Reveal Your Deepest Kubernetes Metrics

KubeCon EU 2018

FRESHTRACKS.io

@bob_cotton

About Me

FRESHTRACKS.io

- CTO & Co-Founder - FreshTracks.io - A CA Accelerator Incubation
- bob@freshtracks.io
- @bob_cotton
- Father, Fly Fisher & Avid homebrewer

Agenda

FRESHTRACKS.io

- Determining Important Metrics
  - Four Golden Signals
  - USE Method
  - RED Method
- Sources of metrics
  - Node
  - kubelet and containers
  - Kubernetes API
  - etcd
  - Derived metrics (kube-state-metrics)
- Metric Aggregation through the Kubernetes Hierarchy

What are the Important Metrics?

Four Golden Signals

FRESHTRACKS.io

- Latency
  - The time it takes to service a request.
- Errors
  - The rate of requests that fail, either explicitly, implicitly, or by policy
- Traffic
  - A measure of how much demand is being placed on your system
- Saturation
  - How "full" your service is.

USE Method

- Introduced by Brendan Gregg for reasoning about system resources
  - Resources are all physical server functional components (CPUs, disks, busses...)
- Utilization
  - 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
  - The count of error events

RED Method

FreshTracks.io

- Introduced by Tom Wilkie
  - A subset of the Four Golden Signals for measuring Services
- Rate
  - The number of requests per second
- Errors
  - The number of errors per second
- Duration
  - The length of time required to service the request

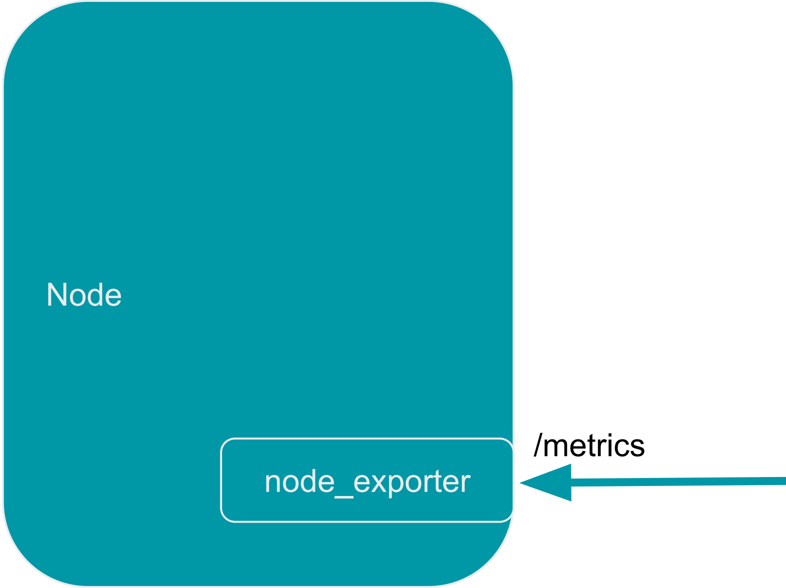
USE is for Resources
RED is for Services

Kubernetes Has Both!

Sources of Metrics in Kubernetes

Node Metrics from node_exporter

- node_exporter installed a DaemonSet
  - One instance per node
- Standard Host Metrics
  - Load Average
  - CPU
  - Memory
  - Disk
  - Network
  - Many others
- ~1000 Unique series in a typical node



@bob_cotton

USE for Node CPU

FreshTracks.io

Utilization	**node_cpu**	`sum(rate(node_cpu{mode!=”idle”,mode!=”iowait”, mode!~”^(?:guest.*)$”}[5m])) BY (instance)`
Saturation	**node_load1**	`sum(node_load1) by (node) / count(node_cpu{mode="system"})by (node) * 100`
Errors	N/A	Not exposed by node_exporter

