Monitoring Kubernetes with Prometheus

Tom Wilkie, July 2017

Grafana Labs

Tom Wilkie *VP Product, Grafana Labs*
Previously*:* Kausal, Weaveworks, Google, Acunu, Xensource
Twitter: @tom_wilkie Email: tom@grafana.com

Grafana Labs

Prometheus Kubernetes Monitoring & Alerting Getting Started

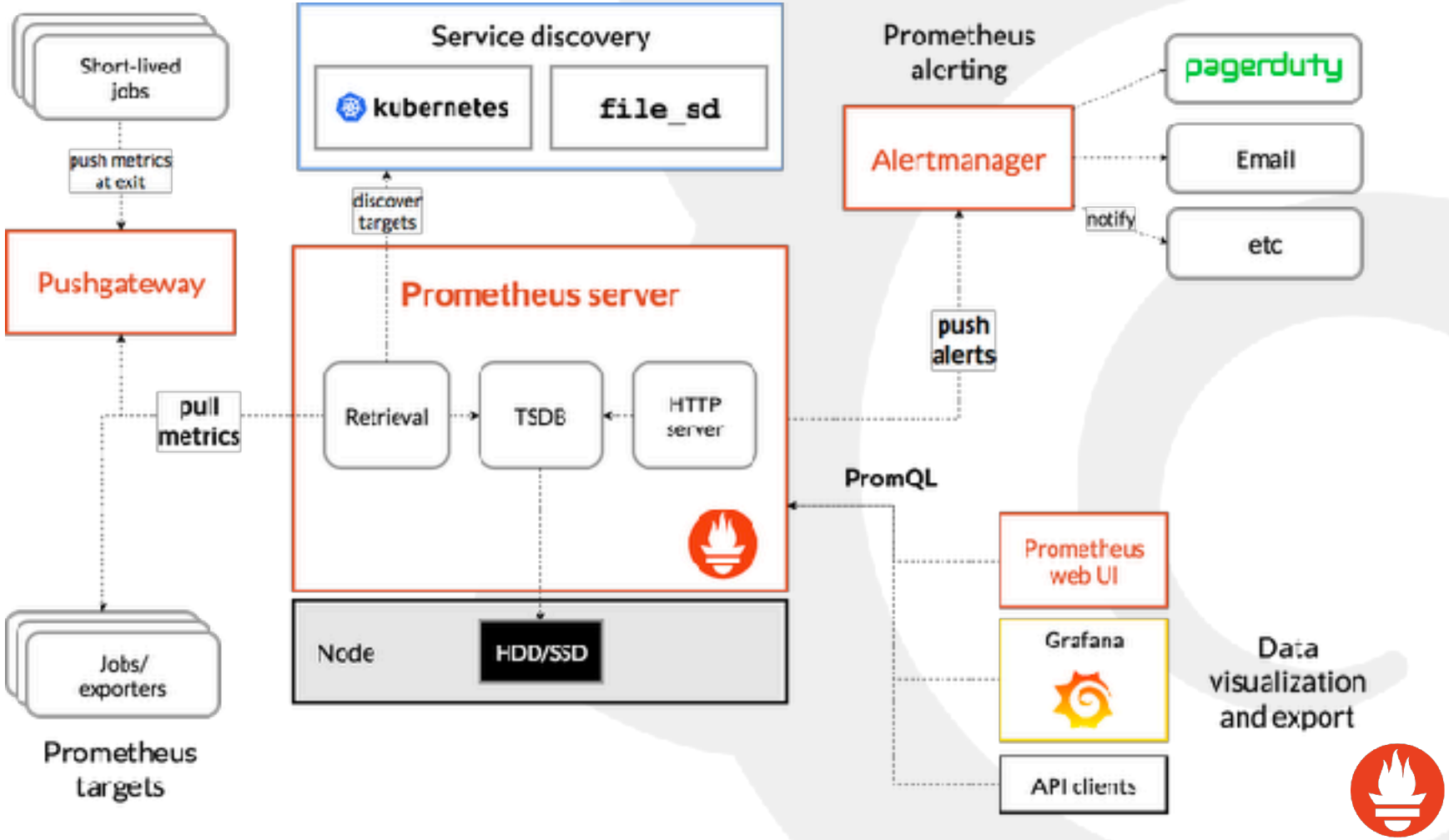
Grafana Labs

Prometheus

- A monitoring & alerting system.
- Inspired by Google's BorgMon
- Originally built by SoundCloud in 2012
- Open Source, now part of the CNCF

- Simple text-based metrics format
- Multidimensional datamodel
- Rich, concise query language

Grafana Labs


Short-lived jobs
push metrics at exit
Pushgateway
Service discovery
kubernetes
file_sd
discover targets
Prometheus server
Retrieval
TSDB
HTTP server
pull metrics
Node
HDD/SSD
Jobs/ exporters
Prometheus targets
Prometheus alerting
Alertmanager
pagerduty
Email
etc
notify
push alerts
PromQL
Prometheus web UI
Grafana
API clients
Data visualization and export

