Prometheus – For Big & Little People

Simon Lyall

- Sysadmin (it says “DevOps Engineer” in my job title)
- Large Company, Auckland, New Zealand
- Use Prometheus at home on workstations, home servers and hosted Vms
- Run Prometheus at work on Kubernetes Clusters, EC2 infrastructure, etc

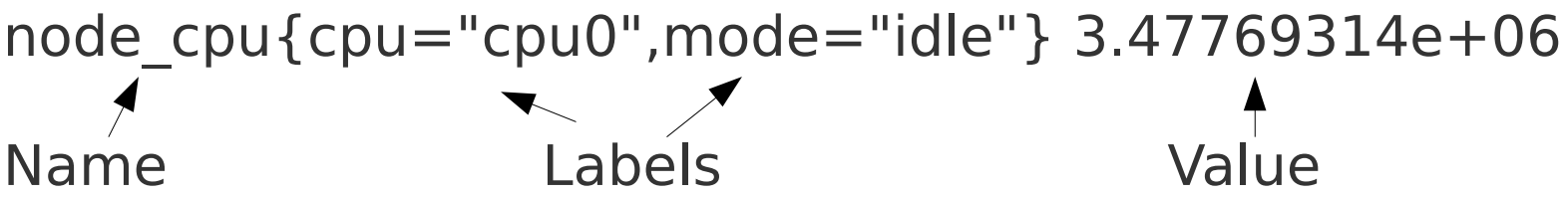
Prometheus – For Big & Little People

- Intro
- Getting Data
- Alerting
- Storage
- Display
- Extras
- Summary

Prometheus Introduction

- Metrics
  - Name [Labels] + Timestamp + Value
- Single Daemon
  - Connects to “exporters” via HTTP GET
  - Gets list of values
  - Gathered often. 10s or 15s common
- Stored on local Disk
- Exported via API

Prometheus Introduction



node_uname_info{domainname="(none)",machine="x86_64",nodename="prometheus.darkmere.gen.nz",release="3.10.0-862.14.4.el7.x86_64",sysname="Linux",version="#1 SMP Wed Sep 26 15:12:11 UTC 2018"} 1

Prometheus Introduction

```
api_http_request_latencies_second{quantile="0.5"}

count(up{nodename=~".*web.*”} == 1)

sum(kube_node_labels{job="kube-state-metrics"}) by
(label_failure_domain_beta_kubernetes_io_zone)
```

Getting Data

- Exporters
  - Gather metrics from source
  - Expose http endpoint
  - Around 100 distributed
- Directly metriced Apps
- Internal apps should expose metrics

Getting Data – Many Layers

- Prometheus can gather data on all layers of the stack
- For Kubernetes:
  - Cloudwatch
  - node_exporter
  - Kublet
  - Cadvisor
  - Kube-state-metrics

Getting Data – Even More Layers

- Kubernetes (cont)
  - JMX (JVM)
  - Application directly exposed
  - Service Mesh
  - Load balancers
  - Blackbox
  - Other apps

Getting Data - Problems

- Thousands of metrics per Server
- Overlaps of metrics (ie memory used by app)
- Alignment between layers (instance v node , pod v container)
- Some source costly (ie Cloudwatch)
- Code/Apps not instrumented

Getting Data

- Small
  - Standard exporters (node, db, blackbox)
  - Textfile via Node_exporter for special stats
- Medium
  - Many standard exporters
  - Textfile, mtail, gateways to other monitoring systems
- Large
  - Instrumented code
  - Federation and summaries