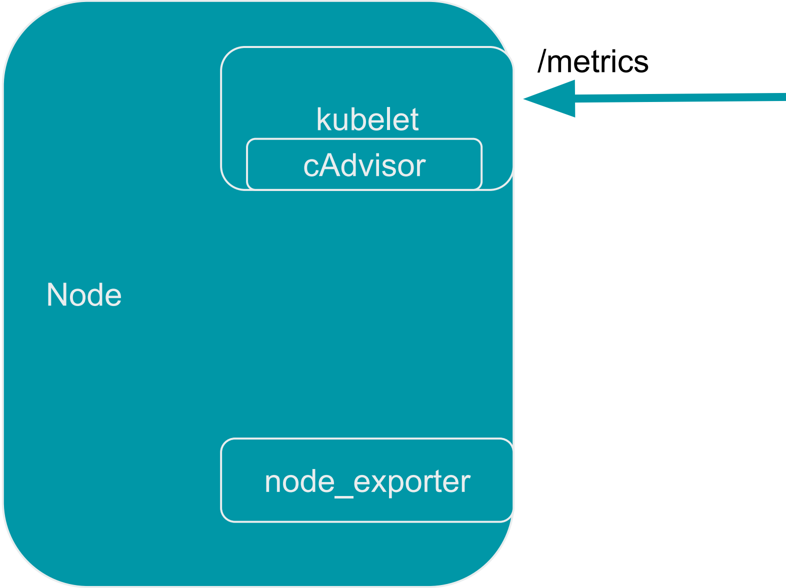
USE for Node Memory

Utilization	node_memory_MemAvailable node_memory_MemTotal kube_node_status_capacity_memory_bytes kube_node_status_allocatable_memory_bytes	1 - sum(node_memory_MemAvailable) by (node)/ sum(node_memory_MemTotal) by (node) 1- sum(kube_node_status_allocatable_memory_bytes) by (exported_node) / sum(kube_node_status_capacity_memory_bytes) by (exported_node)
Saturation	Don’t go into swap!	
Errors	node_edac_correctable_errors_total node_edac_uncorrectable_errors_total node_edac_csrow_correctable_errors_total node_edac_csrow_uncorrectable_errors_total	Only available on some systems

Container Metrics from cAdvisor

FreshTracks.io

- cAdvisor is embedded into the kubelet, so we scrape the kubelet to get container metrics
- These are the so-called Kubernetes “core” metrics
- For each container on the node:
  - CPU Usage (user and system) and time throttled
  - Filesystem read/writes/limits
  - Memory usage and limits
  - Network transmit/receive/dropped



@bob_cotton

USE for Container CPU

Utilization	**container_cpu_usage_seconds_total**	sum(rate(**container_cpu_usage_seconds_total**[5m])) by (container_name)
Saturation	**container_cpu_cfs_throttled_seconds_total**	sum(rate(**container_cpu_cfs_throttled_seconds_total**[5m]) by (container_name)
Errors	N/A	

USE for Container Memory

Utilization	**container_memory_usage_bytes** **container_memory_working_set_bytes**	sum(**container_memory_working_set_bytes**{name!~"POD"}) by (name)

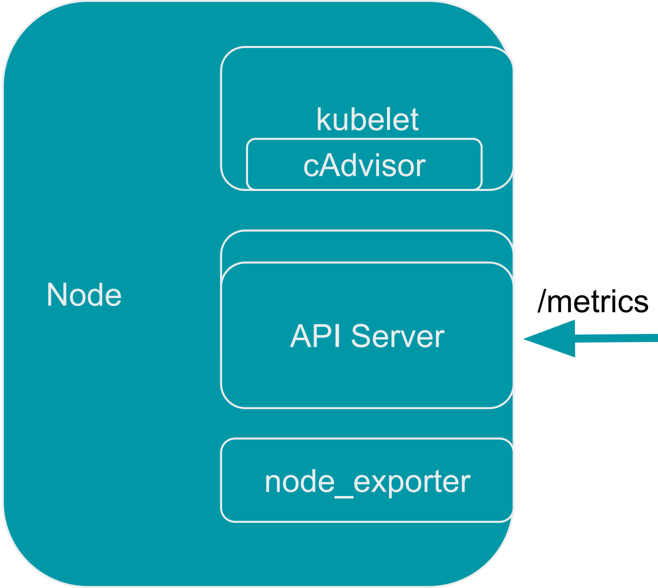
Saturation	Ratio of:
	container_memory_working_set_bytes / kube_pod_container_resource_limits_memory_bytes
		sum(**container_memory_working_set_bytes**)
		by (container_name) /
		sum(label_join(**kube_pod_container_resource_limits_memory_bytes**, "container_name", "", "container"))
		by (container_name)

