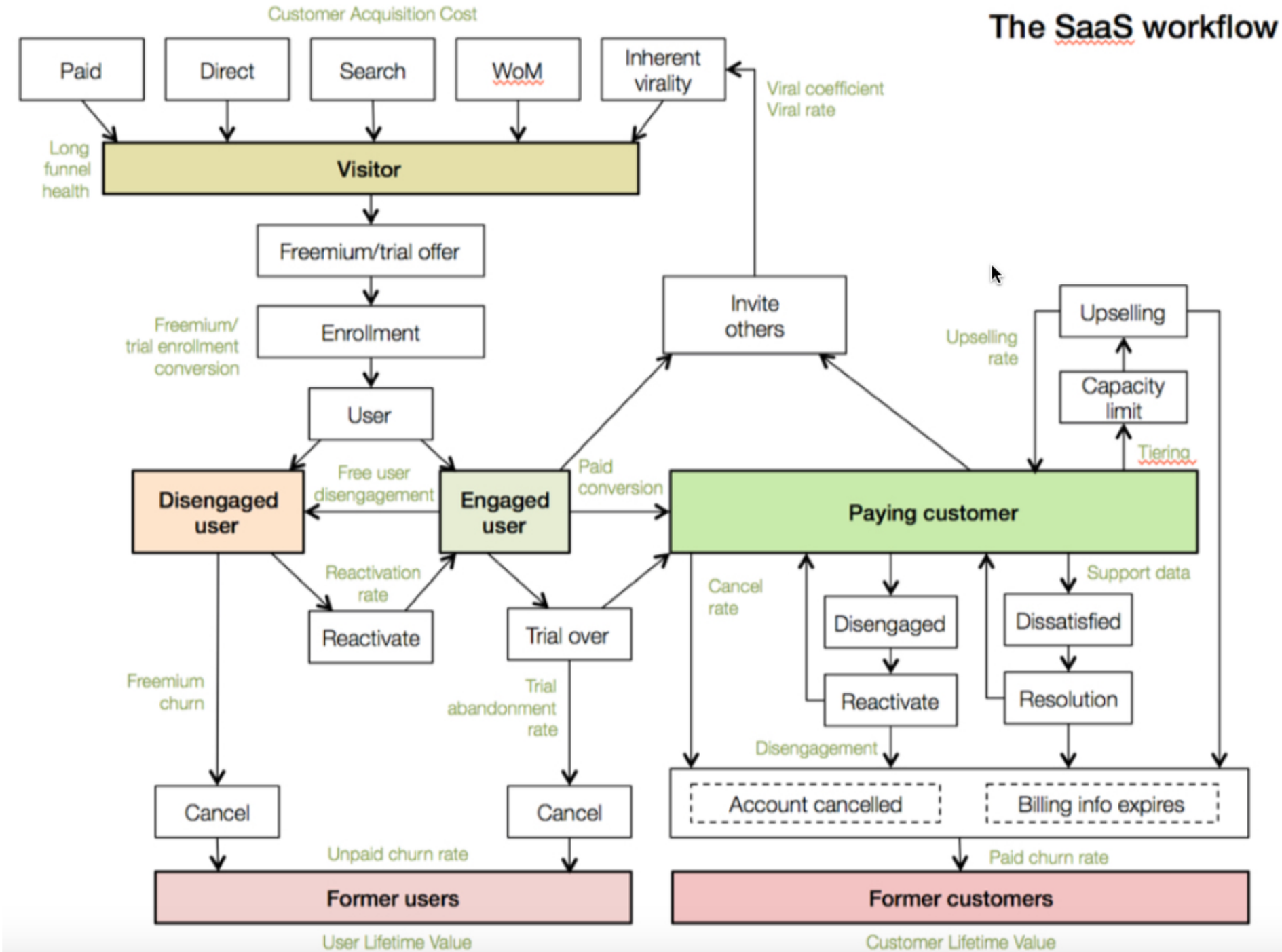
Retention
Turning users into repeat visitors

Referral
Enticing users to refer their friends

Revenue:
Generating ongoing revenue from each user

Lean Analytics

(book by Alistair Croll & Benjamin Yoskovitz)



Plan A

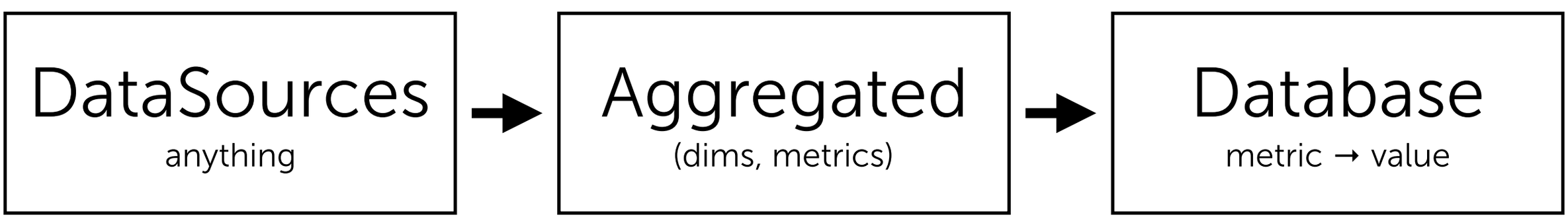

A
A
B
R
R
quisition:
Attra g new users
Activation
Motivating users to take action
Rete on
Turnin ers into re t visitors
Referral
Enticing users to refer their friends
Revenue:
Generating ongoing revenue from each user

Plan A

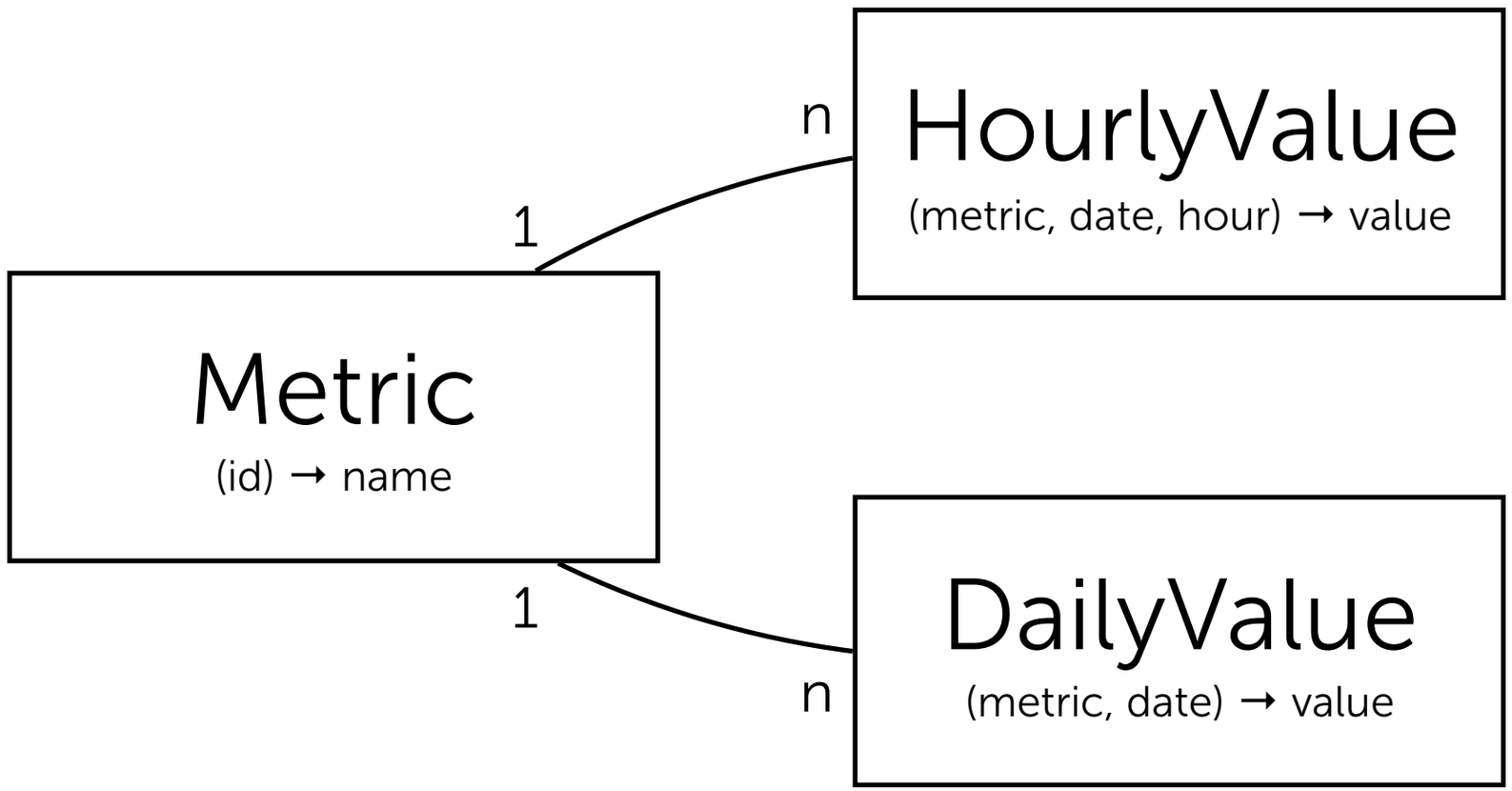
- What **business**? **Software as a Service**
- What **stage**? **Empathy / Stickyness**
- What **metric** matters?
  - Rate from acquisition to activation.
  - QOS (both for display and measure improvements).

