Reveal your deepest Kubernetes Metrics

Hello, I’m Lyle Henkeman

- Cloud Engineer @Skyworkz / Big Data Platform Engineer @Nike
- *#k8s #aws #serverless #cicd*
- https://github.com/LyleHenkeman

SKYWORKZ

Getting Started

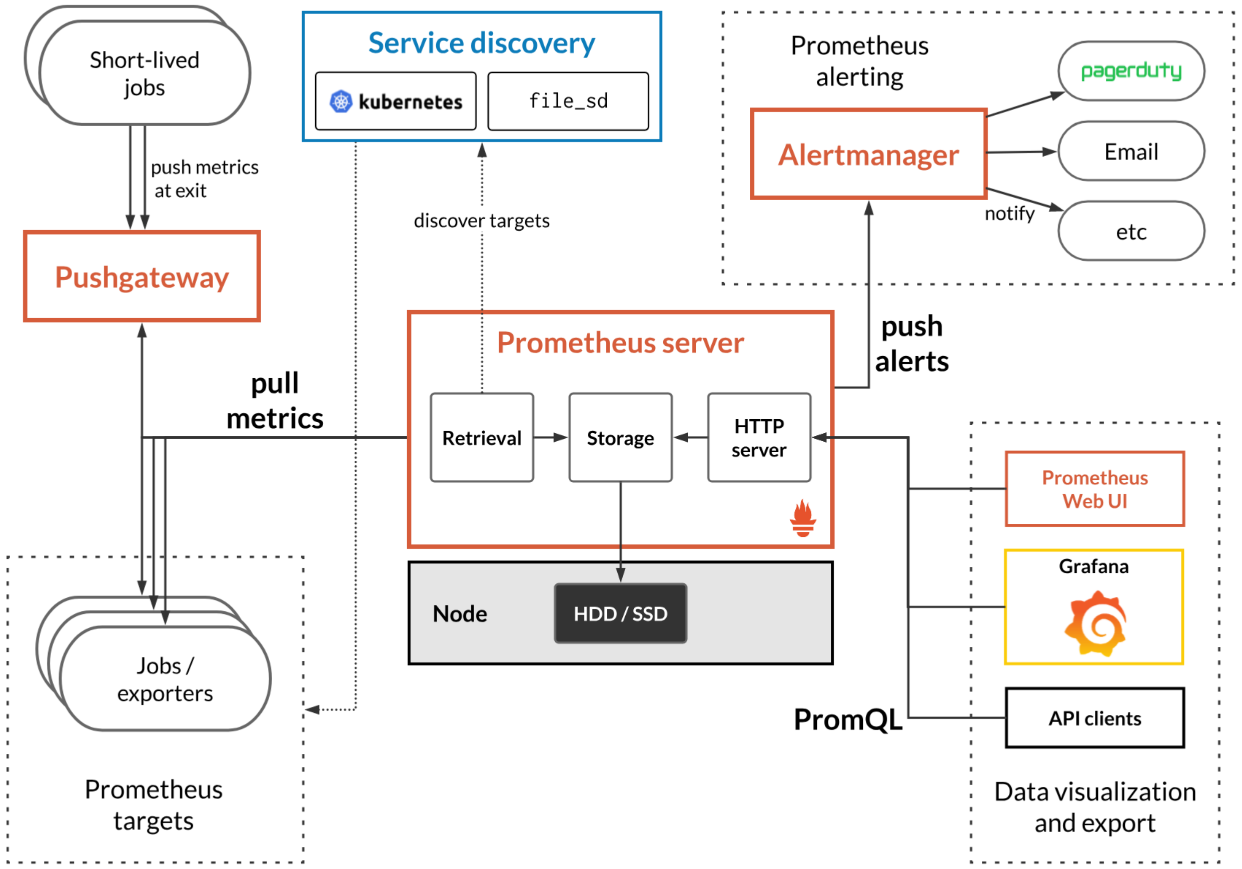
- Prometheus / Grafana
- Kubernetes - Side Cars vs Daemonsets
- Metrics

Prometheus

- A monitoring & alerting system.
- Inspired by Google's BorgMon.
- Originally built by SoundCloud in 2012.
- Open Source.
- Simple text-based metrics format.
- Rich, concise query language.

SKYWORKZ

What does Prometheus look like?



