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

<table>
<thead>
<tr>
<th>Name</th>
<th>Labels</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_cpu{cpu=&quot;cpu0&quot;,mode=&quot;idle&quot;}</td>
<td>3.47769314e+06</td>
<td></td>
</tr>
</tbody>
</table>

```text
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 metrized 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
## Alerts

<table>
<thead>
<tr>
<th>Alert Name</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeadMansSwitch</td>
<td>1</td>
</tr>
<tr>
<td>CrashLooping2</td>
<td></td>
</tr>
<tr>
<td>APIHighRequestLatency-Bogus</td>
<td></td>
</tr>
<tr>
<td>BlackBoxAuth0High</td>
<td></td>
</tr>
<tr>
<td>BlackBoxCareerHigh</td>
<td></td>
</tr>
<tr>
<td>CrashLooping1</td>
<td></td>
</tr>
<tr>
<td>CrashLooping3</td>
<td></td>
</tr>
<tr>
<td>CrashLoopingCount1</td>
<td></td>
</tr>
<tr>
<td>CrashLoopingCount2</td>
<td></td>
</tr>
</tbody>
</table>
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 2\textsuperscript{nd} 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 Datasource
  - 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

<table>
<thead>
<tr>
<th>Service</th>
<th>Status</th>
<th>Probe Duration</th>
<th>DNS Lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL Cert Expiry</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP Status Code</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Average Probe Duration**: 411 ms
- **Average DNS Lookup**: 144.0 ms
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
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
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 2\textsuperscript{nd} 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.