Errors	**container_memory_failcnt** -- Number of memory usage hits limits. **container_memory_failures_total** -- Cumulative count of memory allocation failures.	sum(rate(**container_memory_failures_total** {type="pgmajfault"}[5m])) by (container_name)

Kubernetes Metrics from the K8s API Server

FRESHTRACKS.io

- Metrics about the performance of the K8s API Server
  - Performance of controller work queues
  - Request Rates and Latencies
  - Etcd helper cache work queues and cache performance
  - General process status (File Descriptors/Memory/CPU Seconds)
  - Golang status (GC/Memory/Threads)



@bob_cotton

RED for Kubernetes API Server

Rate	**apiserver_request_count**	`sum(rate(apiserver_request_count[5m])) by (verb)`
Errors	**apiserver_request_count**	`rate(apiserver_request_count{code=~"^(?:5..)$"}[5m]) / rate(apiserver_request_count[5m])`
Duration	**apiserver_request_latencies_bucket**	`histogram_quantile(0.9, rate(apiserver_request_latencies_bucket[5m])) / 1e+06`

K8s Derived Metrics from kube-state-metrics

- Counts and metadata about many K8s types
  - Counts of many “nouns”
  - Resource Limits
  - Container states
    - ready/**restarts**/running/terminated/waiting
- *_labels series carries labels
  - Series has a constant value of **1**
  - Join to other series for on-the-fly labeling using `left_join`

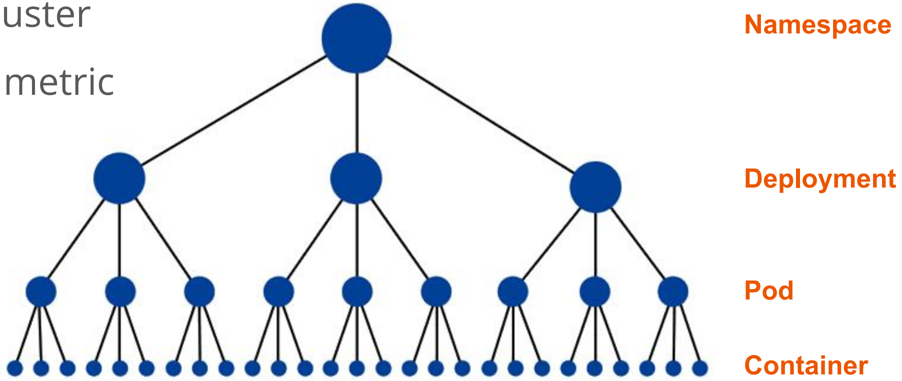
Etcd Metrics from etcd

- Etcd is “master of all truth” within a K8s cluster
  - Leader existence and leader change rate
  - Proposals committed/applied/pending/failed
  - Disk write performance
  - Inbound gRPC stats
    - `etcd_http_received_total`
    - `etcd_http_failed_total`
    - `etcd_http_successful_duration_seconds_bucket`
  - Intra-cluster gRPC stats
    - `etcd_network_member_round_trip_time_seconds_bucket`
    - `...`

Core Metrics Aggregation

FRESHTRACKS.io

- K8s clusters form a hierarchy
- We can aggregate the “core” metrics to any level
- This allows for some interesting monitoring opportunities
- Using Prometheus “recording rules” aggregate the core metrics at every level
- Insights into all levels of your Kubernetes cluster
- This also applies to any custom application metric



@bob_cotton

Thanks

@bob_cotton

Resources

FRESHTRACKS.io

- USE Method
- RED Method
- Deep Dive into Kubernetes Metrics
- kube-state-metrics

Scheduling and Autoscaling i.e. The Metrics Pipeline

@bob_cotton

The New “Metrics Server”