SKYWORKZ

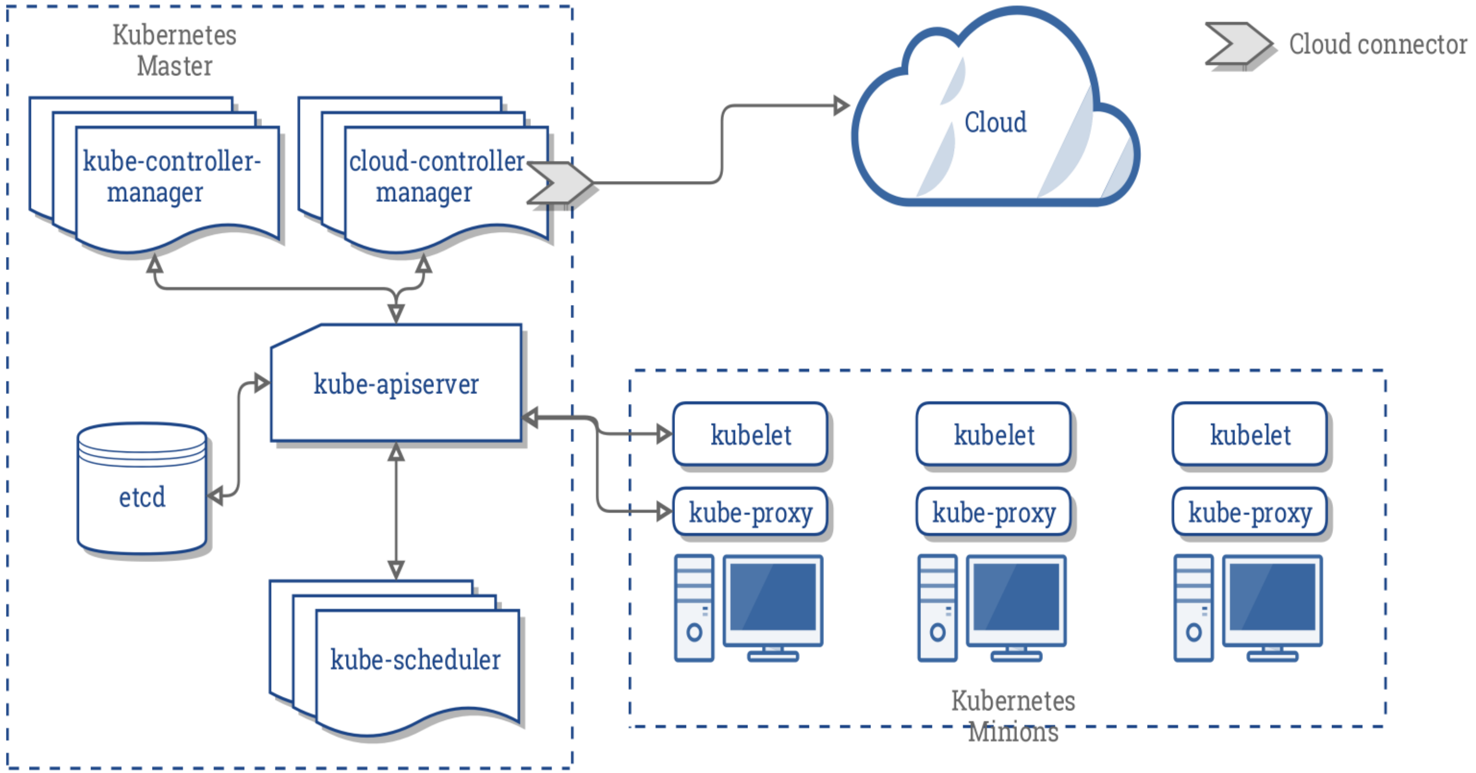
Demo!

