Fool-proof K8s dashboards for sleep-deprived oncalls

David Kaltschmidt
@davkals

Kubecon 2019

I'm David

All things UX at Grafana Labs

If you click and are stuck, reach out to me.
david@grafana.com

Twitter: @davkals

Grafana Labs

Outline

- Quick Grafana intro
- Dashboarding for k8s oncalls
- Maturity level framework

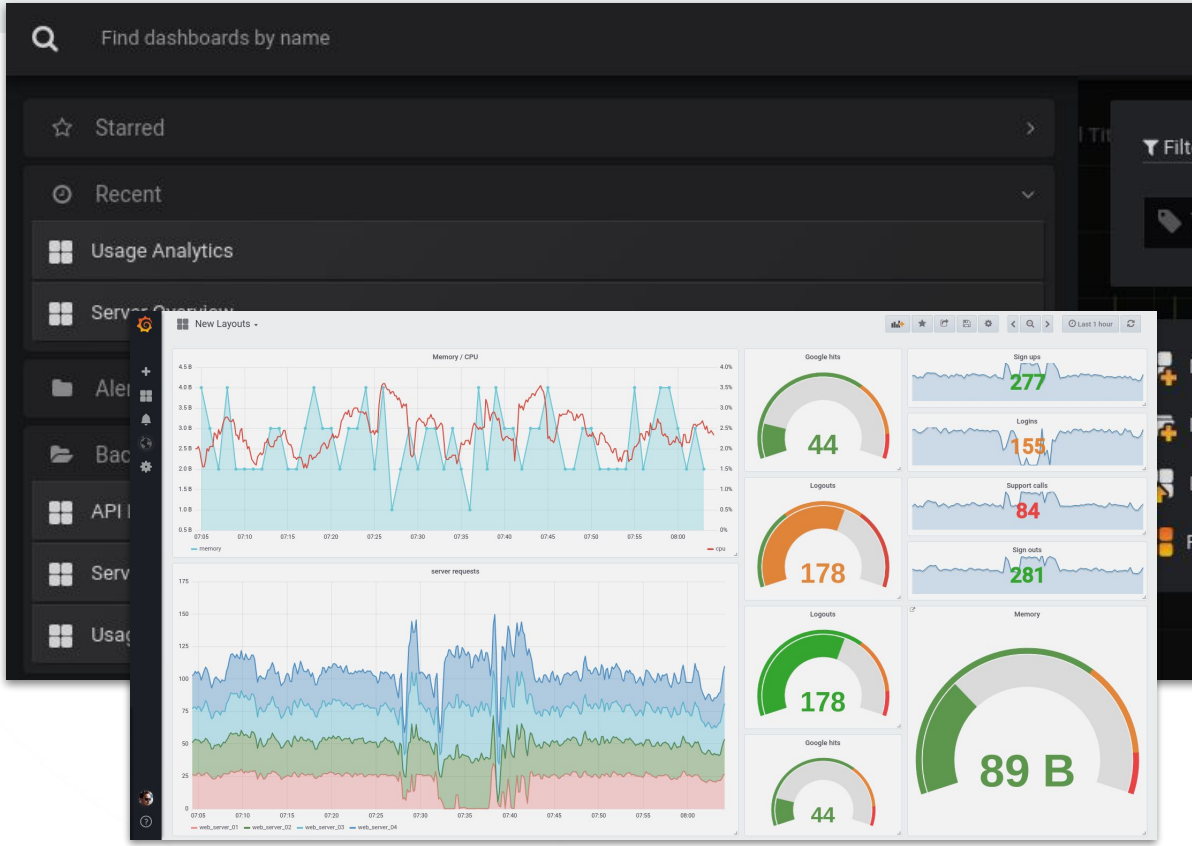
Grafana Labs

Grafana intro and updates

Grafana

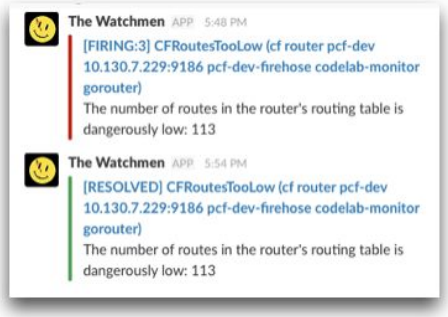
Dashboarding solution

Observability platform



Grafana Labs

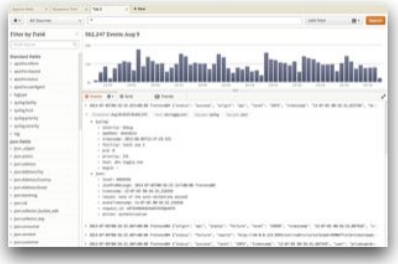
1. Alert



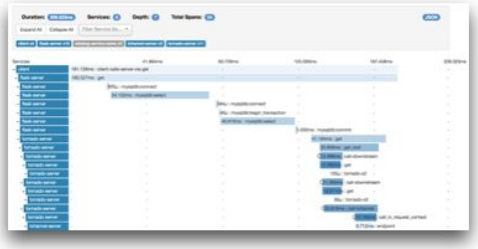
2. Dashboard

3. Adhoc Query

4. Log Aggregation



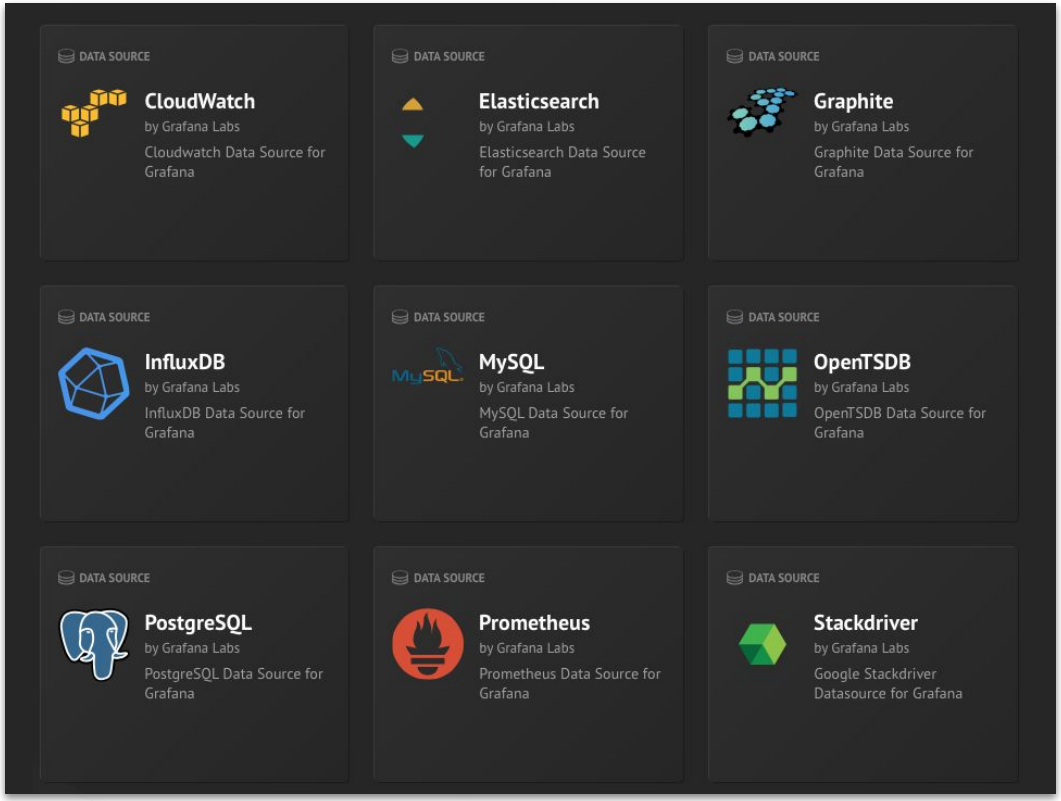
5. Distributed Tracing



Fix!

Troubleshooting journey

Unified way to look at data from different sources



6.0 release: New graph panel controller to quickly iterate how to visualize

https://github.com/grafana/loki

Loki BETA release: live tailing and context view

Dashboarding for Kubernetes oncalls

On-call

- On call is hard. You're not creating new things, you're fighting fires.
- Majority of on-call activities revolve around troubleshooting

Image: Slack alert

On-call

- We’re focussing on the role that dashboards play in troubleshooting
- When you look at dashboards, you’re not only trying to find the issue, but you also want to quickly eliminate areas that are working fine

Image: a couple of panels where only one has red

On call for Kubernetes

- In kubernetes the elimination is made difficult by an explosion of concepts
  - Namespaces, services, pods, containers, replicasets, daemonsets
  - Clusters and node