FRESHTRACKS.io

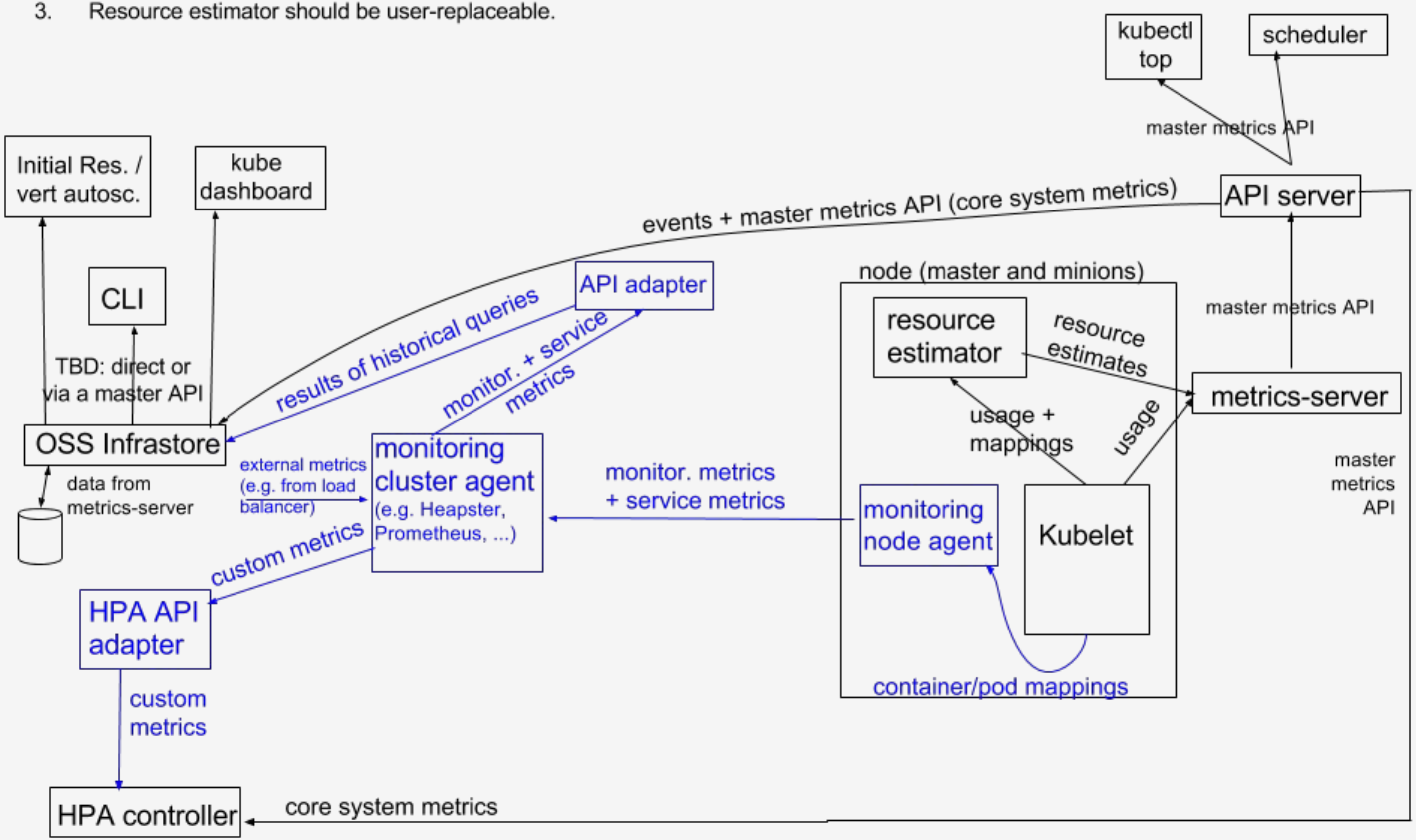
- Replaces Heapster
- Standard (versioned and auth) API aggregated into the K8s API Server
- In “beta” in K8s 1.8
- Used by the scheduler and (eventually) the Horizontal Pod Autoscaler
- A stripped-down version of Heapster
- Reports on “core” metrics (CPU/Memory/Network) gathered from cAdvisor
- For internal to K8s use only.
- Pluggable for custom metrics

Monitoring architecture proposal: OSS

(arrows show direction of metrics flow)

Notes

1. Arrows show direction of metrics flow.
2. Monitoring pipeline is in blue. It is user-supplied and optional.
3. Resource estimator should be user-replaceable.