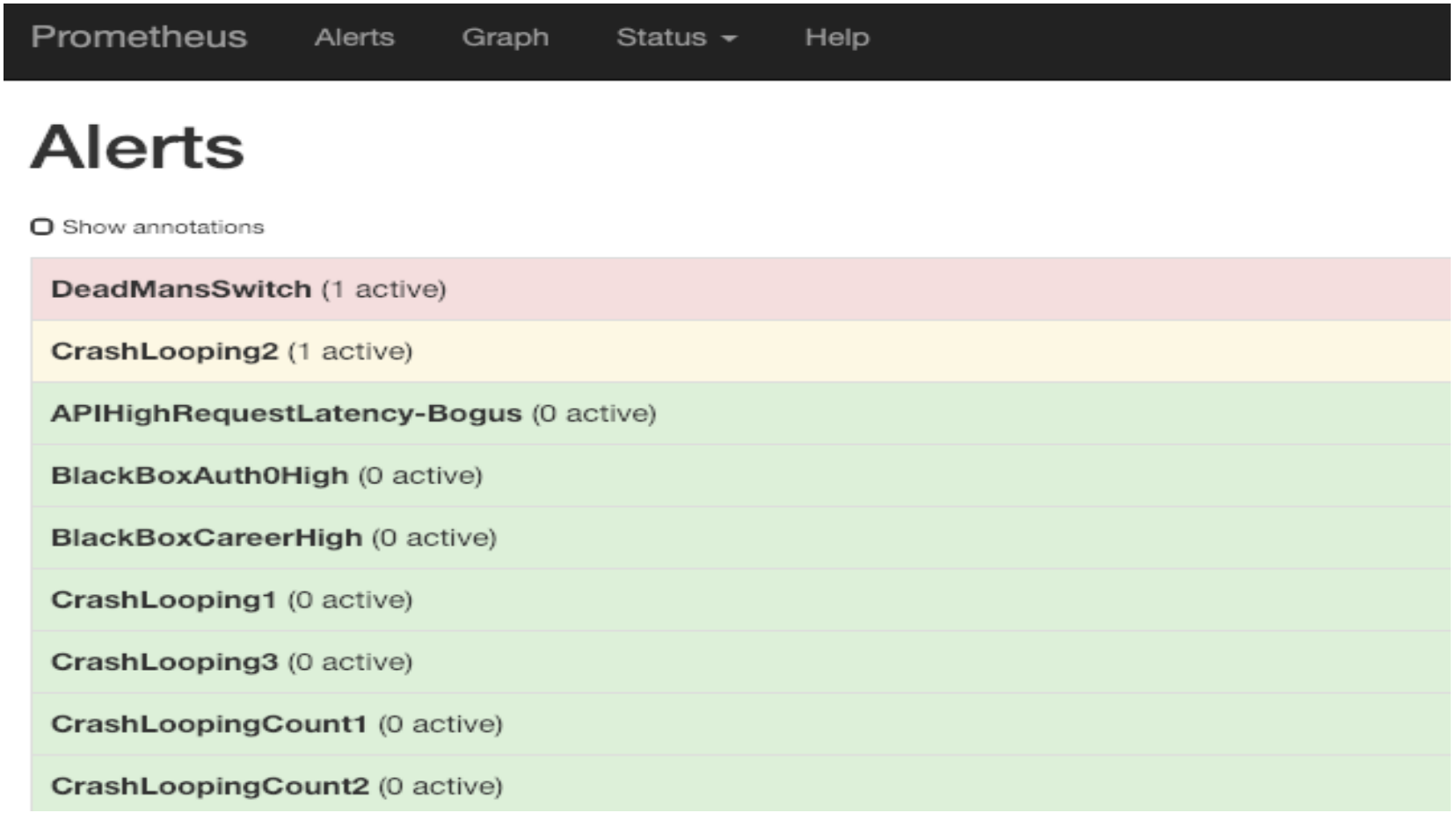
Service Discovery

- Static in Config
- Built in EC2 / Openstack / Kubernetes / etc
- File Based

Service Discovery

- Small – Static config, Simple auto discovery
- Medium – Mainly Auto discovery, templated config
- Large – Lots of templates, Split servers

Alerting


Prometheus
Alerts
Graph
Status
Help
Alerts
Show annotations
DeadMansSwitch (1 active)
CrashLooping2 (1 active)
APIHighRequestLatency-Bogus (0 active)
BlackBoxAuth0High (0 active)
BlackBoxCareerHigh (0 active)
CrashLooping1 (0 active)
CrashLooping3 (0 active)
CrashLoopingCount1 (0 active)
CrashLoopingCount2 (0 active)