IMPLEMENT

Idea


DataSources
anything
Aggregated
(dims, metrics)
Database
metric → value

Model


HourlyValue
(metric, date, hour) → value
n
1
Metric
(id) → name
1
n
DailyValue
(metric, date) → value

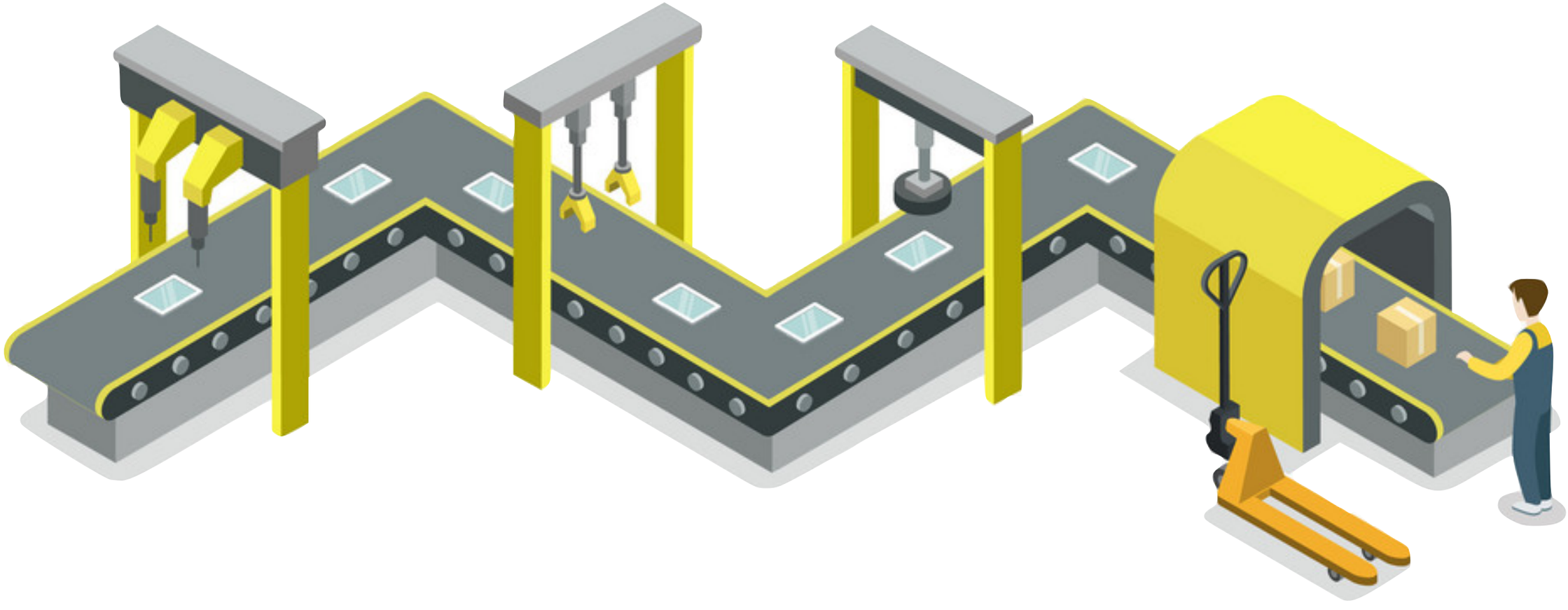
Quick to write.

Not the best.

Keywords to read more: Star and Snowflake Schemas

Bonobo

Extract Transform Load



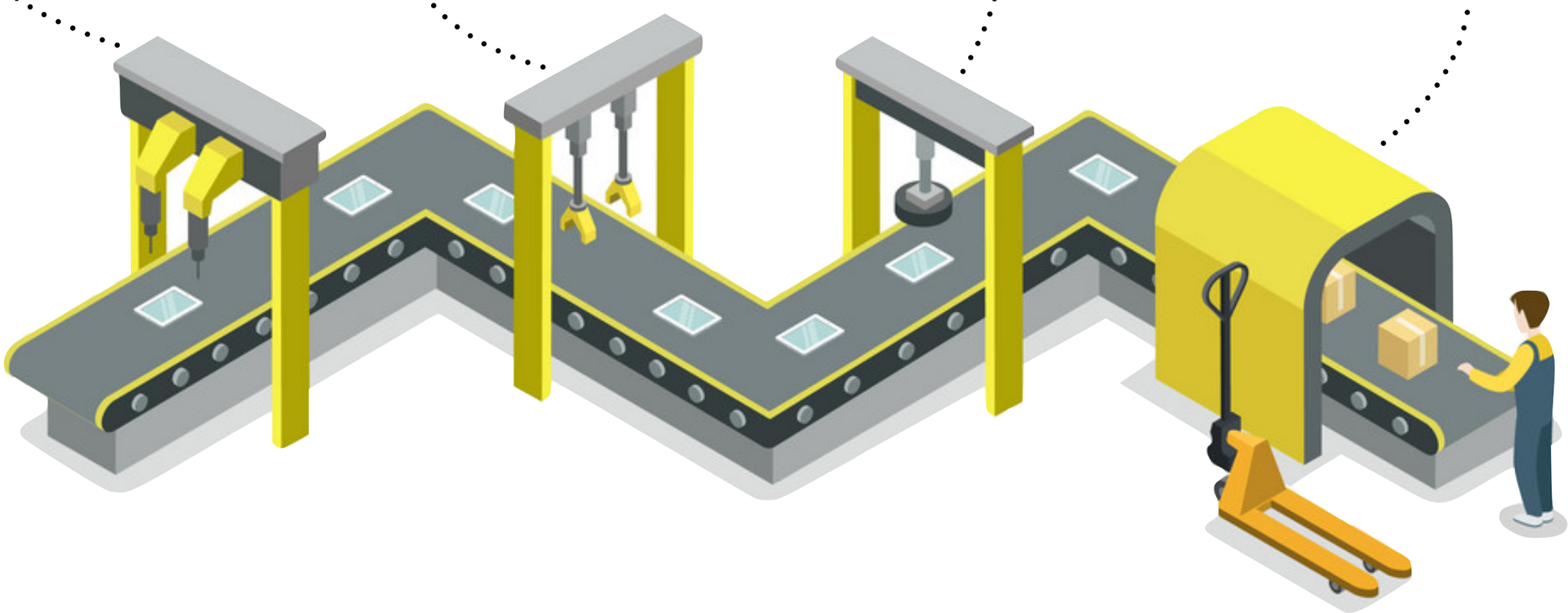
Bonobo

```
Select('''
  SELECT *
  FROM …
  WHERE …
''')
```

```
def qualify(row):
  yield (
    row,
    'active' if …
    else 'inactive'
  )
```

```
Join('''
  SELECT count(*)
  FROM …
  WHERE uid = %(id)s
''')
```

```
def report(row):
  send_email(
    render(
      'email.html', row
    )
  )
```



Bonobo

- Independent **threads**.
- Data is passed **first in, first out**.
- Supports any kind of **directed acyclic graphs**.
- Standard Python **callable and iterators**.
- Getting started still fits in here (ok, barely)

```
$ pip install bonobo
$ bonobo init somejob.py
$ python somejob.py
```

Let’s write our jobs.