@bob_cotton

Feeding the Horizontal Pod Autoscaler

FRESHTRACKS.io

- Before the metrics server the HPA utilized Heapster for it’s Core metrics
  - This will be the metrics-server going forward
- API Adapter will bridge to third party monitoring system
  - e.g. Prometheus

Labels, Re-Label and Recording Rules Oh My...

@bob_cotton

Label/Value Based Data Model

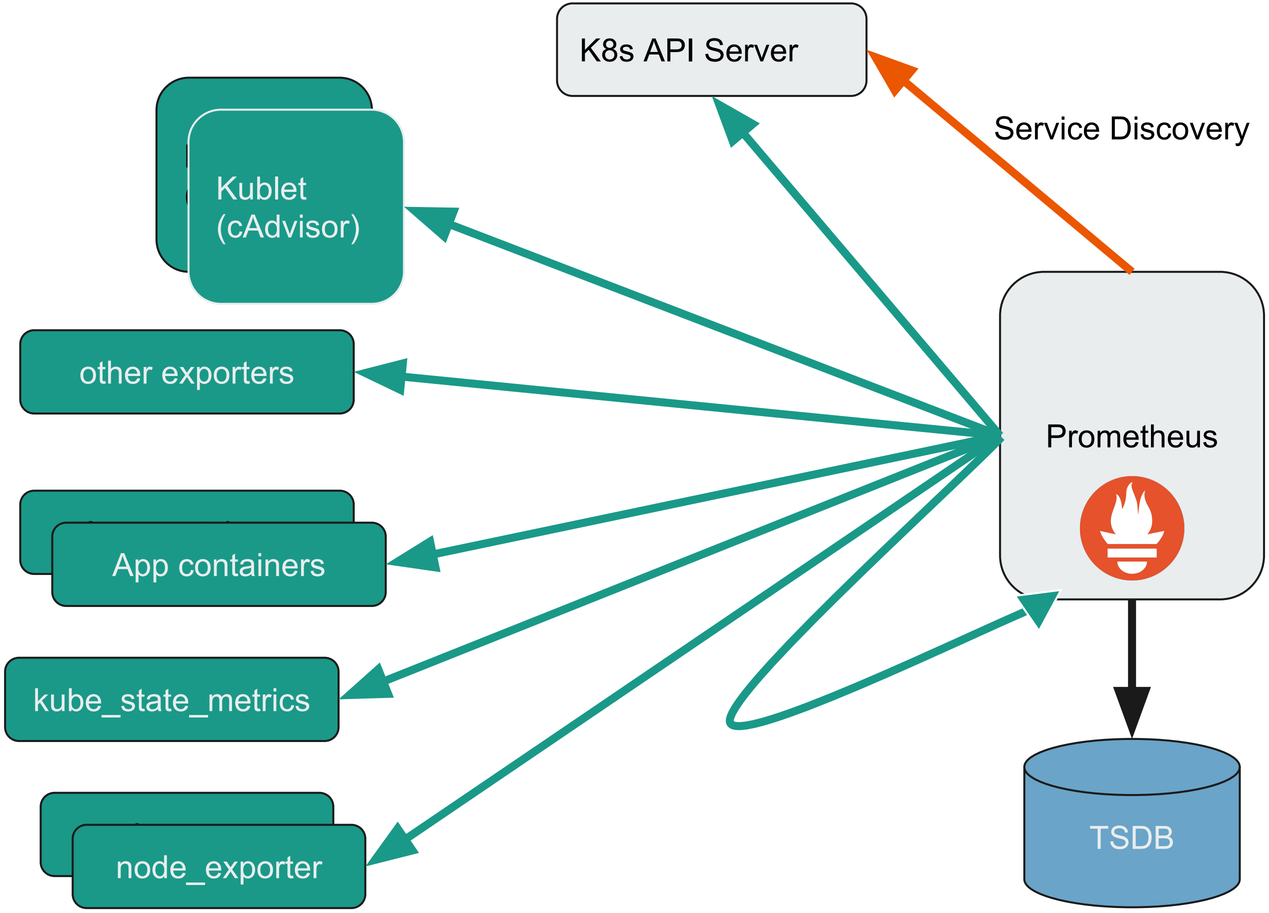
- Graphite/StatsD
  - apache.192-168-5-1.home.200.http_request_total
  - apache.192-168-5-1.home.500.http_request_total
  - apache.192-168-5-1.about.200.http_request_total
- Prometheus
  - http_request_total{job=”apache”, instance=”192.168.5.1”, path=”/home”, status=”200”}
  - http_request_total{job=”apache”, instance=”192.168.5.1”, path=”/home”, status=”500”}
  - http_request_total{job=”apache”, instance=”192.168.5.1”, path=”/about”, status=”200”}
- Selecting Series
  - *.*.home.200.*.http_requests_total
  - http_requests_total{status=”200”, path=”/home”}