Alerting

- Prom + alert rules + alert manager (xN)
- Amtool
- Silences (eg during maintenance)
- Labels very important
- Slack / Email
- PagerDuty / Victorops / Opsgenie / pagertree
- Free -> $50 per user per month

Alerting – Low priority Alerts

- Schedule alert during office hours
- Have on dashboard?
- Avoid email
- Keep low (say <20 /day)

Alerting

- Small
  - Send oncall everything
- Medium
  - Prioritise high/low split
- High
  - Multiple Groups / Multiple Levels
  - Prioritise Filtering, Automating fixes

Storage

- Problem
  - Even small installations do thousands of writes per second
  - Reads/Queries may run against huge amounts of data
- Standard TSDB
  - Handles the above. But...
  - Not redundant
  - Can get corrupt
  - Doesn’t scale forever
- Replacements – Complicated and new

Storage

- Small Site
  - Backup data regularly. Rollback to backup in event of outage
- Medium Site
  - Backup data regularly. Flip to 2nd instance in event of outage.
- Large Site
  - External clustered storage (Thanos, M3, InfluxDB)
  - Federate to scale collection/storage

Display

- Built in dashboard
  - Okay for testing, developing rules
- API
  - Can be used by other tools (eg Grafana)
  - Although doesn’t seem to be common (and ???)
- Lack of good tools to explore thousands of different metrics easily