localhost:8080/metrics

localhost:8080/metrics

```
# HELP cortex_ring_ingester_ownership_percent The percent ownership of the ring by ingester
# TYPE cortex_ring_ingester_ownership_percent gauge
cortex_ring_ingester_ownership_percent{ingester="ingester-7677ff7766-6nm68"} 0.139281139974315
cortex_ring_ingester_ownership_percent{ingester="ingester-7677ff7766-7hq5d"} 0.1420537524721710 3
cortex_ring_ingester_ownership_percent{ingester="ingester-7677ff7766-845ng"} 0.1458392255347779
cortex_ring_ingester_ownership_percent{ingester="ingester-7677ff7766-8h25r"} 0.1368029604518792 9
cortex_ring_ingester_ownership_percent{ingester="ingester-7677ff7766-bpxc4"} 0.1424883066542652 3
cortex_ring_ingester_ownership_percent{ingester="ingester-7677ff7766-mlktg"} 0.1536875248778815
cortex_ring_ingester_ownership_percent{ingester="ingester-7677ff7766-rmp28"} 0.1398470900347100 2
# HELP cortex_ring_ingesters Number of ingesters in the ring
# TYPE cortex_ring_ingesters gauge
cortex_ring_ingesters{state="ACTIVE"} 7
cortex_ring_ingesters{state="JOINING"} 0
cortex_ring_ingesters{state="LEAVING"} 0
cortex_ring_ingesters{state="PENDING"} 0
cortex_ring_ingesters{state="Unhealthy"} 0
# HELP cortex_ring_tokens Number of tokens in the ring
# TYPE cortex_ring_tokens gauge
cortex_ring_tokens 3584
# HELP go_gc_duration_seconds A summary of the GC invocation durations.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0.000143842
go_gc_duration_seconds{quantile="0.25"} 0.000270938
go_gc_duration_seconds{quantile="0.5"} 0.000337269
go_gc_duration_seconds{quantile="0.75"} 0.000486132
```

Prometheus' data model is very simple:

```
<identifier> → [ (t0, v0), (t1, v1), ... ]
```

Timestamps are millisecond int64, values are float64

https://www.slideshare.net/Docker/monitoring-the-prometheus-way-julius-voltz-prometheus

Prometheus identifiers

```
http_requests_total{job="nginx", instances="1.2.3.4:80", path="/home", status="200"}
http_requests_total{job="nginx", instances="1.2.3.4:80", path="/home", status="500"}
http_requests_total{job="nginx", instances="1.2.3.4:80", path="/settings", status="200"}
http_requests_total{job="nginx", instances="1.2.3.4:80", path="/settings", status="502"}
```

Prometheus series selector

```
http_requests_total{job="nginx", status=~"5.."}
```

Building queries usually starts with a selector

PromQL: `http_requests_total{job="nginx", status=~"5.."}`

```
{job="nginx", instances="1.2.3.4:80", path="/home", status="500"}     34
{job="nginx", instances="1.2.3.4:80", path="/settings", status="502"} 56
{job="nginx", instances="2.3.4.5:80", path="/home", status="500"}     76
{job="nginx", instances="2.3.4.5:80", path="/setting", status="502"}  96
...
```

Can select vectors of values…

PromQL: `http_requests_total{job="nginx", status=~"502"}`**`[1m]`**

```
{job="nginx", instances="1.2.3.4:80", path="/home", status="500"}     [30, 31, 32, 34]
{job="nginx", instances="1.2.3.4:80", path="/settings", status="500"} [4, 24, 56, 56]
{job="nginx", instances="2.3.4.5:80", path="/home", status="500"}     [76, 76, 76, 76]
{job="nginx", instances="2.3.4.5:80", path="/setting", status="500"}  [56, 106, 5, 96]
...
```

And apply functions…

PromQL: `rate(http_requests_total{job="nginx", status=~"502"}[1m])`

```
{job="nginx", instances="1.2.3.4:80", path="/home", status="500"}     0.0666
{job="nginx", instances="1.2.3.4:80", path="/settings", status="500"} 0.866
{job="nginx", instances="2.3.4.5:80", path="/home", status="500"}     0.0
{job="nginx", instances="2.3.4.5:80", path="/settings", status="500"} 2.43
...
```

And aggregate by a dimension…

PromQL: **sum by (path)** (rate(http_requests_total{job="nginx", status=~"502"}[1m]))

```
{path="/home"}     0.0666
{path="/settings"} 3.3
...
```

Do binary operations…

PromQL:
```
sum by (path) (rate(http_requests_total{job="nginx", status=~"502"}[1m]))
/
sum by (path) (rate(http_requests_total{job="nginx"}[1m]))
```