SKYWORKZ

Kubernetes

- Container orchestration

Architecture

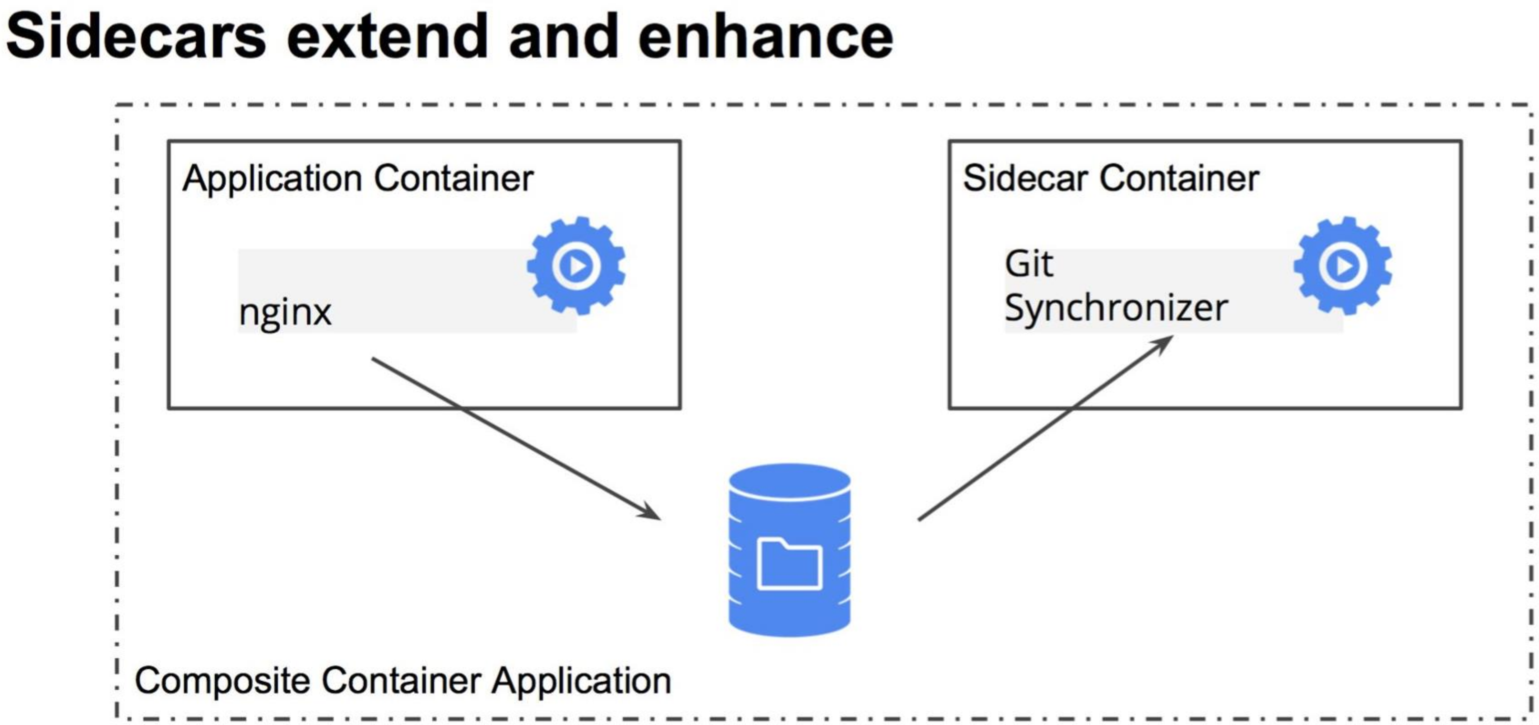


SKYWORKZ

Side Cars

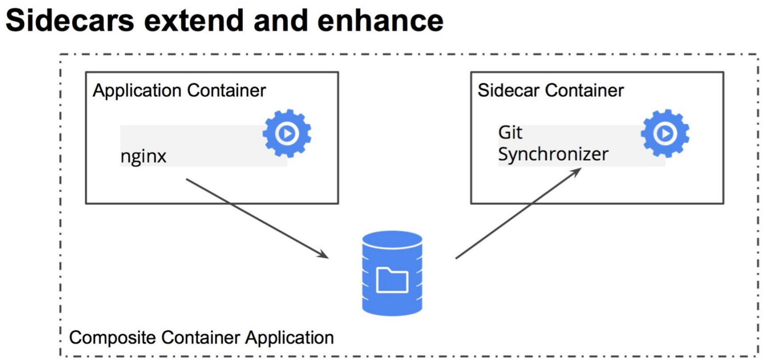
SKYWORKZ

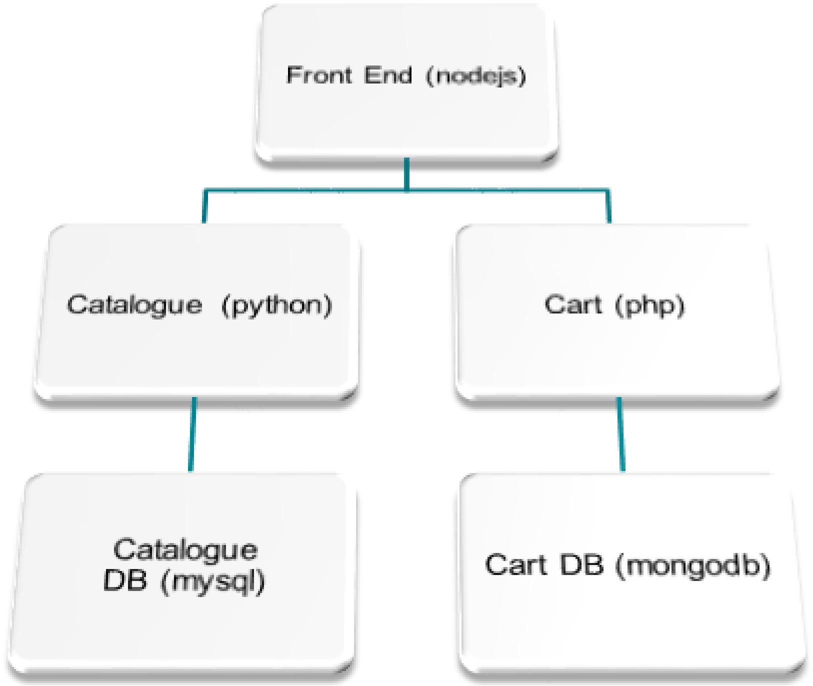
Side Car Pattern



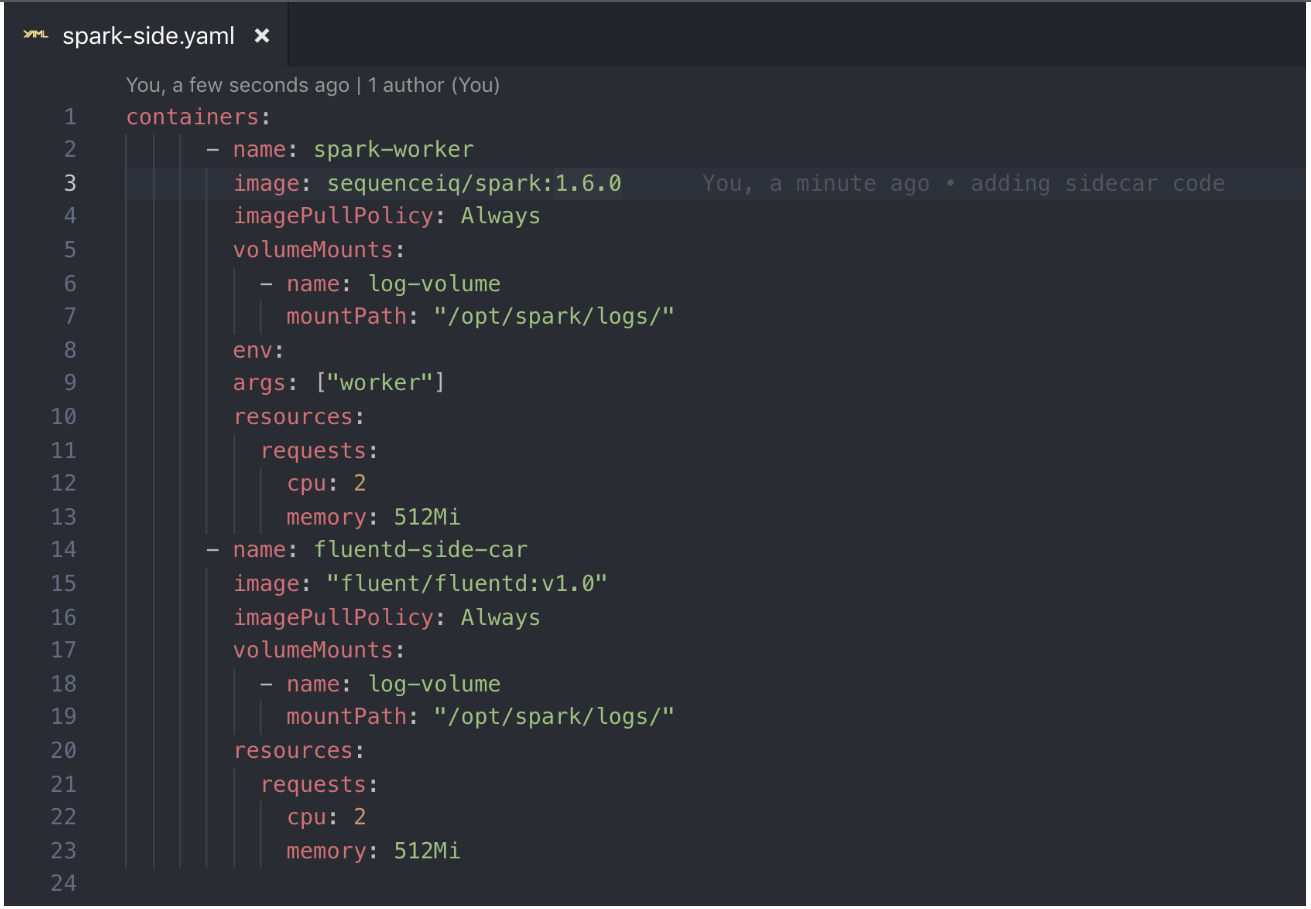
SKYWORKZ

When to use Side Car?





spark-side.yaml



spark-side.yaml ×

You, a few seconds ago | 1 author (You)

```yaml
containers:
    - name: spark-worker
      image: sequenceiq/spark:1.6.0      You, a minute ago • adding sidecar code
      imagePullPolicy: Always
      volumeMounts:
        - name: log-volume
          mountPath: "/opt/spark/logs/"
      env:
      args: ["worker"]
      resources:
        requests:
          cpu: 2
          memory: 512Mi
    - name: fluentd-side-car
      image: "fluent/fluentd:v1.0"
      imagePullPolicy: Always
      volumeMounts:
        - name: log-volume
          mountPath: "/opt/spark/logs/"
      resources:
        requests:
          cpu: 2
          memory: 512Mi