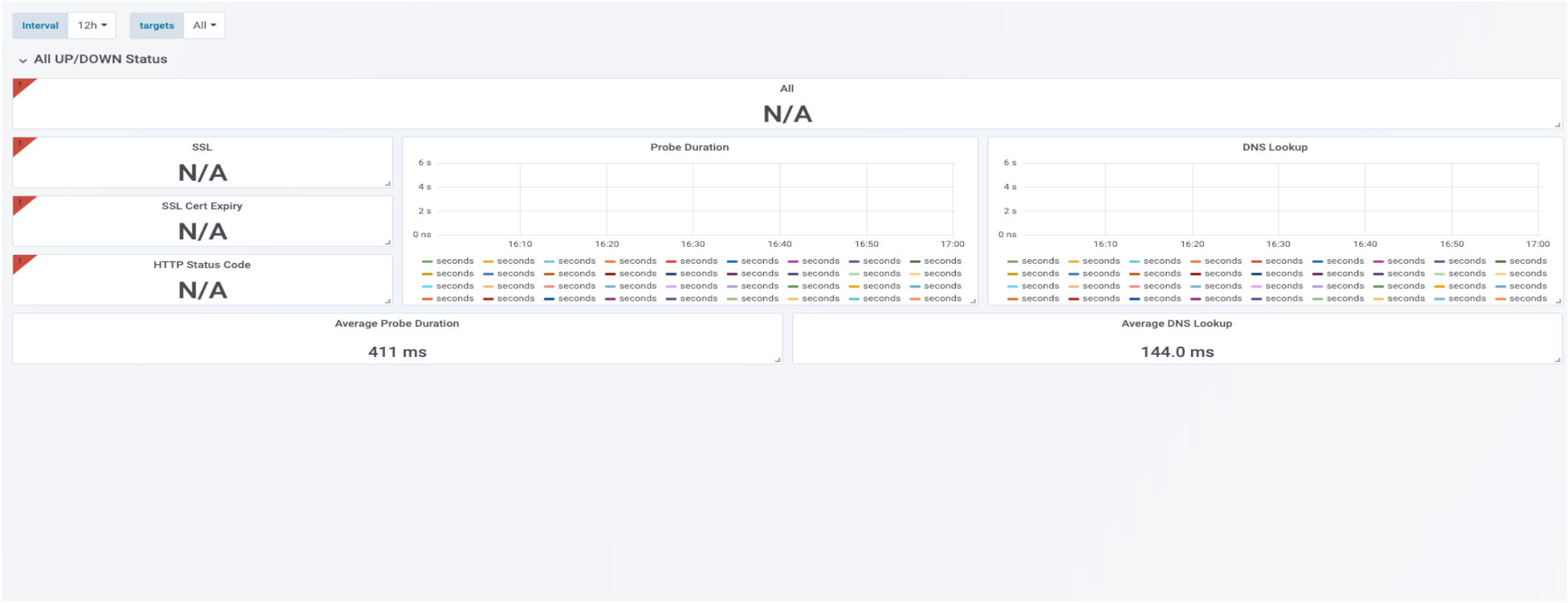
Display - Grafana: The Good

- Best Option
- Well Used Prometheus Datasouce
  - Well tested
  - Some Smart features: Annotations, Variables, Prompting
- Also has Alertmanager Datasource

Display – Grafana: The Bad

- Over 800 publicly shared dashboards that use Prometheus.
- Unfortunately quality varies a lot.
- Sometimes broken by new exporters or prometheus versions
- Don’t match you architecture / naming
- Sometimes just buggy
- Like to reload stats every 10s
- Lack sample picture (so must be downloaded to evaluate)

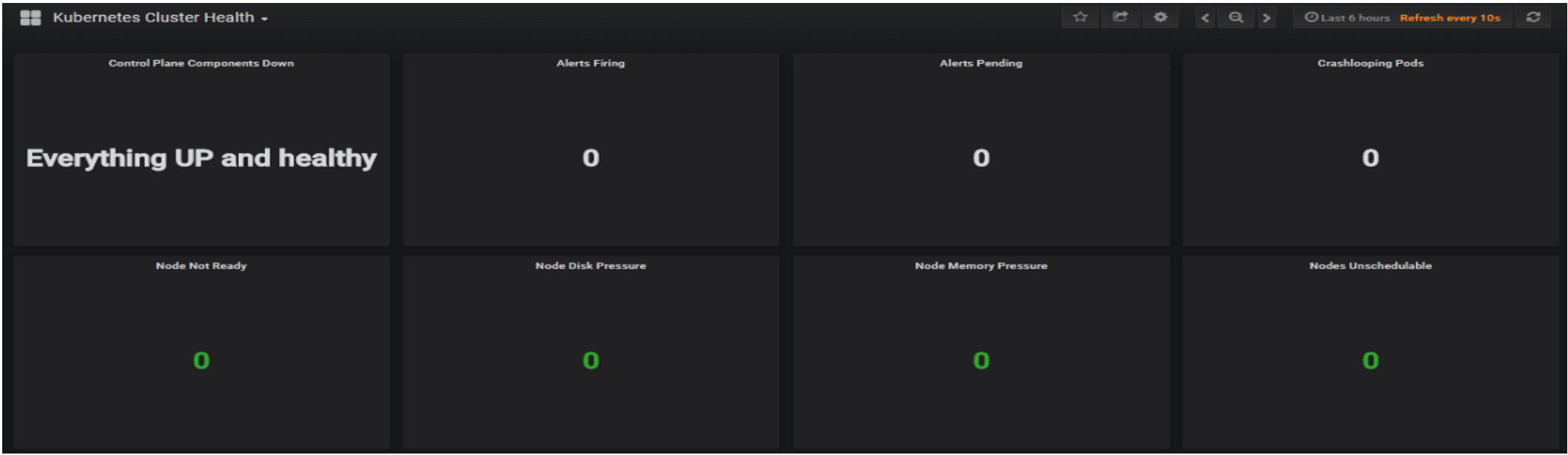
Display – Grafana: The Ugly



Display – Getting Grafana to work

- Just download a bunch and try them out
- But may be a good source for ideas
- Or even be easy to fix
- Check the layout, queries inside.