Extract

… counts from website's database

```
from bonobo.config import use_context, Service
from bonobo_sqlalchemy.readers import Select

@use_context
class ObjectCountsReader(Select):
    engine = Service('website.engine')
    query = '''
    SELECT count(%(0)s.id) AS cnt
    FROM %(0)s
'''
    output_fields = ['dims', 'metrics']

    def formatter(self, input_row, row):
        now = datetime.datetime.now()
        return ({
            'date': now.date(),
            'hour': now.hour,
        }, {
            'objects.{}.count'.format(input_row[1]): row['cnt']
        })
```

Extract

… counts from website's database

```
TABLES_METRICS = {
    AsIs('apercite_account_user'): 'users',
    AsIs('apercite_account_userprofile'): 'profiles',
    AsIs('apercite_account_apikey'): 'apikeys',
}

def get_readers():
    return [
        TABLES_METRICS.items(),
        ObjectCountsReader(),
    ]
```

Normalize

All data should look the same

```
bonobo.SetFields(['dims', 'metrics'])
```

Load

```
class AnalyticsWriter(InsertOrUpdate):
    dims = Option(required=True)
    filter = Option(required=True)

    @property
    def discriminant(self):
        return ('metric_id', *self.dims)

    def get_or_create_metrics(self, context, connection, metrics):
        …

def __call__(self, connection, table, buffer, context, row, engine):
    dims, metrics = row

    if not self.filter(dims, metrics): return

    # Get database rows for metric objects.
    db_metrics_ids = self.get_or_create_metrics(context, connection, metrics)

    # Insert or update values.
    for metric, value in metrics.items():
        yield from self._put(table, connection, buffer, {
            'metric_id': db_metrics_ids[metric],
            **{dim: dims[dim] for dim in self.dims},
            'value': value,
        })
```

Compose

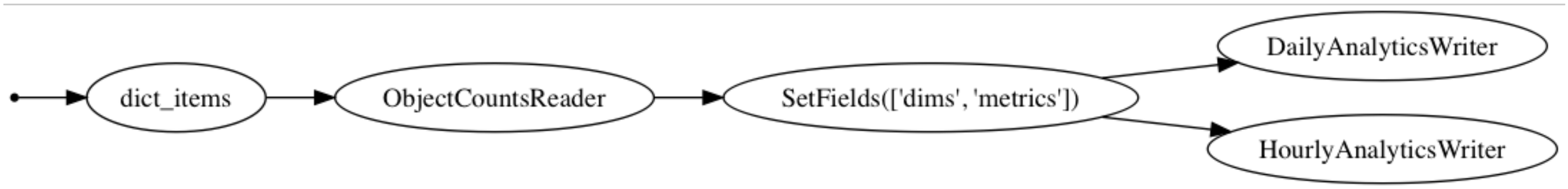
```
def get_graph():
    normalize = bonobo.SetFields(['dims', 'metrics'])
    graph = bonobo.Graph(*get_readers(), normalize)

    graph.add_chain(
        AnalyticsWriter(
            table_name=HourlyValue.__tablename__,
            dims=('date', 'hour',),
            filter=lambda dims, metrics: 'hour' in dims,
            name='Hourly',
        ), _input=normalize
    )
    graph.add_chain(
        AnalyticsWriter(
            table_name=DailyValue.__tablename__,
            dims=('date',),
            filter=lambda dims, metrics: 'hour' not in dims,
            name='Daily',
        ),
        _input=normalize
    )

    return graph
```

Inspect

```
bonobo inspect --graph job.py | dot -o graph.png -T png
```



Configure

```
def get_services():
    return {
        'sqlalchemy.engine': EventsDatabase().create_engine(),
        'website.engine': WebsiteDatabase().create_engine(),
    }
```

Run

```
$ python -m apercite.analytics read objects --write
- dict_items in=1 out=3 [done]
- ObjectCountsReader in=3 out=3 [done]
- SetFields(['dims', 'metrics']) in=3 out=3 [done]
- HourlyAnalyticsWriter in=3 out=3 [done]
- DailyAnalyticsWriter in=3 [done]
```

Got it.

Let’s add readers.

We’ll run through, you’ll have the code.

Google Analytics

```
@use('google_analytics')
def read_analytics(google_analytics):
    reports = google_analytics.reports().batchGet(
        body={…}
    ).execute().get('reports', [])

    for report in reports:
        dimensions = report['columnHeader']['dimensions']
        metrics = report[‘columnHeader'][‘metricHeader'][‘metricHeaderEntries']
        rows = report['data']['rows']

    for row in rows:
        dim_values = zip(dimensions, row['dimensions'])
        yield (
            {
                GOOGLE_ANALYTICS_DIMENSIONS.get(dim, [dim])[0]:
                GOOGLE_ANALYTICS_DIMENSIONS.get(dim, [None, IDENTITY])[1](val)
                for dim, val in dim_values
            },
            {
                GOOGLE_ANALYTICS_METRICS.get(metric['name'], metric['name']):
                GOOGLE_ANALYTICS_TYPES[metric['type']](value)
                for metric, value in zip(metrics, row['metrics'][0]['values'])
            },
        )
```

Prometheus

```
class PrometheusReader(Configurable):
    http = Service('http')
    endpoint = 'http://{}:{}/api/v1'.format(PROMETHEUS_HOST, PROMETHEUS_PORT)
    queries = […]

    def __call__(self, *, http):
        start_at, end_at = self.get_timerange()
        for query in self.queries:
            for result in http.get(…).json().get('data', {}).get('result', []):
                metric = result.get('metric', {})
                for ts, val in result.get('values', []):
                    name = query.target.format(**metric)
                    _date, _hour = …
                   yield {
                        'date': _date,
                        'hour': _hour,
                    }, {
                        name: float(val)
                    }
```

Spider counts

```
class SpidersReader(Select):
    kwargs = Option()
    output_fields = ['row']

    @property
    def query(self):
        return '''
            SELECT spider.value AS name,
                   spider.created_at AS created_at,
                   spider_status.attributes AS attributes,
                   spider_status.created_at AS updated_at
            FROM
                spider
            JOIN …
            WHERE spider_status.created_at > %(now)s
            ORDER BY spider_status.created_at DESC
        '''

    def formatter(self, input_row, row):
        return (row, )
```

Spider counts

```
def spider_reducer(self, left, right):
    result = dict(left)
    result['spider.total'] += len(right.attributes)

    for worker in right.attributes:
        if 'stage' in worker:
            result['spider.active'] += 1
        else:
            result['spider.idle'] += 1

    return result
```

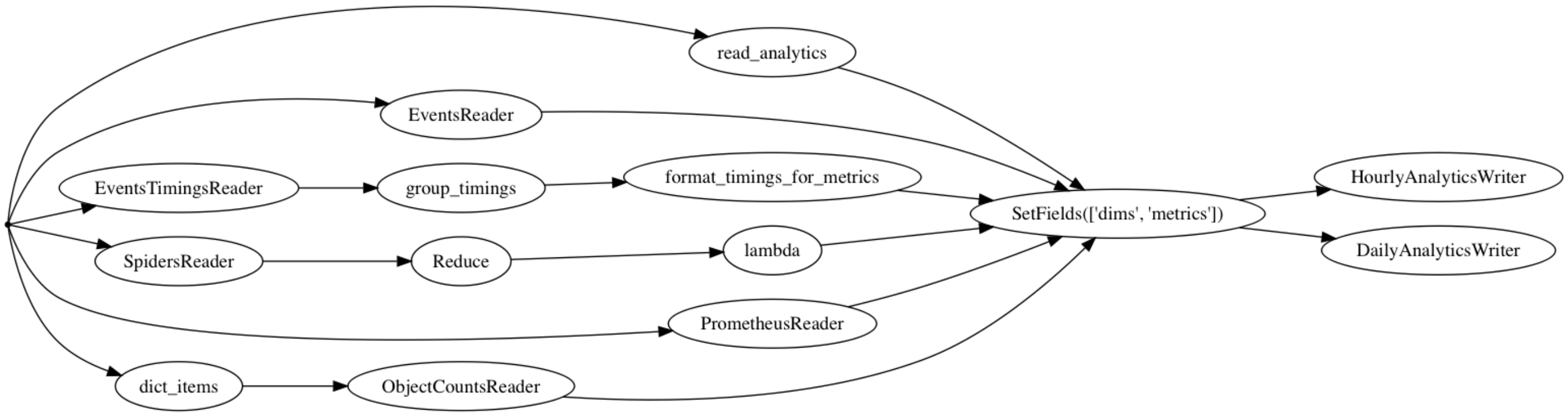
Spider counts

```
now = datetime.datetime.utcnow() - datetime.timedelta(minutes=30)

def get_readers():
    return (
        SpidersReader(kwargs={'now': now}),
        Reduce(spider_reducer, initializer={
            'spider.idle': 0,
            'spider.active': 0,
            'spider.total': 0,
        }),
        (lambda x: ({'date': now.date(), 'hour': now.hour}, x))
    )
```

etc.