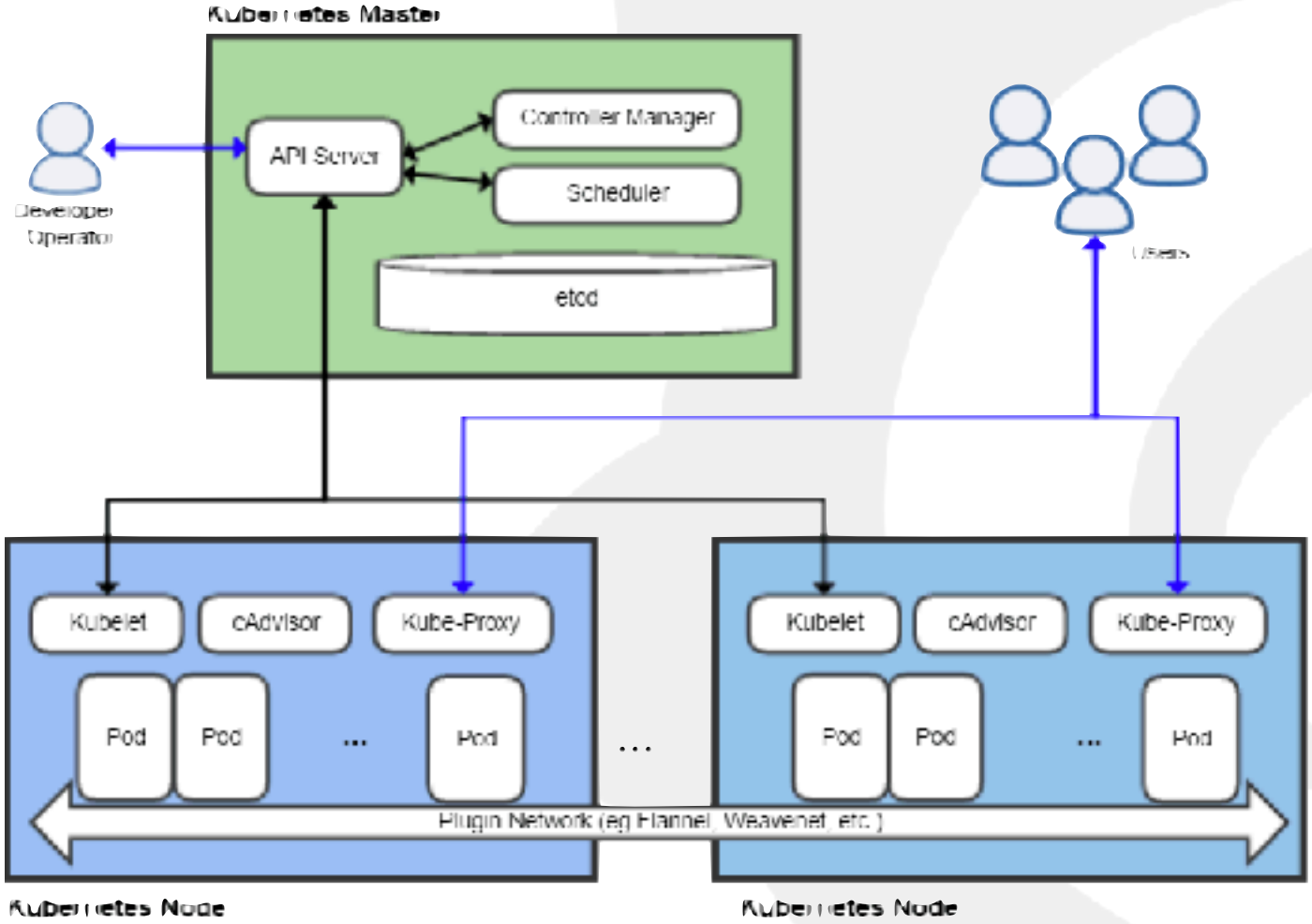
```
{path="/home"}     0.001
{path="/settings"} 1.0
...
```

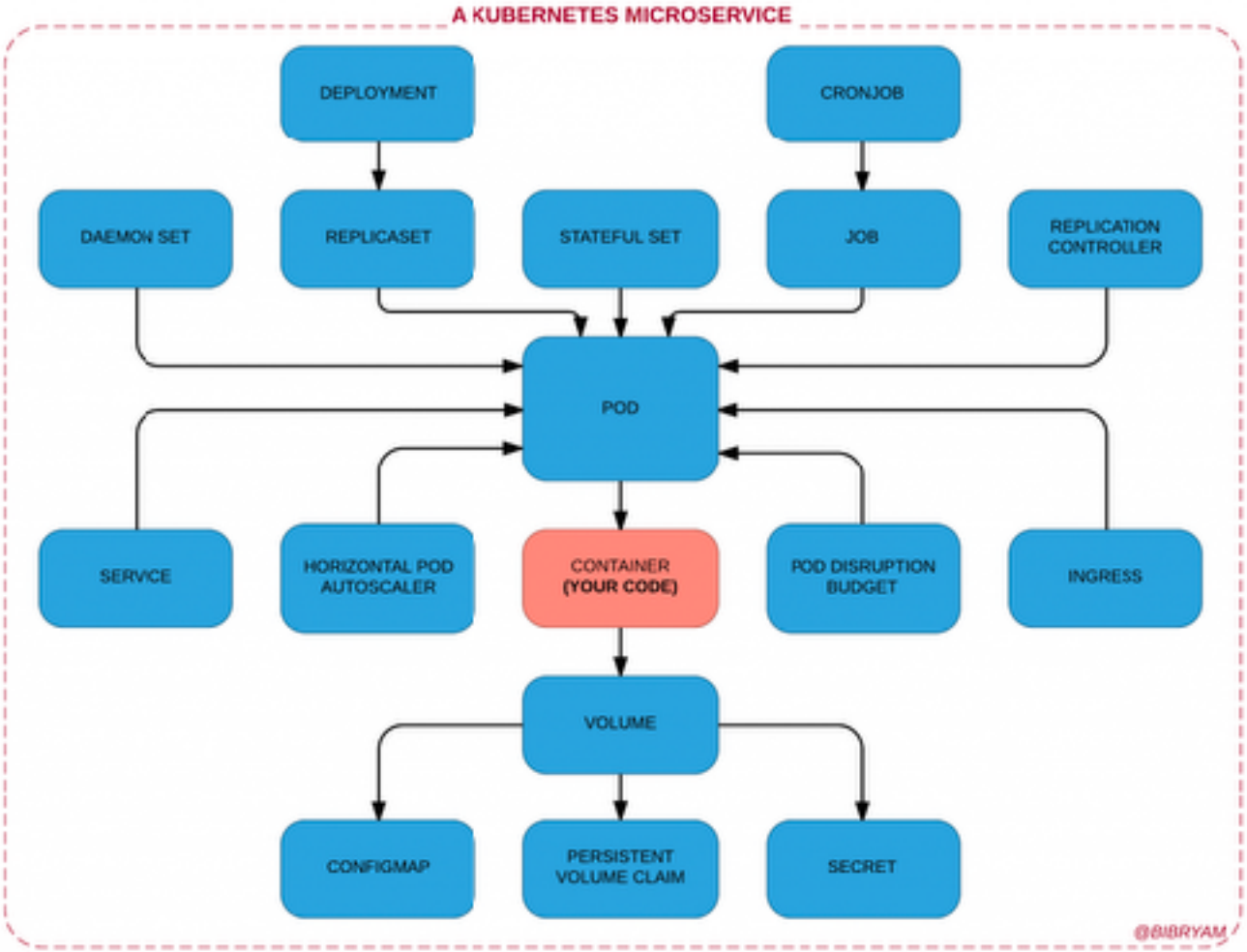
Kubernetes

- Platform for managing containerized workloads and services
- “operating system for you datacenter”
- Inspired by Google’s Borg
- Also part of the CNCF