```

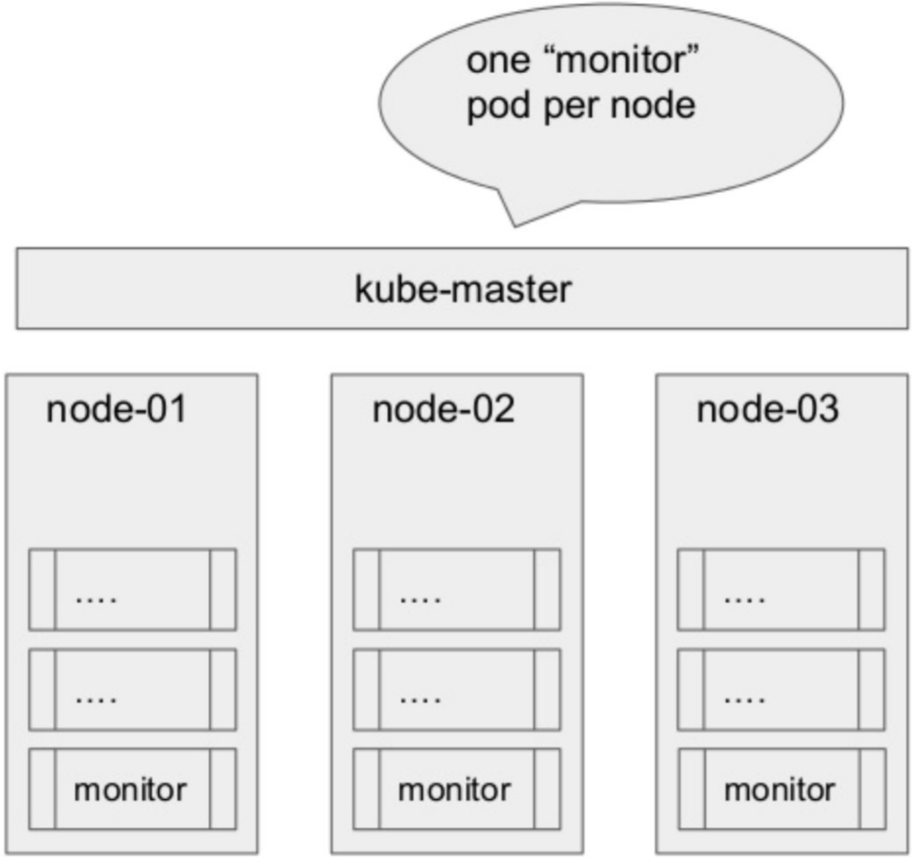
SKYWORKZ

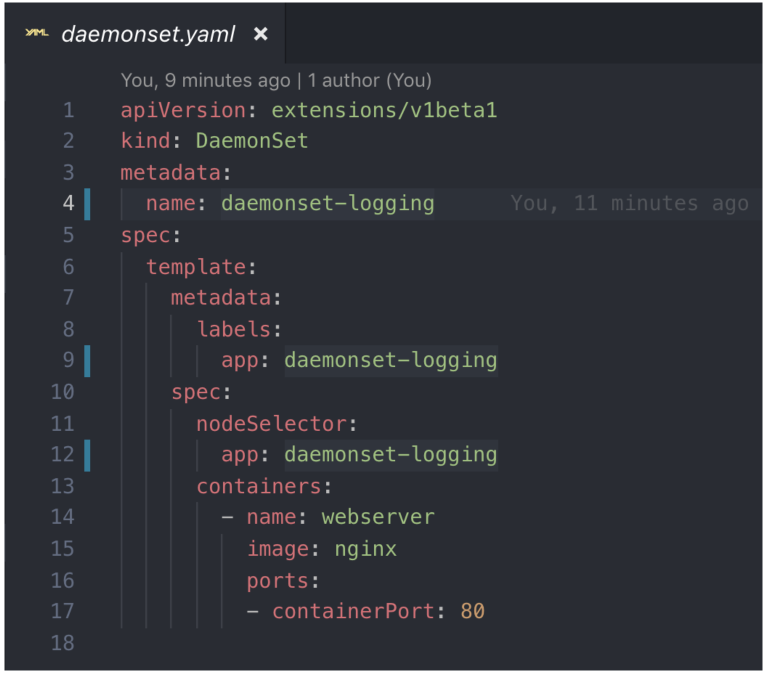
Demo!

Daemon Sets

SKYWORKZ

Daemonset Pattern





```yaml
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: daemonset-logging
spec:
  template:
    metadata:
      labels:
        app: daemonset-logging
    spec:
      nodeSelector:
        app: daemonset-logging
      containers:
        - name: webserver
          image: nginx
          ports:
          - containerPort: 80

```

SKYWORKZ

Demo!

SKYWORKZ

Metrics

SKYWORKZ

What should I monitor?