Inspect



We can generate ETL graphs with all readers or only a few.

Run

```
$ python -m apercite.analytics read all --write

- read_analytics in=1 out=91 [done]
- EventsReader in=1 out=27 [done]
- EventsTimingsReader in=1 out=2039 [done]
- group_timings in=2039 out=24 [done]
- format_timings_for_metrics in=24 out=24 [done]
- SpidersReader in=1 out=1 [done]
- Reduce in=1 out=1 [done]
- <lambda> in=1 out=1 [done]
- PrometheusReader in=1 out=3274 [done]
- dict_items in=1 out=3 [done]
- ObjectCountsReader in=3 out=3 [done]
- SetFields(['dims', 'metrics']) in=3420 out=3420 [done]
- HourlyAnalyticsWriter in=3420 out=3562 [done]
- DailyAnalyticsWriter in=3420 out=182 [done]
```

Easy to **build**.

Easy to **add** or **replace** parts.

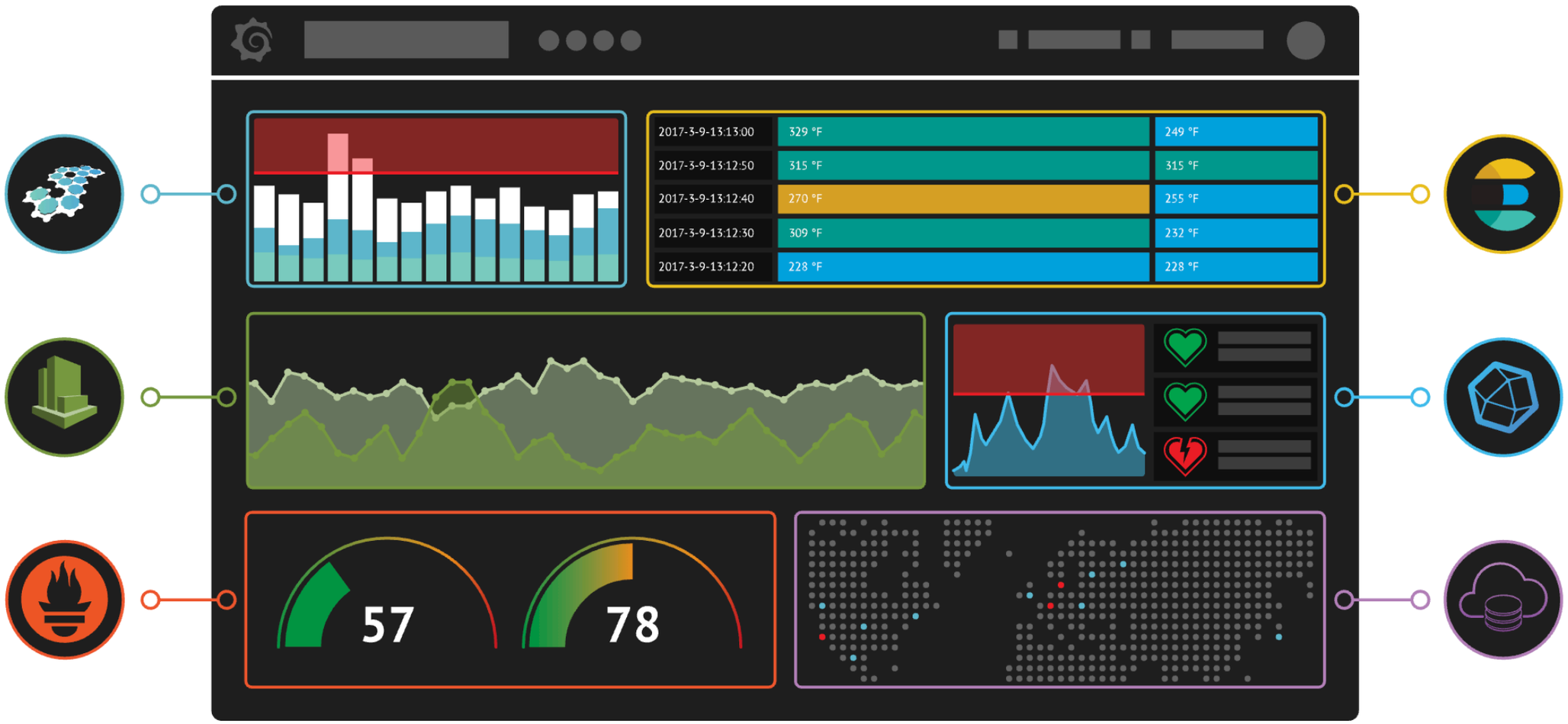
Easy to **run**.

Told ya, slight bias.