- Distributed, fault tolerant architecture
- Rich object model for you applications

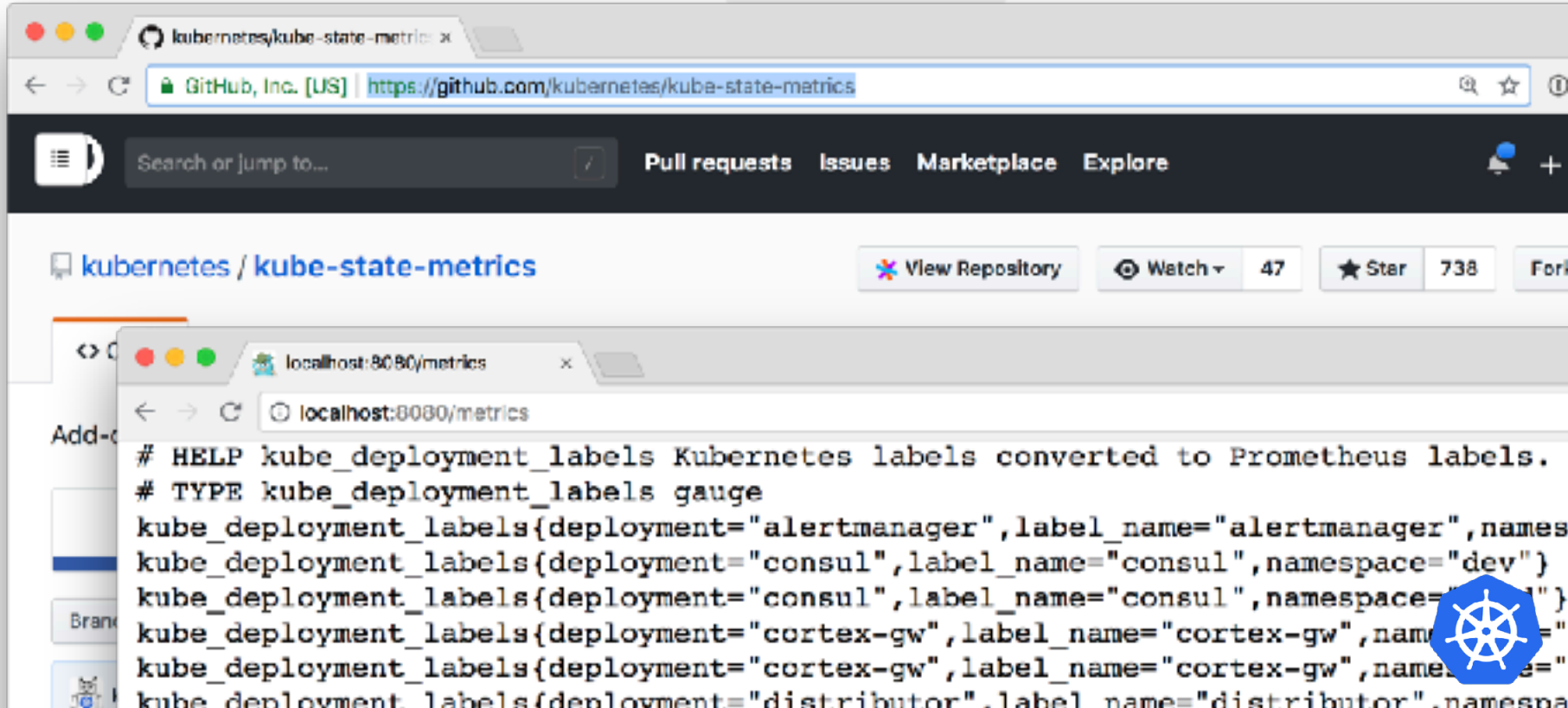
Grafana Labs


Kubernetes Master
API Server
Controller Manager
Scheduler
etcd
Developer
Operator
Users
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
...
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Plugin Network (eg Flannel, Weavenet, etc.)
Kubernetes Node
Kubernetes Node