Kubernetes Labels

FreshTracks.io

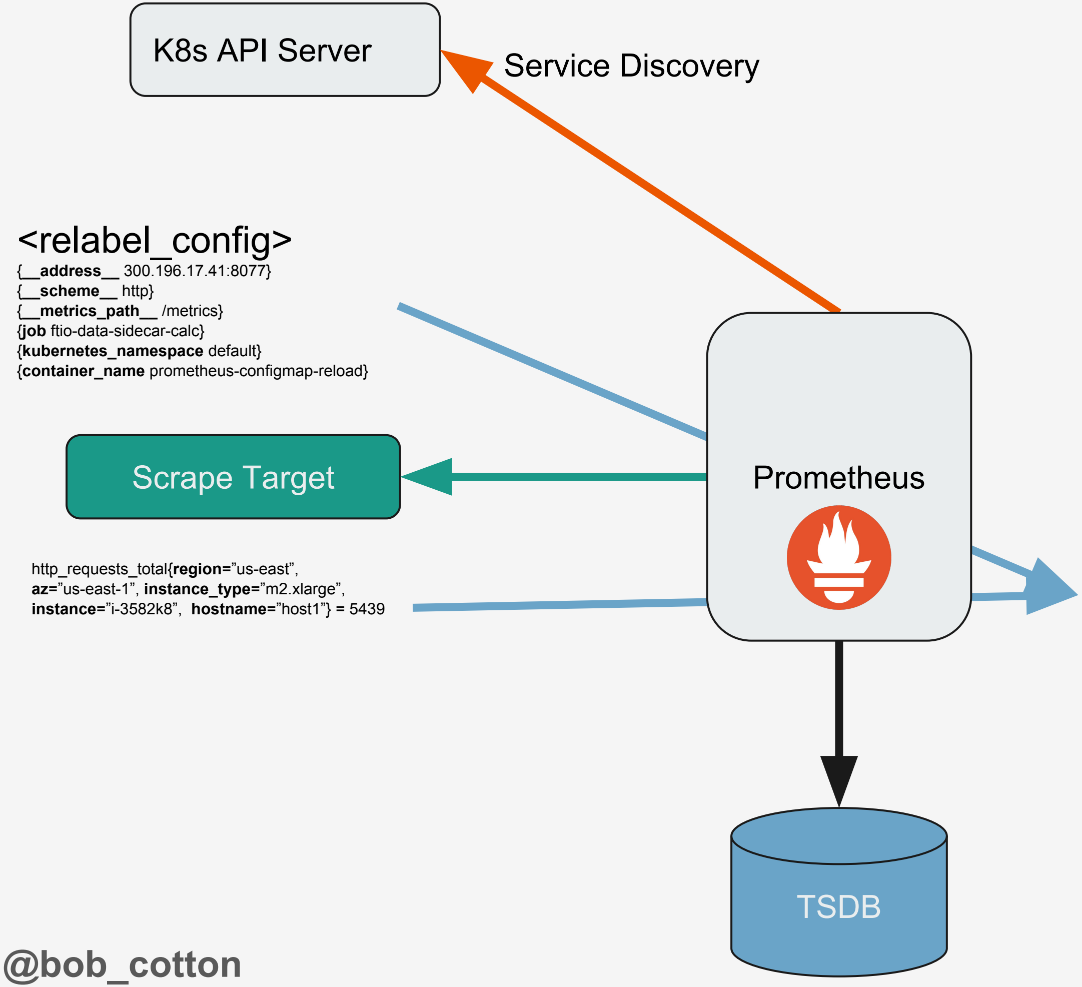
- Kubernetes gives us labels on all the things
- Our scrape targets live in the context of the K8s labels
  - This comes from service discovery
- We want to enhance the scraped metric labels with K8s labels