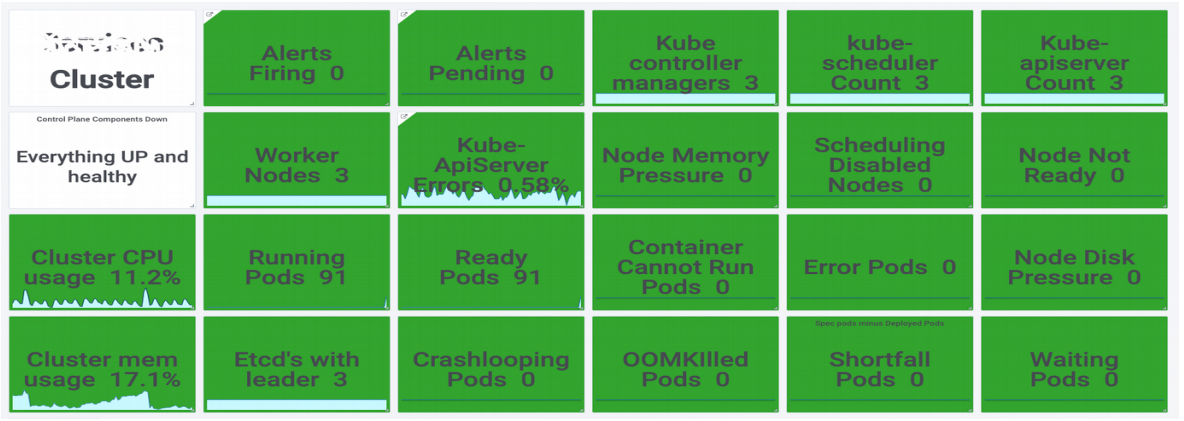
Display: Grafana Before



Displays: Grafana After

Display – Arranging: Overview

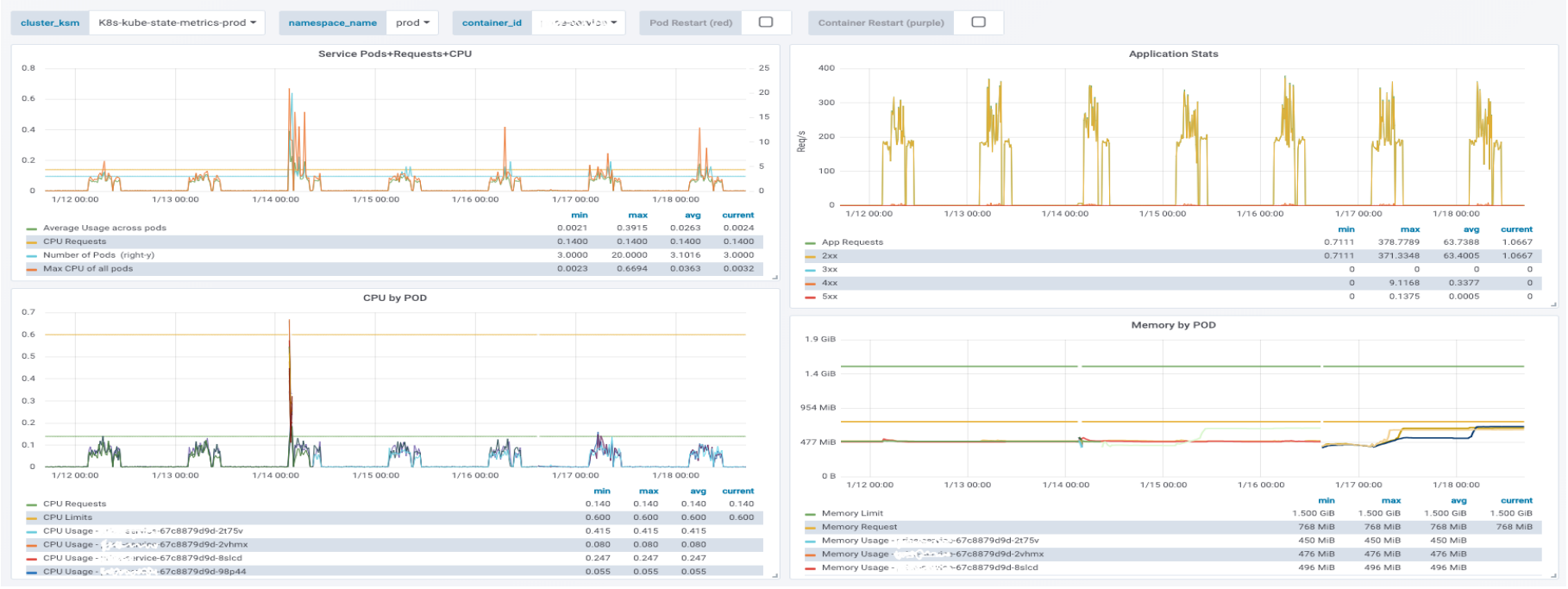
- Overview dashboards
- Good for big screens
- Show summary