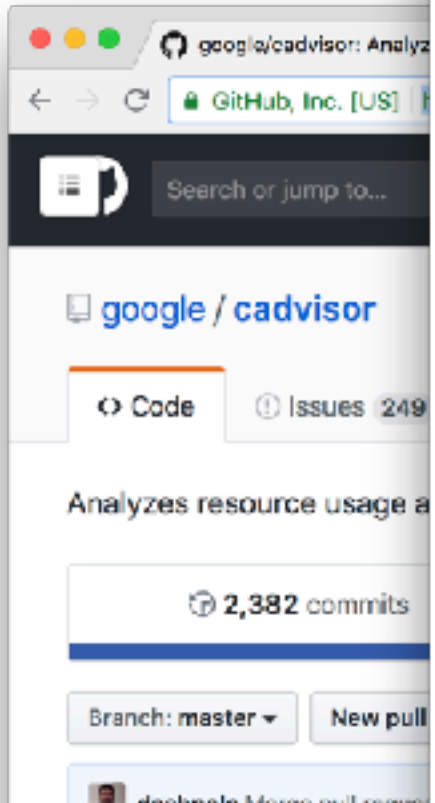


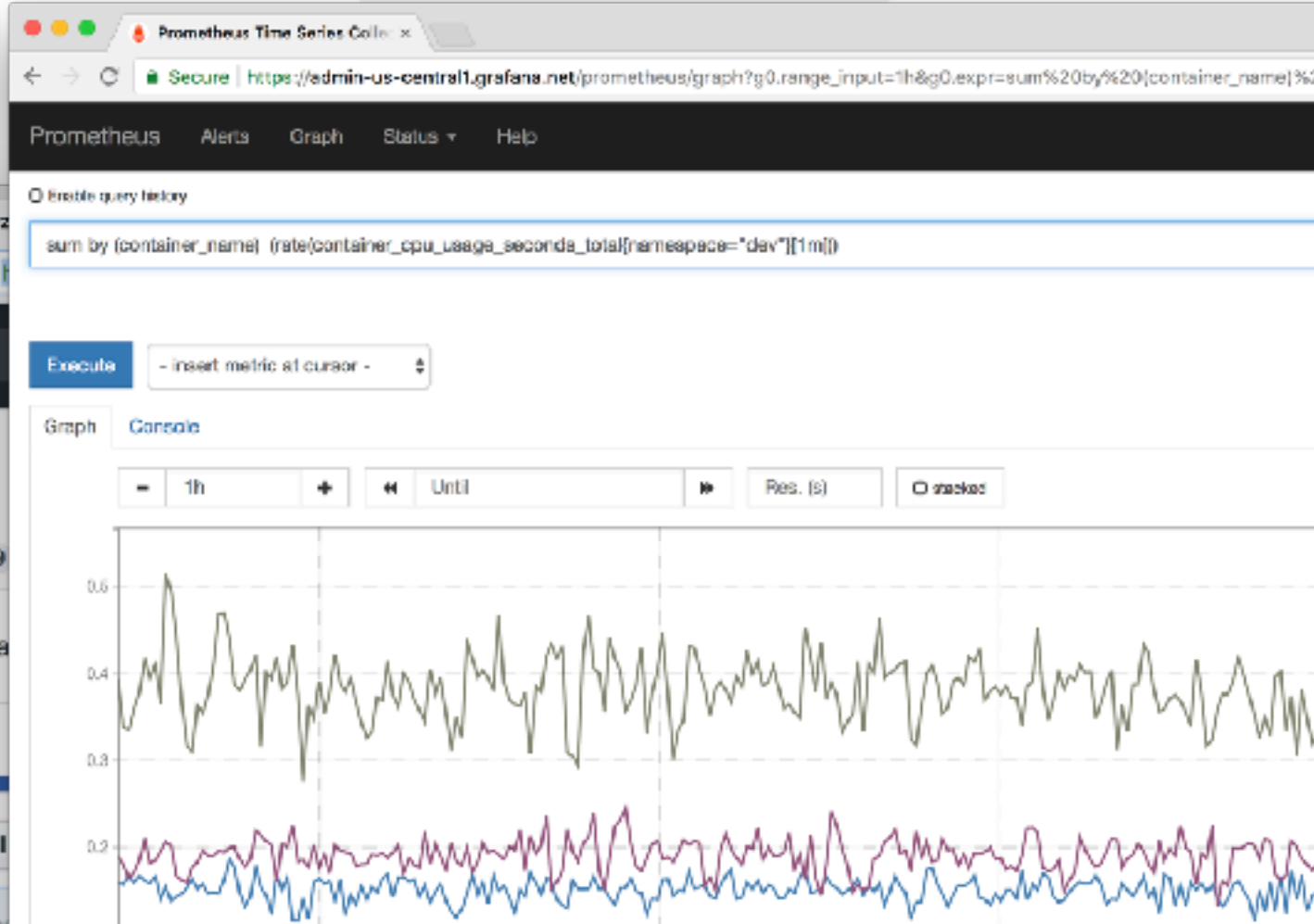
https://thenewstack.io/myth-cloud-native-portability/

kube-state-metrics



cAdvisor


google/cadvisor: Analyz
GitHub, Inc. [US]
Search or jump to...
google / cadvisor
Code
Issues 249
Analyzes resource usage a
2,382 commits
Branch: master
New pull


Prometheus Time Series Colle
Secure
https://admin-us-central1.grafana.net/prometheus/graph?g0.range_input=1h&g0.expr=sum%20by%20(container_name)%2
Prometheus
Alerts
Graph
Status
Help
Enable query history
sum by (container_name) (rate(container_cpu_usage_seconds_total{namespace="dev"}[1m]))
Execute
- insert metric at cursor -
Graph
Console
1h
Until
Res. (s)
stacked
0.6
0.4
0.2

Monitoring & Alerting

Grafana Labs

What should I monitor?