- This is why we need relabel rules in Prometheus



FRESHTRACKS.io

@bob_cotton



```
0="{__address__ 300.196.17.41}"
1="{__meta_kubernetes_namespace default}"
2="{__meta_kubernetes_pod_annotation_freshtracks_io_data_sidecar true}"
3="{__meta_kubernetes_pod_annotation_freshtracks_io_path /metrics2}"
4="{__meta_kubernetes_pod_annotation_kubernetes_io_created_by "kind":"SerializedReference"?}"
5="{__meta_kubernetes_pod_annotation_kubernetes_io_limit_ranger LimitRanger plugin set: cpu
request for container prometheus-configmap-reload; cpu request for container data-sidecar}"
6="{__meta_kubernetes_pod_annotation_prometheus_io_port 8077}"
7="{__meta_kubernetes_pod_annotation_prometheus_io_scrape false}"
8="{__meta_kubernetes_pod_container_name prometheus-configmap-reload}"
9="{__meta_kubernetes_pod_host_ip 172.20.42.119}"
10="{__meta_kubernetes_pod_ip 100.96.17.41}"
11="{__meta_kubernetes_pod_label_freshtracks_io_cluster bowl.freshtracks.io}"
12="{__meta_kubernetes_pod_label_pod_template_hash 1636686694}"
13="{__meta_kubernetes_pod_label_run data-sidecar}"
14="{__meta_kubernetes_pod_name data-sidecar-1636686694-83crm}"
15="{__meta_kubernetes_pod_node_name ip-xx-xxx-xx-xxx.us-west-2.compute.internal}"
16="{__meta_kubernetes_pod_ready false}"
17="{__metrics_path__ /metrics}"
18="{__scheme__ http}"
19="{job ftio-data-sidecar-calc}"
```

<relabel_config>

```
{__address__ 300.196.17.41:8077}
{__scheme__ http}
{__metrics_path__ /metrics}
{job ftio-data-sidecar-calc}
{kubernetes_namespace default}
{container_name prometheus-configmap-reload}
```

```
http_requests_total{region=”us-east”,
az=”us-east-1”, instance_type=”m2.xlarge”,
instance=”i-3582k8”,  hostname=”host1”} = 5439
```

```
http_requests_total{region=”us-east”,
az=”us-east-1”,
instance_type=”m2.xlarge”,
instance=”i-3582k8”,
hostname=”host1”,
instance=”300.196.17.41:8077”,
job=”ftio-data-sidecar-calc”,
kubernetes_namespace=”default”,
container_name=”prometheus-configmap-reload”,
} = 5439
```

<metric_relabel_config>

@bob_cotton

Recording Rules

Create a new series, derived from one or more existing series

```
# The name of the time series to output to. Must be a valid metric name.
record: <string>

# The PromQL expression to evaluate. Every evaluation cycle this is
# evaluated at the current time, and the result recorded as a new set of
# time series with the metric name as given by 'record'.
expr: <string>

# Labels to add or overwrite before storing the result.
labels:
  [ <labelname>: <labelvalue> ]
```

Recording Rules

FRESHTRACKS.io

Create a new series, derived from one or more existing series

```
record: pod_name:cpu_usage_seconds:rate5m
expr: sum(rate(container_cpu_usage_seconds_total{pod_name=~"^(?:.+)$"}[5m]))
  BY (pod_name)
labels:
  ft_target: "true"
```