USE Method

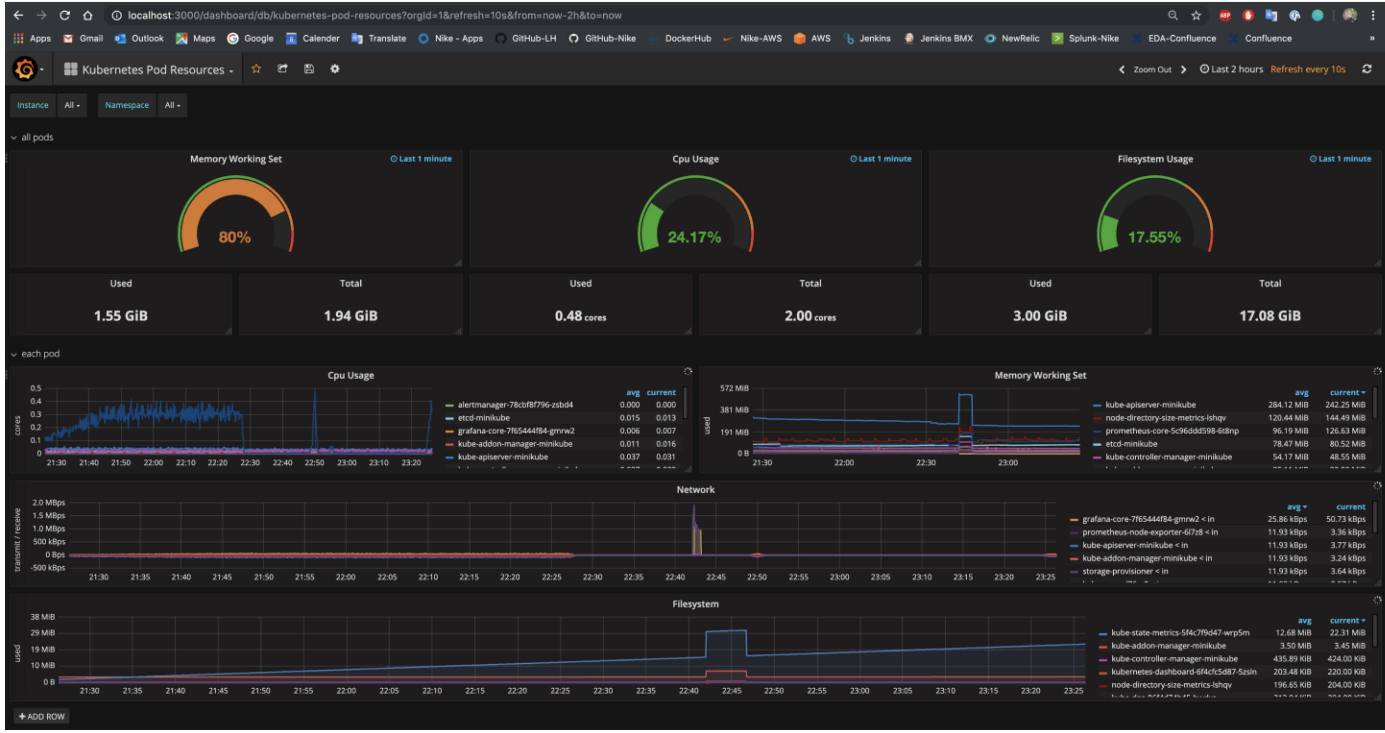
- Utilization- Amount of time that resource is busy, obvious for CPU.
- Saturation - Amount of work that resource had to do.
- Errors - Self-explanatory.

RED Method introduced by Tom Wilkie (Grafana Labs)

- **R**ate - the number of requests, per second, you're services are serving.
- **E**rrors - the number of failed requests per second.
- **D**uration - distributions of the amount of time each request takes.

USE Method

- Cluster and node level metrics
- Node_exporter runs as a daemonset



```
localhost:3000/dashboard/db/kubernetes-pod-resources?orgId=1&refresh=10s&from=now-2h&to=now
```

Kubernetes Pod Resources — Last 2 hours, Refresh every 10s

Instance: All — Namespace: All

all pods

	Used	Total
Memory Working Set (Last 1 minute): 80%	1.55 GiB	1.94 GiB
Cpu Usage (Last 1 minute): 24.17%	0.48 cores	2.00 cores
Filesystem Usage (Last 1 minute): 17.55%	3.00 GiB	17.08 GiB

each pod

Cpu Usage

	avg	current
alertmanager-78cbf8f796-zsbd4	0.000	0.000
etcd-minikube	0.015	0.013
grafana-core-7f65444f84-gmrw2	0.006	0.007
kube-addon-manager-minikube	0.011	0.016
kube-apiserver-minikube	0.037	0.031

Memory Working Set

	avg	current
kube-apiserver-minikube	284.12 MiB	242.25 MiB
node-directory-size-metrics-lshqv	120.44 MiB	144.49 MiB
prometheus-core-5c96ddd598-6t8np	96.19 MiB	126.63 MiB
etcd-minikube	78.47 MiB	80.52 MiB
kube-controller-manager-minikube	54.17 MiB	48.55 MiB