USE Method

- Utilisation, Saturation, Errors…

RED Method

- Requests, Errors, Duration…

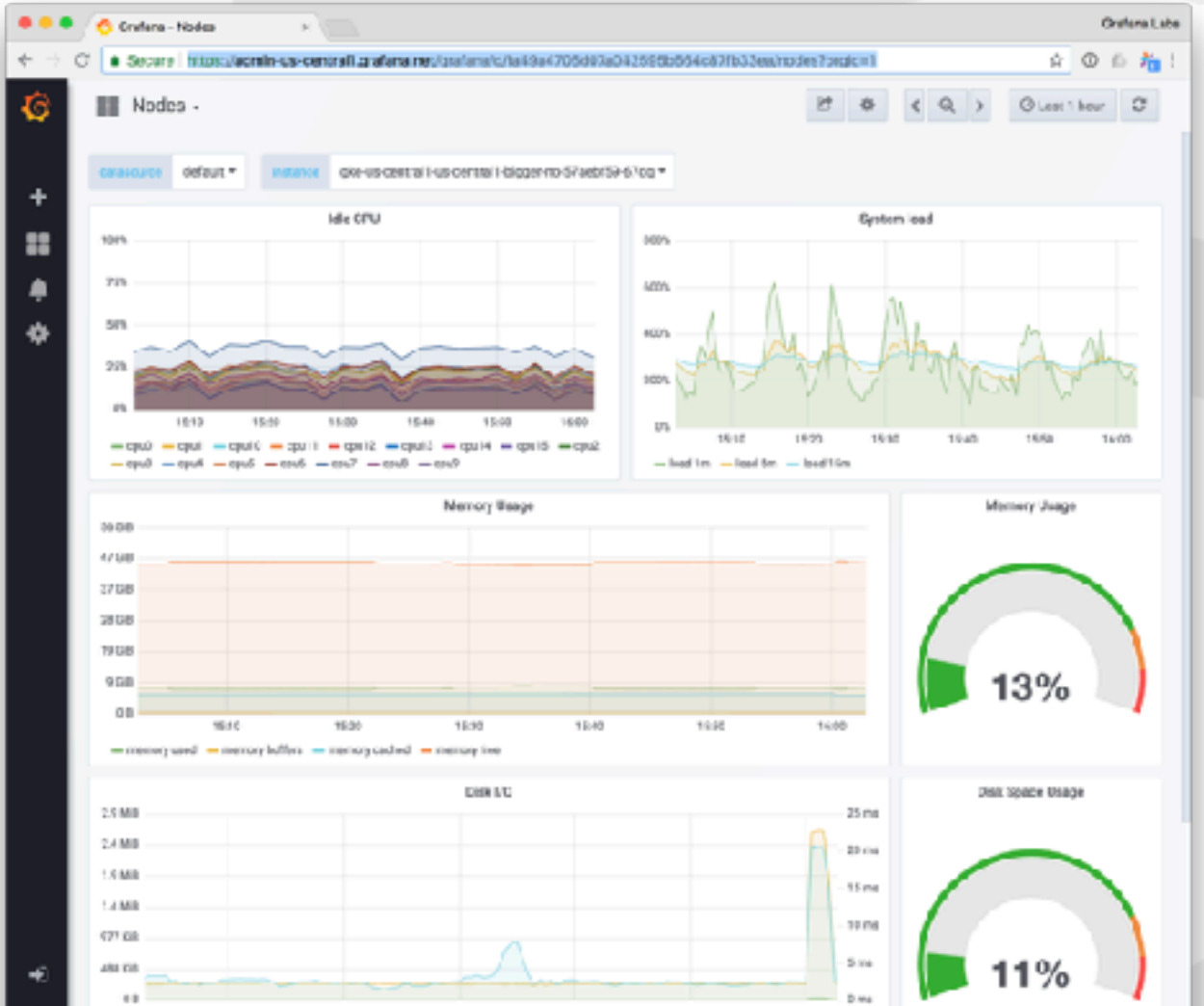
??? Method

- Expected system state…

Grafana Labs

USE Method

- cluster and node level metrics
- node_exporter run as a daemonset



USE Method

CPU Utilisation:

```
1 - avg(rate(node_cpu{mode="idle"}[1m]))
```

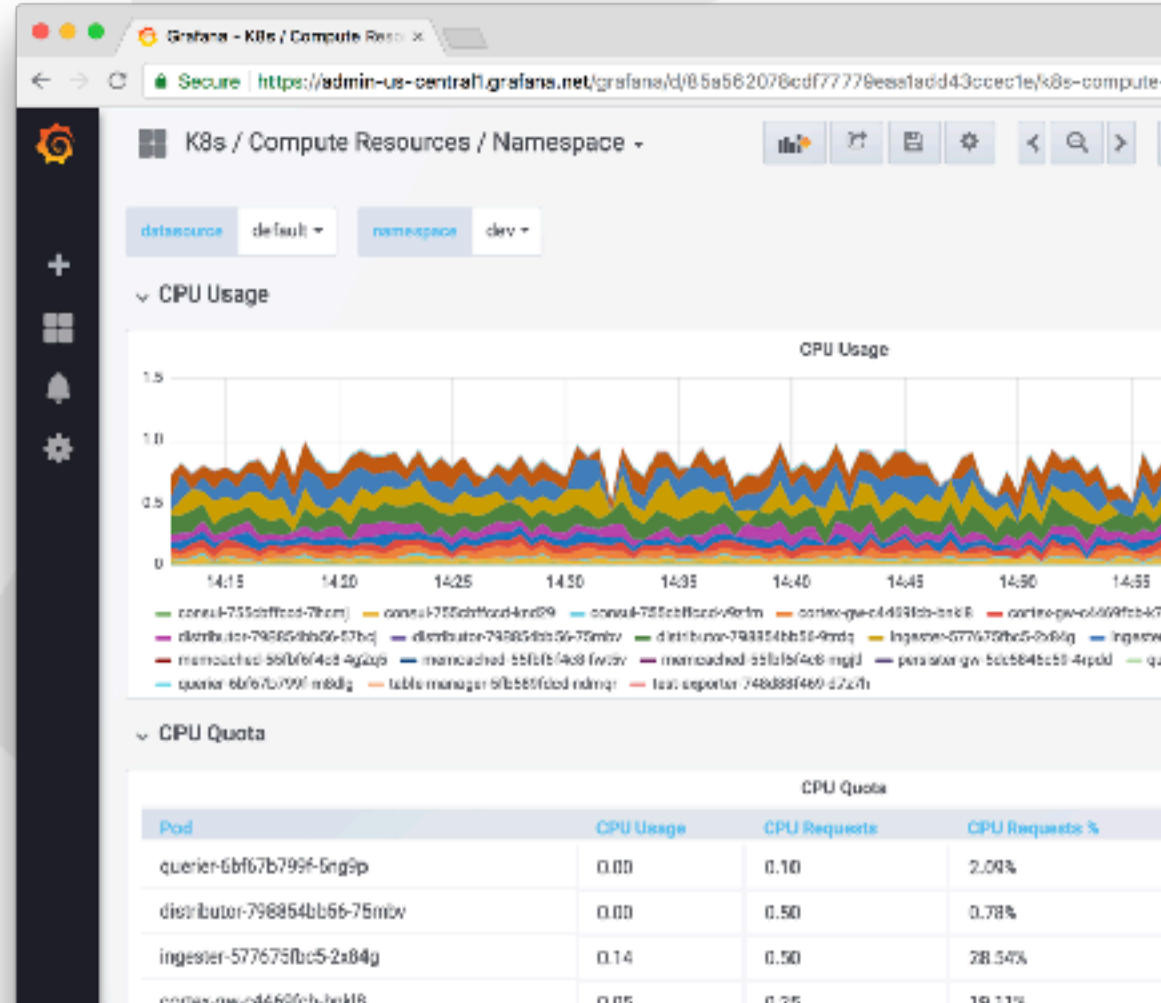
CPU Saturation:

```
sum(node_load1)/ sum(node:node_num_cpu:sum)
```