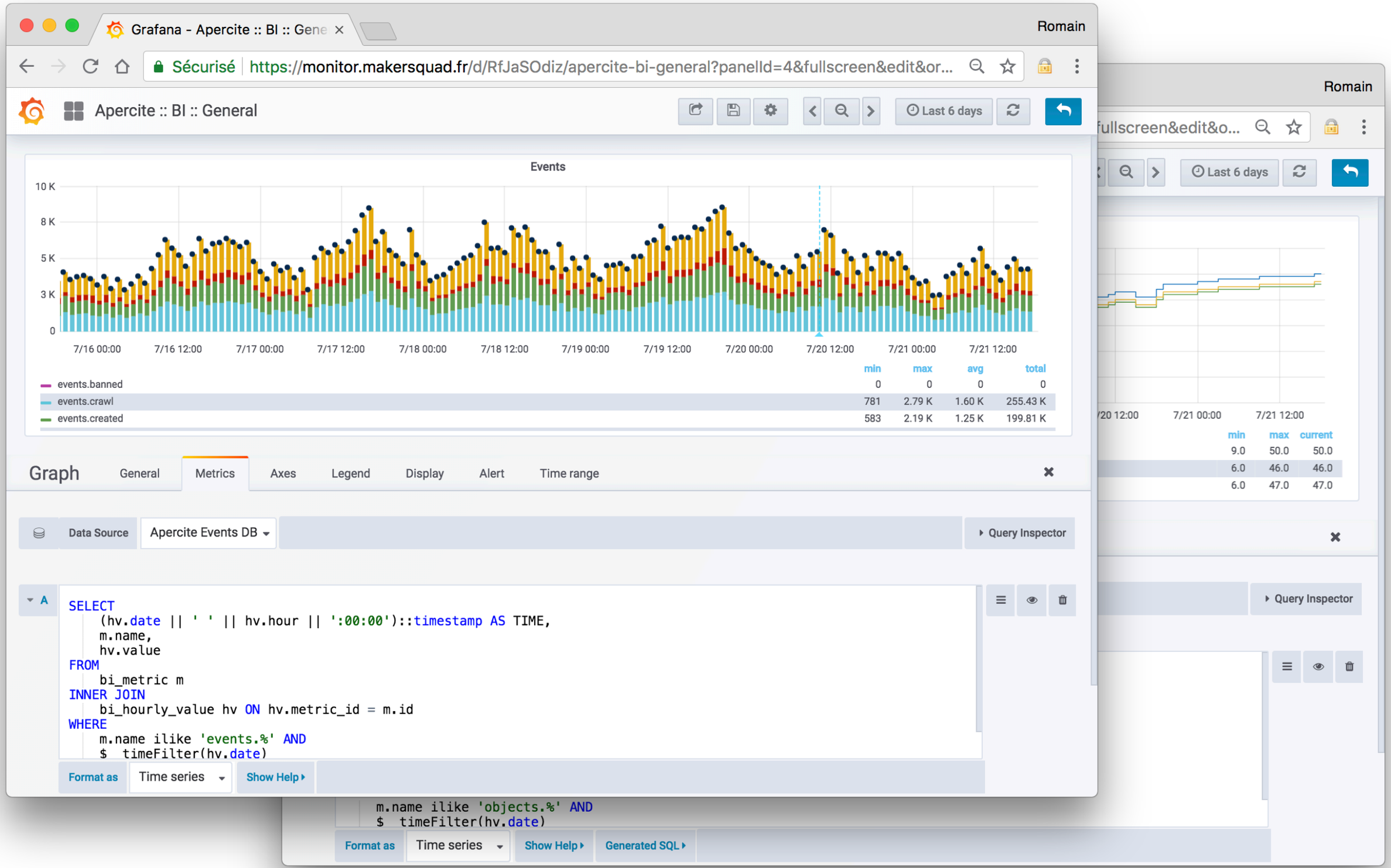
VISUALIZE

Grafana

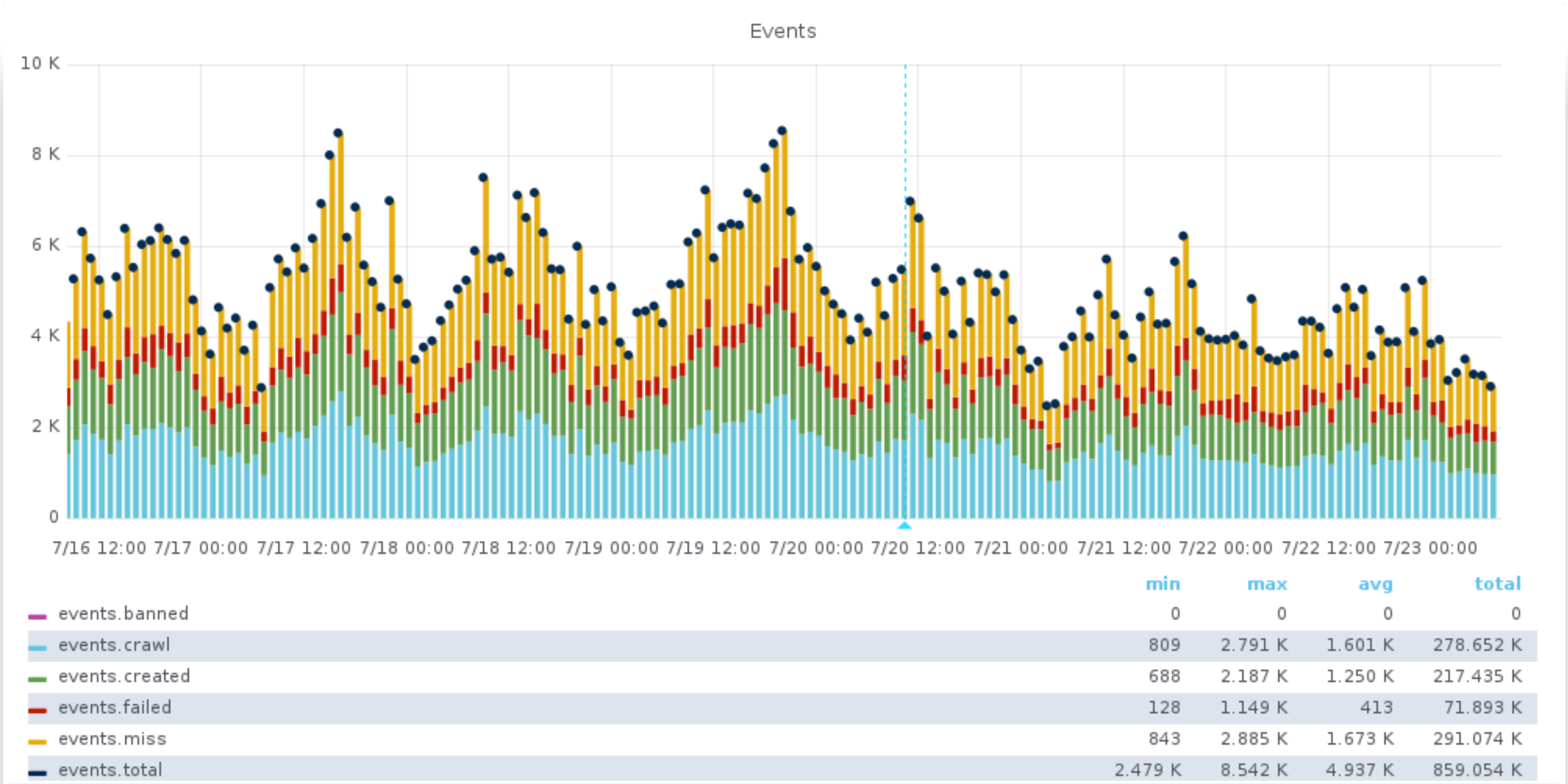
Analytics & Monitoring



Dashboards

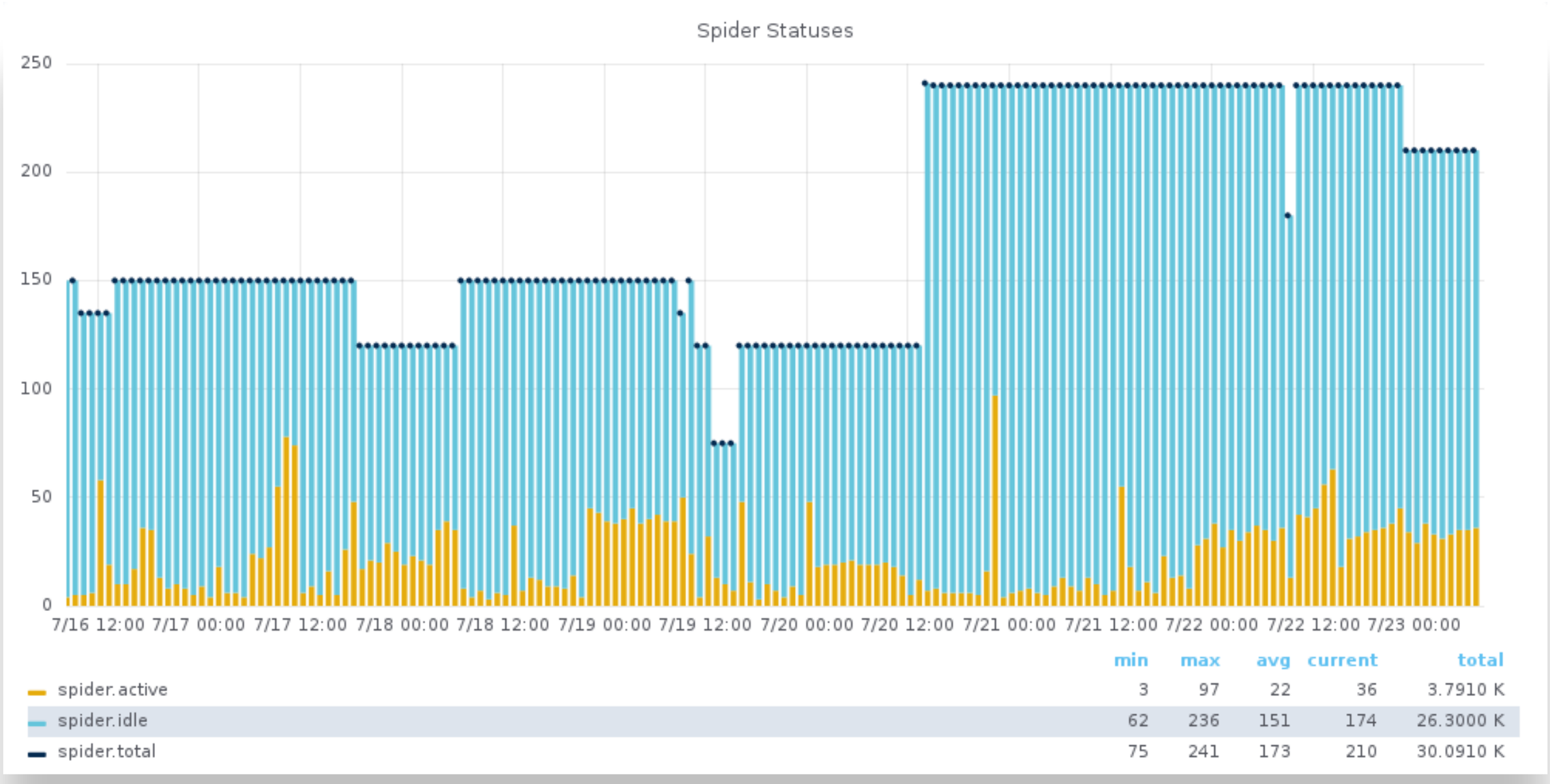

Grafana - Apercite :: BI :: Gene
Romain
Sécurisé
https://monitor.makersquad.fr/d/RfJaSOdiz/apercite-bi-general?panelId=4&fullscreen&edit&or...
Apercite :: BI :: General
Last 6 days
Events
10 K
8 K
5 K
3 K
0
7/16 00:00
7/16 12:00
7/17 00:00
7/17 12:00
7/18 00:00
7/18 12:00
7/19 00:00
7/19 12:00
7/20 00:00
7/20 12:00
7/21 00:00
7/21 12:00
min max avg total
events.banned 0 0 0 0
events.crawl 781 2.79 K 1.60 K 255.43 K
events.created 583 2.19 K 1.25 K 199.81 K
Graph
General
Metrics
Axes
Legend
Display
Alert
Time range
Data Source
Apercite Events DB
Query Inspector
A
SELECT
(hv.date || ' ' || hv.hour || ':00:00')::timestamp AS TIME,
m.name,
hv.value
FROM
bi_metric m
INNER JOIN
bi_hourly_value hv ON hv.metric_id = m.id
WHERE
m.name ilike 'events.%' AND
$__timeFilter(hv.date)
Format as
Time series
Show Help
fullscreen&edit&o...
Last 6 days
/20 12:00
7/21 00:00
7/21 12:00
min max current
9.0 50.0 50.0
6.0 46.0 46.0
6.0 47.0 47.0
Query Inspector
m.name ilike 'objects.%' AND
$__timeFilter(hv.date)
Format as
Time series
Show Help
Generated SQL