Image: Diagram of pods and containers

On call for Kubernetes

- How can effective dashboarding guide you through this jungle?
- the tools we put in place should reduce cognitive burden, not add to it

Image: List of hierarchical dashboards

The path to 1,000 dashboards

- Image: GIF of endless dashboard lists

The path to 1,000 dashboards

- Dashboard sprawl negatively affects time to find the right dashboard
- Enablers:
  - Everyone can modify dashboards (being able to edit and save gives uncertainty about panel purpose)
  - Duplicating dashboards and changing “one thing” (worse: keeping original tags)
  - One-off dashboards with a specific purpose that have been forgotten

Introducing DMM:
Dashboarding Maturity Model

Dashboarding Maturity Model

- Practices and vocabulary to use and to make decisions
- 3 levels: low, medium, high
- You may be in various levels on different parts of your dashboarding practice

Dashboarding maturity levels

Low

Default state
(no strategy)

Medium

Managing use of
methodical dashboards

High

Optimizing use,
consistency by design

Low maturity: Sprawl

- Everyone can modify, no reviews
- Duplicate used regularly, tags lose meaning
- One-off dashboards

Image: Duplicate button

Low maturity: Single point of failure

- No version control
- Live version is source of truth

Image: Angry tech worker

Low maturity: Browsing

- No alerts
- Need to browse regularly

Image: Dev staring at dashboard

Dashboarding maturity levels

Low

Default state
(no strategy)

Medium

Managing use of
methodical dashboards

High

Optimizing use,
consistency by design

Medium maturity: Sprawl prevention

- Use of template variables (instead of duplicating dashboards) [Docs]

Image: Template variable UI

Medium maturity: Methodical dashboards

- Hierarchical dashboards
- Aggregated views with drill-down
- Hierarchies:
  - Cluster -> node
  - Namespace -> pod -> container

Image: Cluster panel with drill-down link

Medium maturity: Methodical dashboards

- USE method for resources [Ref]:
  For each resource measure utilization, saturation, errors
- RED methods for services [Video]

Image: Service dashboard

Medium maturity: Methodical dashboards

- Normalizing panel axis
- Expressive charts

Image: Dashboard panel of CPU usage

Medium maturity: Managing dashboards

- Version controlled dashboard sources
- Currently by copy/pasting JSON
- RFC in our design doc

Image: Github PR

Medium maturity: Infrequent browsing

- Most dashboards are linked to by alerts
- Or you arrive via drill-down
  - related: hierarchical dashboards make use of template variables, it's very impractical to go through the variable list and select the one after the other

Image: Alert with link to dashboard

Dashboarding maturity levels

Low

Default state
(no strategy)

Medium

Managing use of
methodical dashboards

High

Optimizing use,
consistency by design

High maturity: Optimizing use

- Actively reducing sprawl
- Regularly reviewing existing dashboards
- Tracking use

Image: Robot hoover

High maturity: Consistency by design

- Use of scripting libraries to generate dashboards
  - grafonnet (Jsonnet)
  - grafanalib (Python)
- Define timeseriesGraph(DS, QUERY) once, apply same attributes/styles across all dashboards

High maturity: Consistency by design

- Scripting languages reduce change sets for reviews

Image: PR of single-line change

High maturity: Mixins

- Mixins are sets of dashboards and alerts for a given software, peer-reviewed [Kubernetes mixins]

Image: Repo screenshot

Future workflow: Dashboard as code

- Live edit JSON and preview dashboards
- Live edit JSONNET sources or Python sources and preview in browser
- Open PR directly from Grafana

Image: Mockup of the edit and preview experience

Dashboarding maturity levels

Low

Default state
(no strategy)

- Everyone can modify
- Duplicate used regularly
- One-off dashboards
- No version control
- Lots of browsing

Medium

Managing use of methodical dashboards

- prevention of sprawl
- use of template variables
- methodical dashboards
- hierarchical dashboards
- normalised panels axis
- version control
- infrequent browsing

High

Optimizing use,
consistency by design

- active sprawl reduction
- use of scripting libraries
- use of mixins
- no editing in the browser
- browsing is the exception

DMM for oncalls:

Your dashboarding practices should reduce cognitive load, not add to it.

Tack for listening

UX feedback to
david@grafana.com
@davkals