Grafana Labs

USE Method

- Can also look at container level metrics from cAdvisor…
- …and combine them with metadata from kube-state-metrics.



USE Method

Container CPU usage by “app” label

```
sum by (namespace, label_name) (
      sum(rate(container_cpu_usage_seconds_total[5m])) by (pod
    * on (pod_name) group_left(label_name)
      label_join(kube_pod_labels, "pod_name", ",", "pod")
)
```

Grafana Labs

RED Method

- Metrics exposed by components for RED-style monitoring

RED Method

Most useful alert I've found:

```
  100 * sum by(instance, job) (
    rate(rest_client_requests_total{code!~"2.."}[5m])
  )
/
  sum by(instance, job) (
    rate(rest_client_requests_total[5m])
  )
```

Grafana Labs

??? Method

Alert expressions are invariants that describe a healthy system

```
kube_deployment_spec_replicas !=
kube_deployment_status_replicas_available
```

```
rate(kube_pod_container_status_restarts_total[15m]) > 0
```

Grafana Labs

??? Method

Alert expressions are invariants that describe a healthy system

```
(kube_pod_status_phase{phase!~"Running|Succeeded"}) > 0
```

```
sum(kube_pod_container_resource_requests_cpu_cores)
/ sum(node:node_num_cpu:sum)
>
(count(node:node_num_cpu:sum) - 1)
/ count(node:node_num_cpu:sum)
```

Grafana Labs

Getting Started

Grafana Labs

Getting setup

- github.com/coreos/prometheus-operator - Job to look after running Prometheus on Kubernetes
- github.com/coreos/kube-prometheus - Set of configs for running all there other things you need.

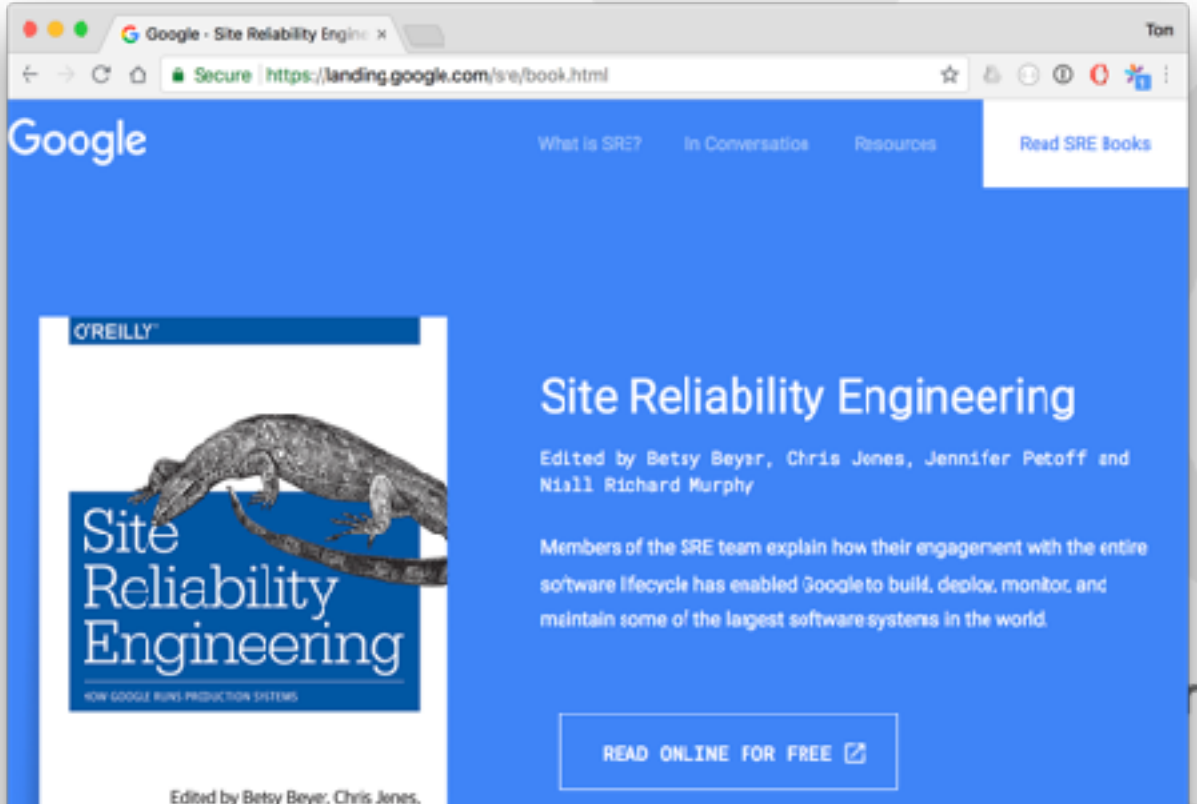
- github.com/kausalco/public/tree/master/prometheus-ksonnet - My configs for running Prometheus, Alertmanager, Grafana etc
- github.com/kubernetes-monitoring/kubernetes-mixin - Joint project to unify and improve common alerts for Kubernetes.