Quality of Service



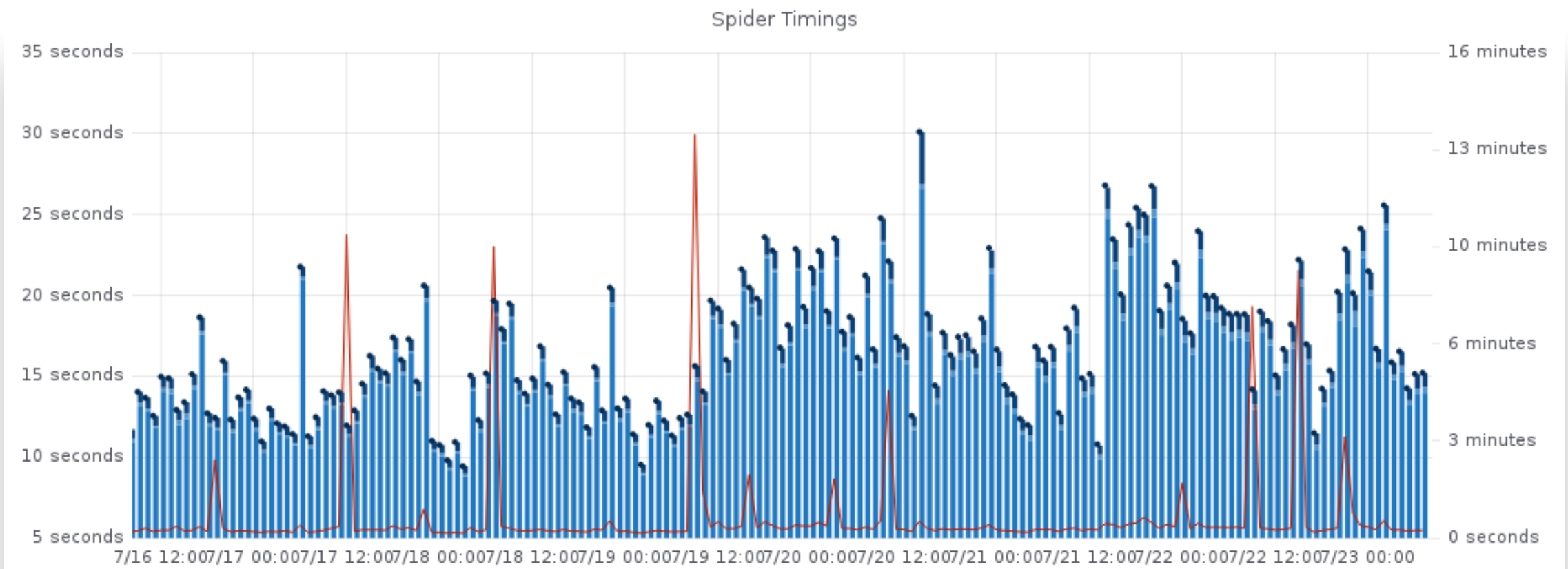
	min	max	avg	total
events.banned	0	0	0	0
events.crawl	809	2.791 K	1.601 K	278.652 K
events.created	688	2.187 K	1.250 K	217.435 K
events.failed	128	1.149 K	413	71.893 K
events.miss	843	2.885 K	1.673 K	291.074 K
events.total	2.479 K	8.542 K	4.937 K	859.054 K

Quality of Service



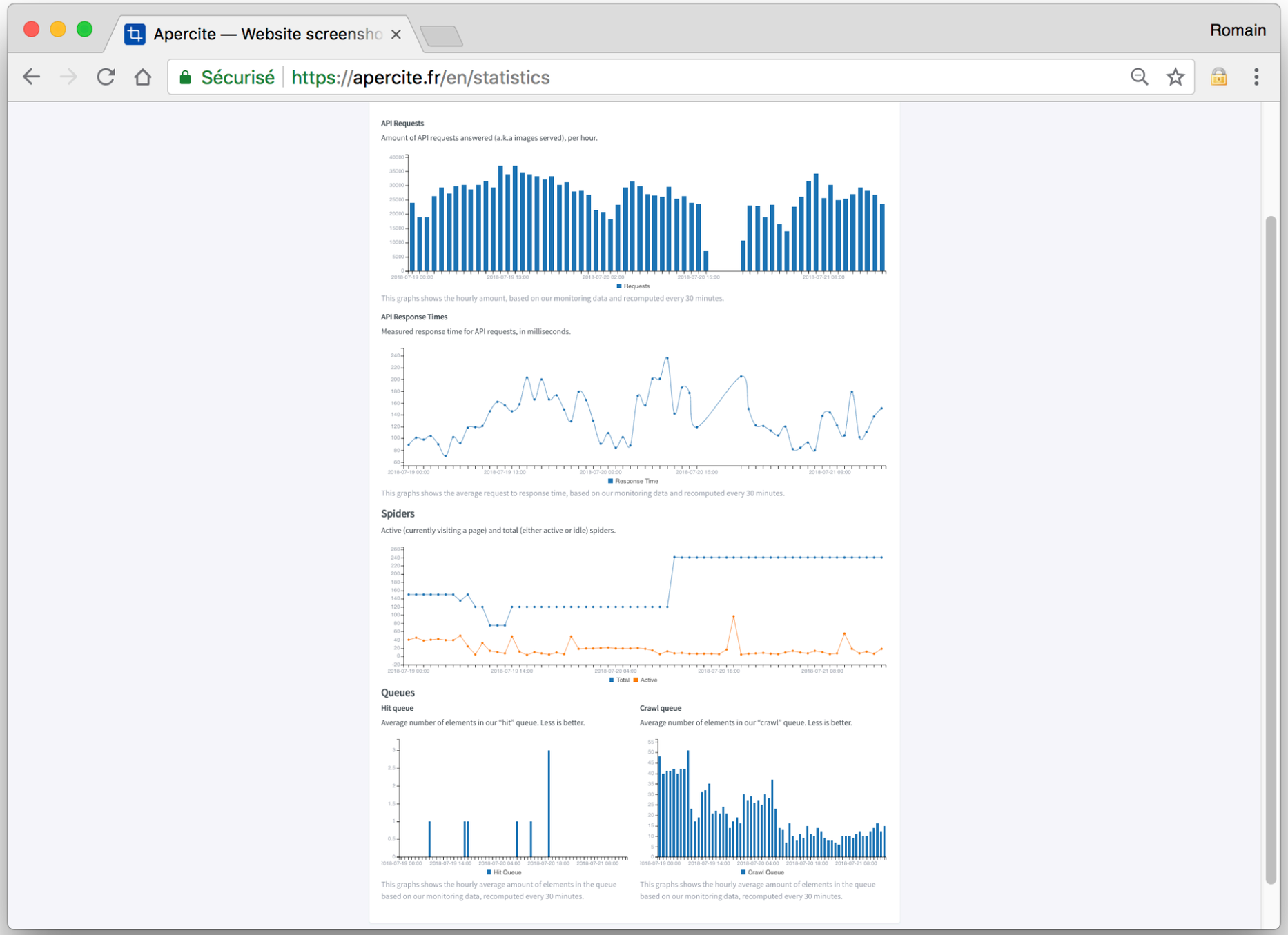
	min	max	avg	current	total
spider.active	3	97	22	36	3.7910 K
spider.idle	62	236	151	174	26.3000 K
spider.total	75	241	173	210	30.0910 K