Network

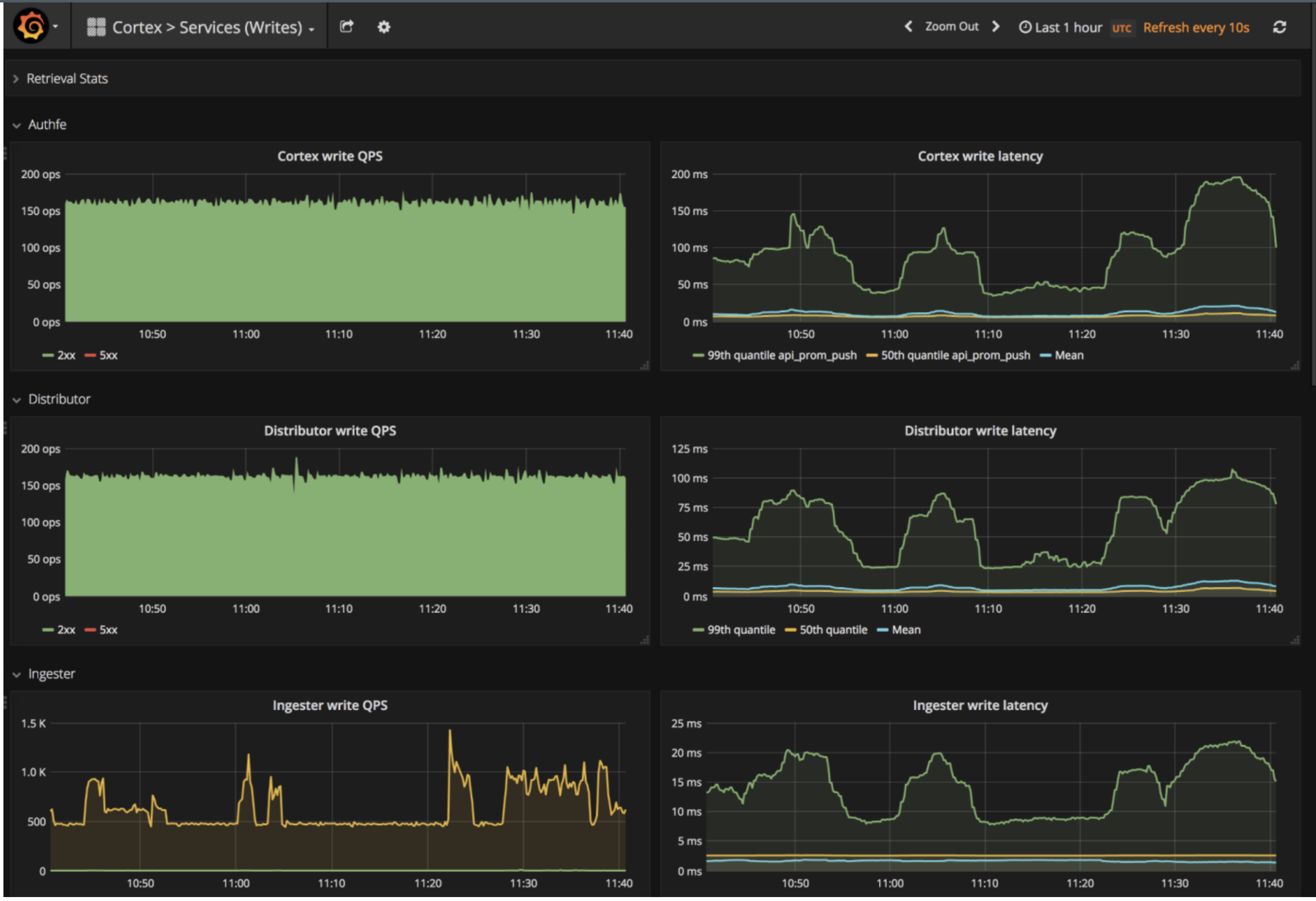
	avg	current
grafana-core-7f65444f84-gmrw2 < in	25.86 kBps	50.73 kBps
prometheus-node-exporter-6l7z8 < in	11.93 kBps	3.36 kBps
kube-apiserver-minikube < in	11.93 kBps	3.77 kBps
kube-addon-manager-minikube < in	11.93 kBps	3.24 kBps
storage-provisioner < in	11.93 kBps	3.64 kBps

Filesystem

	avg	current
kube-state-metrics-5f4c7f9d47-wrp5m	12.68 MiB	22.31 MiB
kube-addon-manager-minikube	3.50 MiB	3.45 MiB
kube-controller-manager-minikube	435.89 KiB	424.00 KiB
kubernetes-dashboard-6f4cfc5d87-5zsln	203.48 KiB	220.00 KiB
node-directory-size-metrics-lshqv	196.65 KiB	204.00 KiB

RED Method

- Inside service and container level metrics.



The Four Golden Rules

- The four golden signals of monitoring are **latency, traffic, errors**, and **saturation**. If you can only measure four metrics of your user-facing system, focus on these four.

Questions?

Thank you!

Skyworkz - https://skyworkz.nl

Lyle Henkeman - https://github.com/LyleHenkeman