Grafana Labs

More reading…

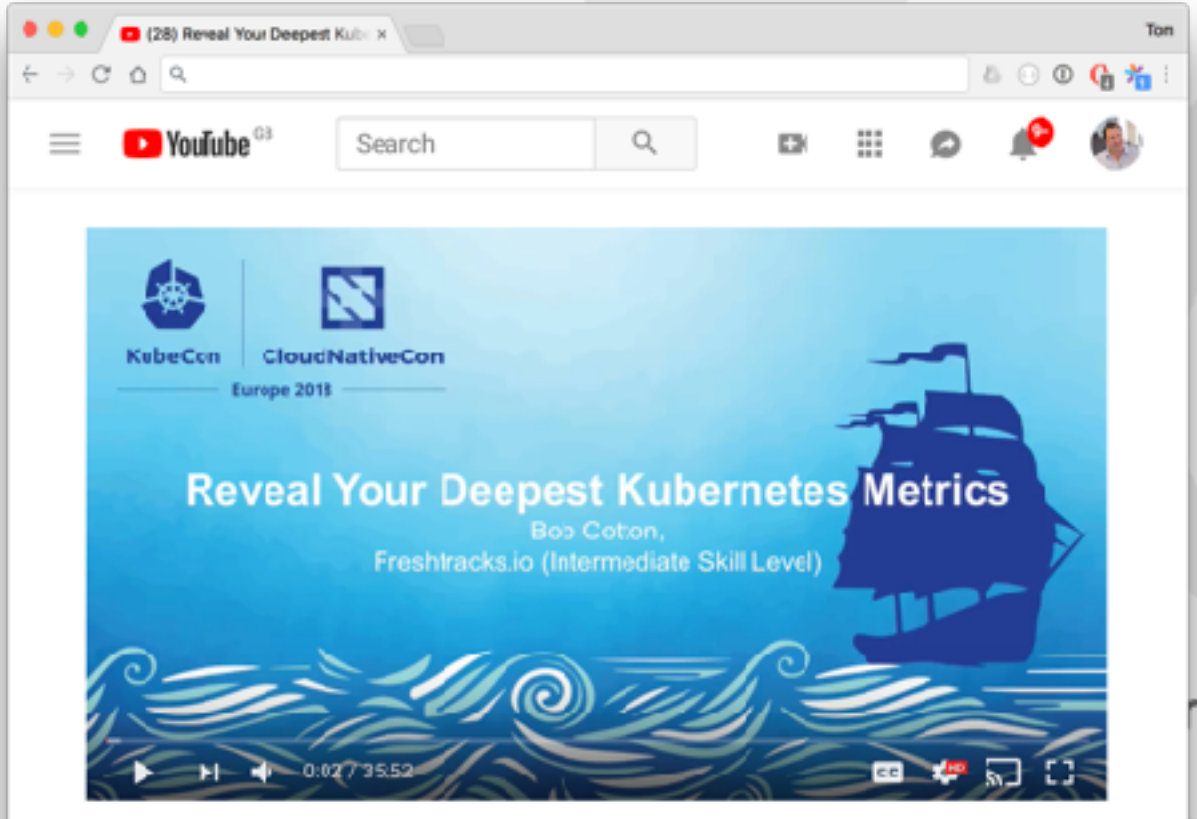
Grafana Labs

https://landing.google.com/sre/book.html



rafana Labs

https://www.youtube.com/watch?v=1oJXMdVi0mM



rafana Labs

http://www.brendangregg.com/usemethod.html

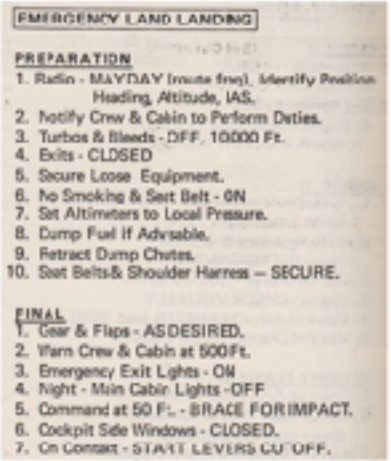
The USE Method

www.brendangregg.com/usemethod.html

The USE Method

The Utilization Saturation and Errors (USE) Method is a methodology for analyzing the performance of any system. It directs the construction of a checklist, which for server analysis can be used for quickly identifying resource bottlenecks or errors. It begins by posing questions, and then seeks answers, instead of beginning with given metrics (partial answers) and trying to work backwards.

The resulting USE Method-derived checklists for different operating systems are listed on the left navigation panel (Linux, Solaris, etc). You can customize these for your environment, adding additional tools that your site uses. There is also the Rosetta Stone of Performance Checklists, automatically generated from some of these. Performance monitoring products can make the USE method easier to follow by providing its metrics via an easy-to-use interface.



EMERGENCY LAND LANDING

PREPARATION
1. Radio - MAYDAY (route freq), Identify Position Heading, Altitude, IAS.
2. Notify Crew & Cabin to Perform Duties.
3. Turbos & Bleeds - OFF, 10,000 Ft.
4. Exits - CLOSED
5. Secure Loose Equipment.
6. No Smoking & Seat Belt - ON
7. Set Altimeters to Local Pressure.
8. Dump Fuel if Advisable.
9. Retract Dump Chutes.
10. Seat Belts & Shoulder Harness – SECURE.

FINAL
1. Gear & Flaps - AS DESIRED.
2. Warn Crew & Cabin at 500 Ft.
3. Emergency Exit Lights - ON
4. Night - Main Cabin Lights - OFF
5. Command at 50 Ft. - BRACE FOR IMPACT.
6. Cockpit Side Windows - CLOSED.
7. On Contact - STATE LEVERS CUT OFF.

Boeing 707 Emergency Checklist (1969)

Intro

rafana Labs

Questions?