Quality of Service



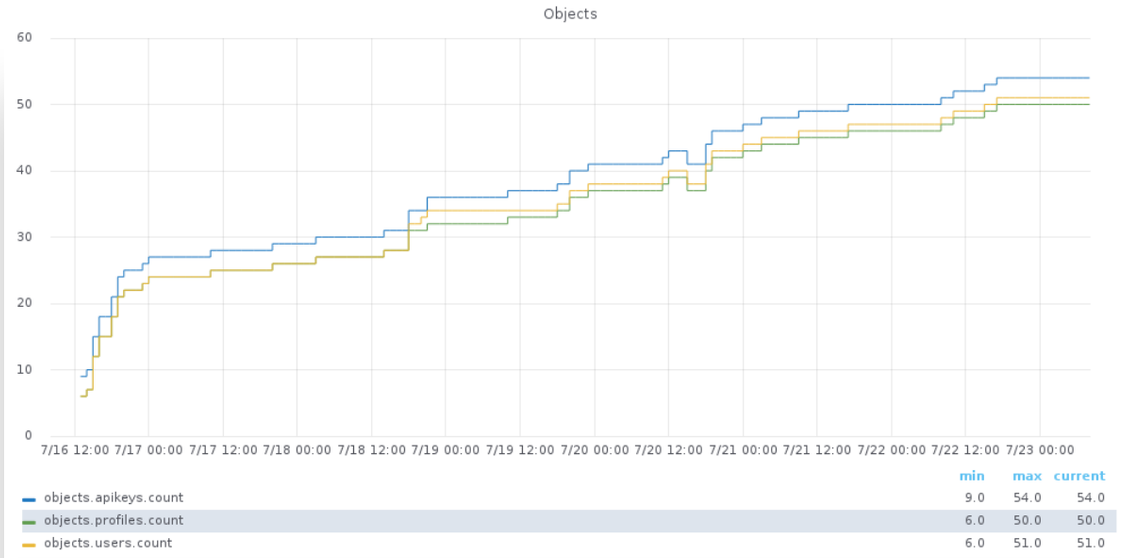
	min	max	avg	current
spider.timings.browser	8 seconds	26 seconds	15 seconds	13 seconds
spider.timings.checks	140 milliseconds	933 milliseconds	265 milliseconds	381 milliseconds
spider.timings.delta (right-y)	9 seconds	13 minutes	42 seconds	15 seconds
spider.timings.thumbs	451 milliseconds	3 seconds	829 milliseconds	855 milliseconds
spider.timings.total	9 seconds	30 seconds	16 seconds	15 seconds

Public Dashboards

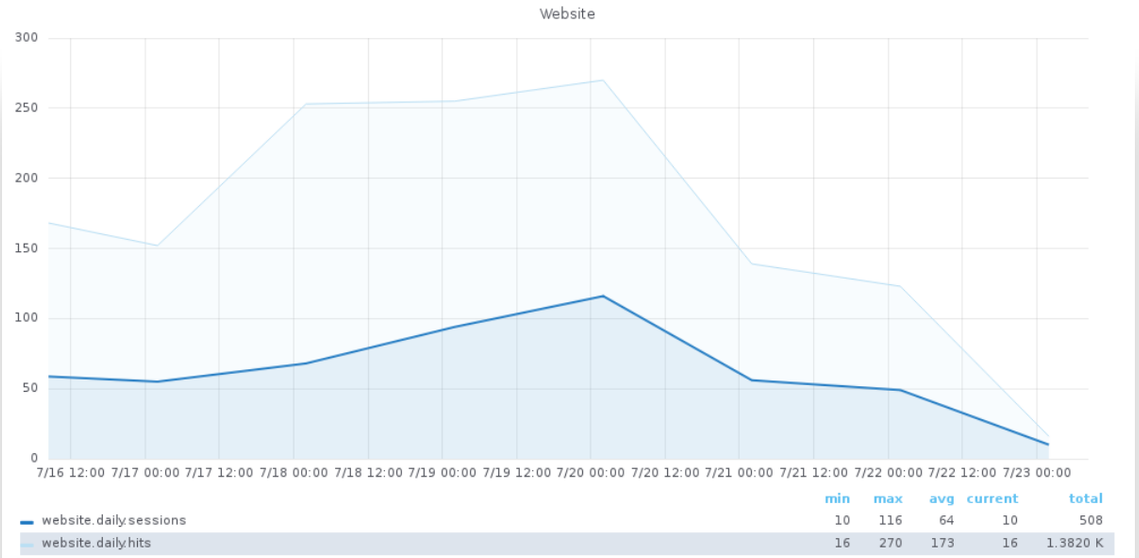


Acquisition Rate

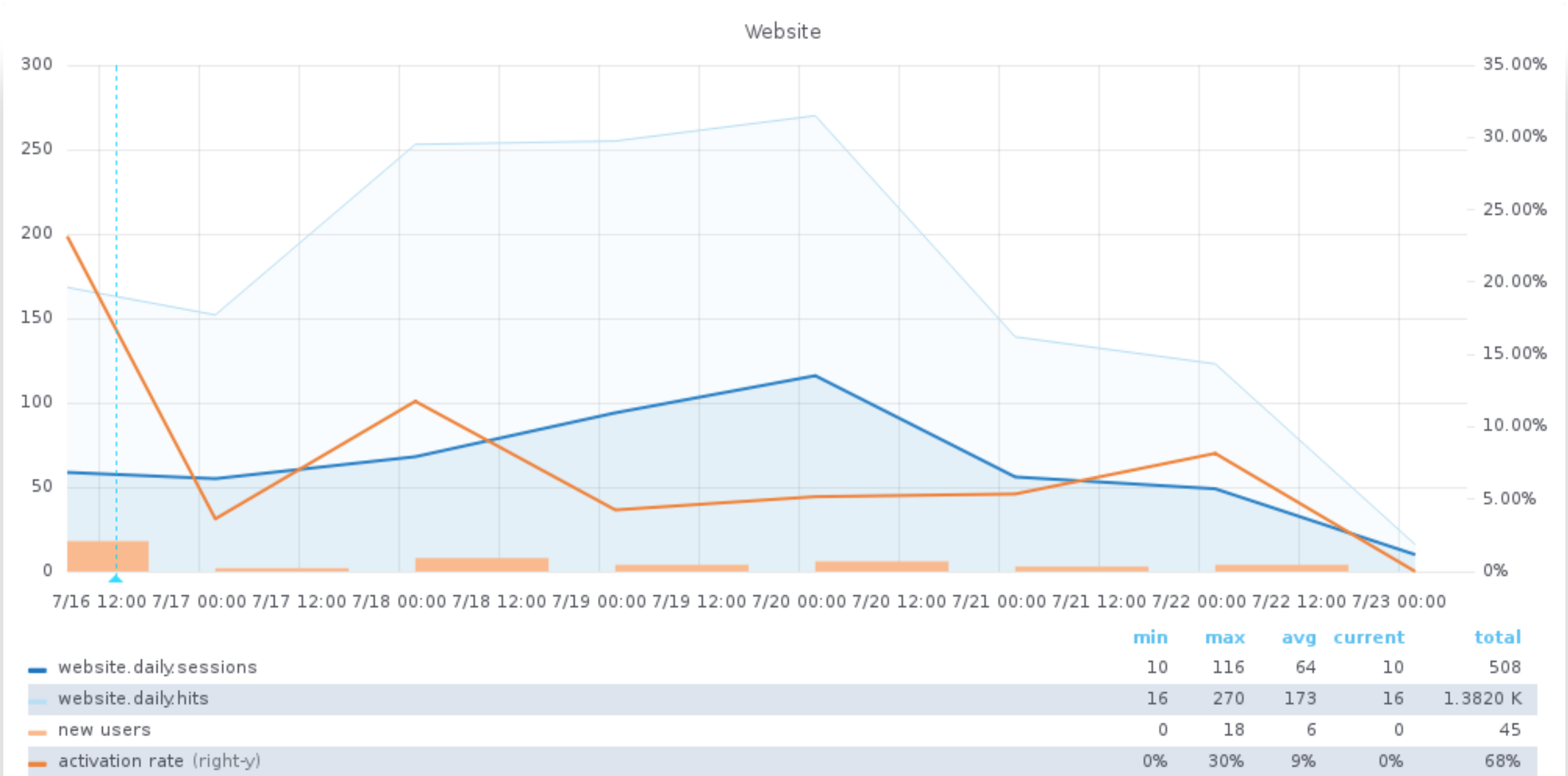
User Counts



New Sessions



Acquisition Rate



	min	max	avg	current	total
website.daily.sessions	10	116	64	10	508
website.daily.hits	16	270	173	16	1.3820 K
new users	0	18	6	0	45
activation rate (right-y)	0%	30%	9%	0%	68%

We’re just getting started.

