

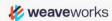
grafanalib: Dashboards as code

GrafanaCon, Amsterdam, March 2018

Jonathan Lange, VP of Engineering

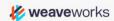


DASHBOARDS!





Grafana is awesome





Manually created dashboards – problems

- Hard to enforce consistency
- Difficult to automate
- Uncertain history





GIT!

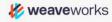


GITOPS!



JSON files in Git

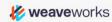
- Enforce consistency with custom linter
- Get meaningful history from Git and GitHub PRs





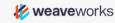
Example

```
"rows": [
  "panels": [
     "legend": {
      "alignAsTable": false,
      "avg": false,
      "current": false,
      "hideEmpty": false,
      "hideZero": false,
      "max": false,
      "min": false,
      "rightSide": false,
      "show": true,
      "sideWidth": null.
      "total": false,
      "values": false
```



JSON files in Git – problems

- PRs almost unreadable
- Graph and dashboard creation still manual
- Lint script very complicated





Why not just *generate* the JSON correctly instead?

What we wanted

- As simple as possible—mostly data
- No defaults cluttering our configuration
- Re-use of common patterns
- Approachable syntax



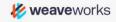
grafanalib

https://github.com/weaveworks/grafanalib

grafanalib

- Python EDSL for Grafana dashboards
- Released December 2016
- 21 contributors
- 400 stars

How does it work?





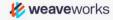
\$ generate-dashboard -o mydash.json \
mydash.dashboard.py





```
dashboard = common.Dashboard(
  title="Scope > Services",
  rows=[
    scope row('Collection', 'scope/collection'),
    scope_row('Query', 'scope/query'),
    scope row('Control', 'scope/control'),
    scope_row('Demo', 'extra/demo'),
```

```
def scope_row(name, job):
    return G.Row(panels=[
        scope_qps_graph(name, job),
        scope_latency_graph(name, job),
])
```



```
def scope_qps_graph(name, job):
  expr_template = 'sum(irate(scope_request_duration_seconds_count{job="%s",status_code=~"%s"}[1m]))'
  expressions = [expr_template % (job, status_re) for status_re in ['1..', '2..', '3..', '4..', '5..']]
  return stacked(Graph(
    data_source=PROMETHEUS,
    title='%s QPS' % (name,),
    expressions=expressions,
    yAxes=[
       YAxis(format=OPS_FORMAT),
       YAxis(format=SHORT FORMAT),
  ))
```

- Makefile to turn *.dashboard.py into *.json
- Dockerfile to combine *.json with base Grafana image
- In your CI, push the Docker image to your registry
- Use a CD tool (like Weave Cloud!) to deploy Grafana to your cluster

Actually using grafanalib

- Easy to add new services—just a few lines of Python
- Easy to review changes to dashboards—no extraneous information
- Easy to automate common patterns—it's just functions
- Consistency between dashboards makes on-call easier—less cognitive overhead

Future

- Huge PR for migrating *.json files to *.dashboard.py to make it easier to get started
- Possible consolidation of Grafana "dashboards as code" projects
 https://community.grafana.com/t/grafana-dashboards-as-code-for-newcomers/5334

Takeaways

- GitOps is pretty cool
- "Dashboards as code" really works
- Try grafanalib!

QUESTIONS?

https://github.com/weaveworks/grafanalib/ Contributions welcome!

We are hiring!

Engineers, Developer Advocates, Community Managers https://weave.works/hiring