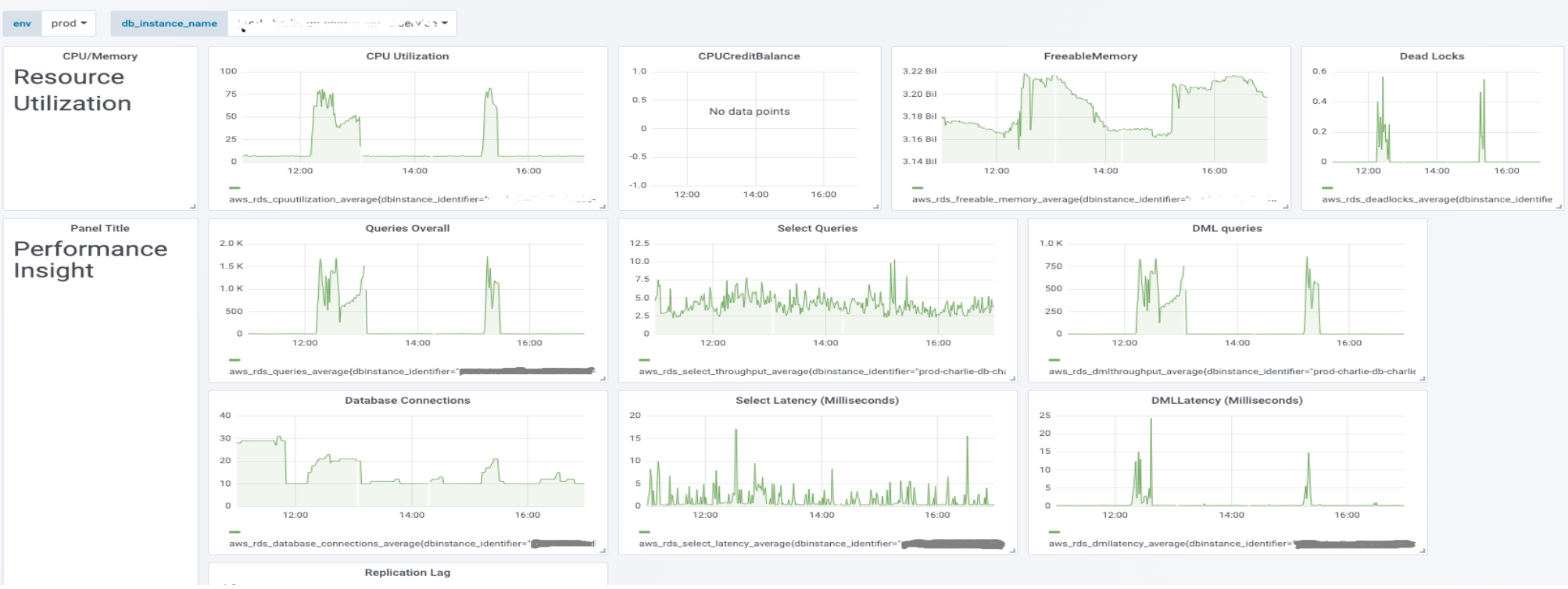
Display: Arranging Drill down

- Drill-down dashboards
- You will be using interactively to change targets, time periods
- Perhaps linked off of summary dashboards