MONITOR

Iteration 0

- Cron job (K8S) runs everything every 30 minutes.
- No way to know if something fails.
- Expensive tasks.
- Hard to run manually.

Airflow

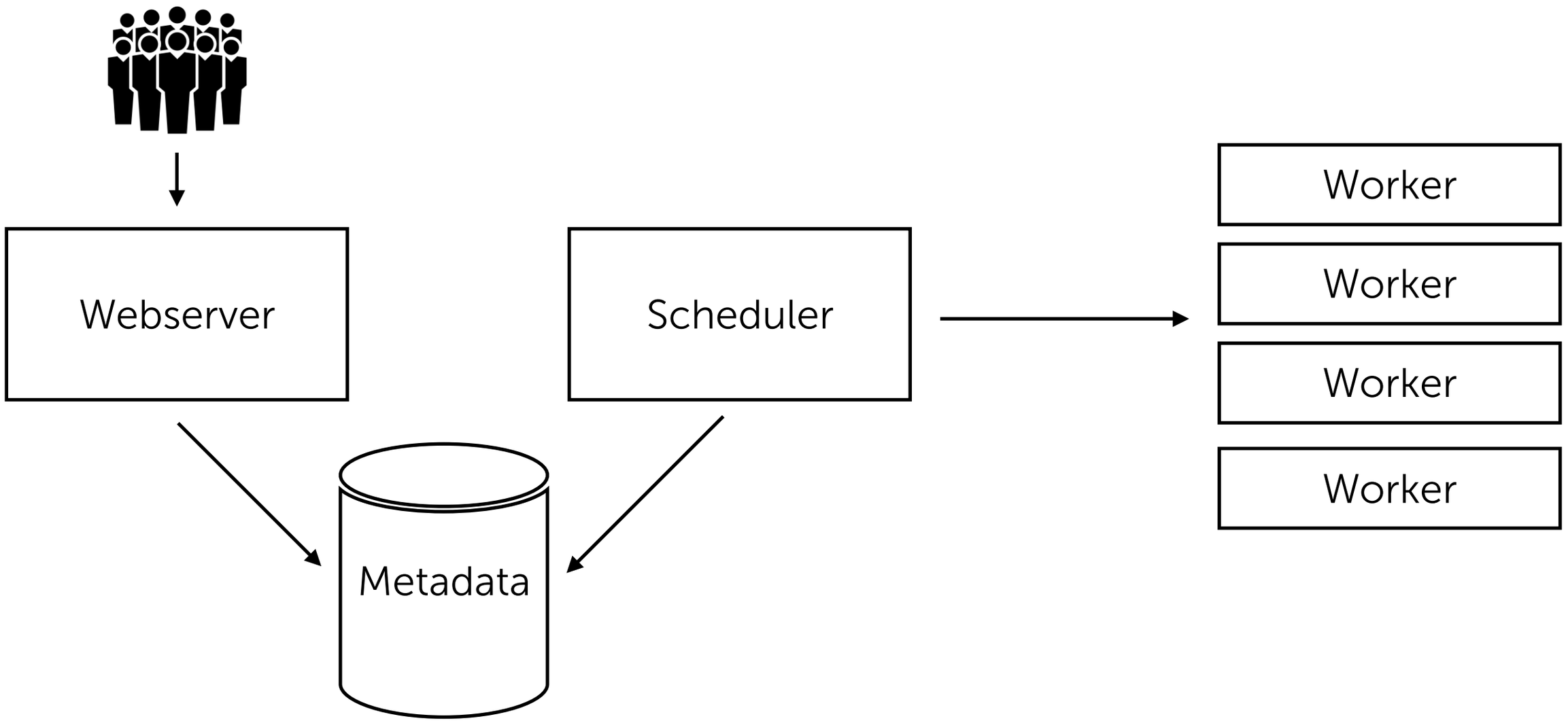
«**Airflow** is a platform to programmatically **author**, **schedule** and **monitor** workflows.»

- Official docs

Airflow

- Created by Airbnb, joined Apache incubation.
- Schedules & monitor jobs.
- Distribute workloads through Celery, Dask, K8S…
- Can run anything, not just Python.

Airflow



Simplified to show high-level concept.
Depends on executor (celery, dask, k8s, local, sequential …)

Airflow - DAGs

Romain

Airflow | DAGs | Data Profiling | Browse | Admin | Docs | About | 09:36 UTC

DAGs

Search:

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	On	apercite.analytics.cleanup	@daily	apercite	1	2018-07-24 00:00	4	
	On	apercite.analytics.events	@hourly	apercite	1	2018-07-25 08:00	71 1	
	On	apercite.analytics.events_timings	@hourly	apercite	1	2018-07-25 08:00	71 1	
	On	apercite.analytics.google_analytics	@hourly	apercite	1	2018-07-25 08:00	71 2	
	On	apercite.analytics.objects	@hourly	apercite	1	2018-07-25 08:00	70 2	
	On	apercite.analytics.prometheus	@hourly	apercite	1	2018-07-25 08:00	71 1	
	On	apercite.analytics.spiders	@hourly	apercite	1	2018-07-25 08:00	70 2	
	Off	read_object_counts		apercite				
	Off	sample_dag	@once	airflow	3	2017-09-10 00:00	5	

Showing 1 to 9 of 9 entries

« < 1 > »

Hide Paused DAGs

DAGs

```python
import shlex
from airflow import DAG
from airflow.operators.bash_operator import BashOperator

def _get_bash_command(*args, module='apercite.analytics'):
    return '(cd /usr/local/apercite; /usr/local/env/bin/python -m {} {})'.format(
        module, ' '.join(map(shlex.quote, args)),
    )

def build_dag(name, *args, schedule_interval='@hourly'):
    dag = DAG(
        name,
        schedule_interval=schedule_interval,
        default_args=default_args,
        catchup=False,
    )

    dag >> BashOperator(
        dag=dag,
        task_id=args[0],
        bash_command=_get_bash_command(*args),
        env=env,
    )

    return dag
```

DAGs

```
# Build datasource-to-metrics-db related dags.
for source in ('google-analytics', 'events', 'events-timings', 'spiders',
'prometheus', 'objects'):
    name = 'apercite.analytics.' + source.replace('-', '_')
    globals()[name] = build_dag(name, 'read', source, '--write')

# Cleanup dag.
name = 'apercite.analytics.cleanup'
globals()[name] = build_dag(name, 'clean', 'all', schedule_interval='@daily')
```

Admin - Connections - Airflow

Romain

Airflow DAGs Data Profiling Browse Admin Docs About 08:04 UTC

Connections

List (29) Create With selected

	Conn Id	Conn Type	Host	Port	Is Encrypted	Is Extra Encrypted
	airflow_ci	mysql	localhost		⊖	⊘
	airflow_db	mysql	localhost		⊖	⊖
	apercite_events	postgres	cloudsql-proxy.database	5432	⊘	⊖
	apercite_website	postgres	cloudsql-proxy.database	5432	⊘	⊖
	aws_default	aws			⊖	⊘
	beeline_default	beeline	localhost	10000	⊖	⊘
	bigquery_default	bigquery			⊖	⊖
	databricks_default	databricks	localhost		⊖	⊖
	druid_ingest_default	druid	druid-overlord	8081	⊖	⊘
	emr_default	emr			⊖	⊘
	fs_default	fs			⊖	⊘

Data Sources

```
from airflow.models import Connection
from airflow.settings import Session

session = Session()
website = session.query(Connection).filter_by(conn_id='apercite_website').first()
events = session.query(Connection).filter_by(conn_id='apercite_events').first()
session.close()
```

```
env = {}

if website:
    env['DATABASE_HOST'] = str(website.host)
    env['DATABASE_PORT'] = str(website.port)
    env['DATABASE_USER'] = str(website.login)
    env['DATABASE_NAME'] = str(website.schema)
    env['DATABASE_PASSWORD'] = str(website.password)

if events:
    env['EVENT_DATABASE_USER'] = str(events.login)
    env['EVENT_DATABASE_NAME'] = str(events.schema)
    env['EVENT_DATABASE_PASSWORD'] = str(events.password)
```

Warning: sub-optimal

Learnings

- Multiple services, not trivial
- Helm charts :-(
- Astronomer Distro :-)
- Read the Source, Luke

Outro

Bonobo helps you build assembly lines.
Does not care about the surrounding factory.

Airflow helps you manage the whole factory.
Does not care about the jobs' content.

Grafana let you see the data, with only a few queries to write.

romain@makersquad.fr

rdorgueil