Display


cluster_ksm
K8s-kube-state-metrics-prod
namespace_name
prod
container_id
Pod Restart (red)
Container Restart (purple)
Service Pods+Requests+CPU
1/12 00:00
1/13 00:00
1/14 00:00
1/15 00:00
1/16 00:00
1/17 00:00
1/18 00:00
min max avg current
Average Usage across pods 0.0021 0.3915 0.0263 0.0024
CPU Requests 0.1400 0.1400 0.1400 0.1400
Number of Pods (right-y) 3.0000 20.0000 3.1016 3.0000
Max CPU of all pods 0.0023 0.6694 0.0363 0.0032
Application Stats
Req's
min max avg current
App Requests 0.7111 378.7789 63.7388 1.0667
2xx 0.7111 371.3348 63.4005 1.0667
3xx 0 0 0 0
4xx 0 9.1168 0.3377 0
5xx 0 0.1375 0.0005 0
CPU by POD
min max avg current
CPU Requests 0.140 0.140 0.140 0.140
CPU Limits 0.600 0.600 0.600 0.600
CPU Usage - 67c8879d9d-2t75v 0.415 0.415 0.415
CPU Usage - 67c8879d9d-2vhmx 0.080 0.080 0.080
CPU Usage - 67c8879d9d-8slcd 0.247 0.247 0.247
CPU Usage - 67c8879d9d-98p44 0.055 0.055 0.055
Memory by POD
1.9 GiB
1.4 GiB
954 MiB
477 MiB
0 B
min max avg current
Memory Limit 1.500 GiB 1.500 GiB 1.500 GiB 1.500 GiB
Memory Request 768 MiB 768 MiB 768 MiB 768 MiB
Memory Usage - 67c8879d9d-2t75v 450 MiB 450 MiB 450 MiB
Memory Usage - 67c8879d9d-2vhmx 476 MiB 476 MiB 476 MiB
Memory Usage - 67c8879d9d-8slcd 496 MiB 496 MiB 496 MiB

Display


env
prod
db_instance_name
CPU/Memory
Resource Utilization
CPU Utilization
CPUCreditBalance
No data points
FreeableMemory
Dead Locks
Panel Title
Performance Insight
Queries Overall
Select Queries
DML queries
Database Connections
Select Latency (Milliseconds)
DMLLatency (Milliseconds)
Replication Lag

Display: Unsolved

- You are collecting hundreds of metrics from each server. You can’t display them all
- Even on a “one server per page drill-down”
- You don’t even know which metrics you should be looking at

Display

- Small
  - Prebuilt
- Medium
  - Mostly prebuilt
- Big
  - Hand-rolled summaries
  - Drilldowns
  - Dashboards for different teams
  - Automatically created Dashboards

Extra: RED and USE

- USE – Resource Scope
  - Utilisation - the average time that the resource was busy servicing work
  - Saturation - the degree to which the resource has extra work which it can't service, often queued
  - Errors – Count of error events
- RED – Request Scoped
  - Rate – Rate of requests
  - Errors – Rate of errors in requests
  - Duration - Distribution

Extra: Self Monitoring

- Run a continuous check against external monitoring site
- External monitoring site alerts of connections not received.
- Options
  - DeadMansSnitch.com
  - Healthcheck.io
  - Some Incident Management vendors

Small Site

- Single Instance of everything
- Config in git
- Backup data regularly. Rollback to backup in event of outage
- Use Free/No/Cheap Incident Management
- Free healthcheck.io failure checking
- Minimal exporters

Medium Site

- Some duplicate/redundant instances
- Config in git, Have some templates
- Backup data regularly. Flip to 2nd instance in event of outage.
- Use Incident Management Vendor
- Paid failure checking or rely on redundancy
- Instrument as much as possible

Large Site

- Duplicate/redundant instances, Scale Horizontally
- Template and autodiscovery everywhere
- External clustered storage
- Use Incident Management Vendor
- Cross monitoring
- Look at filtering what you keep.