Treating Dashboards Like Code

Scott Kidder, Staff Software Engineer @ Mux

Grafanacon, February 26, 2019

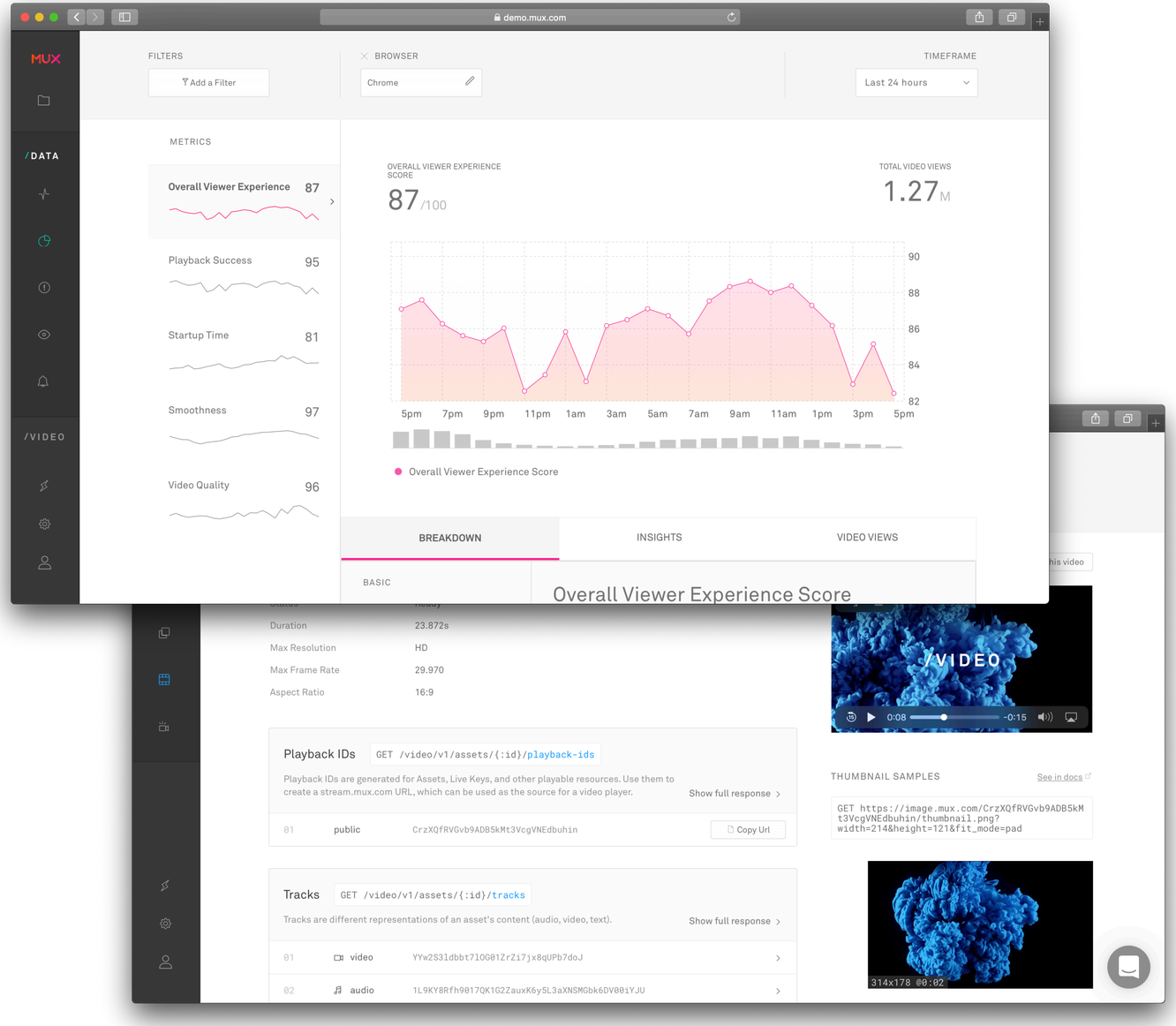
Agenda

- Background on Mux
- Monitoring for Mux Data
- Greenfield Monitoring Opportunities with Mux Video
- Goals for monitoring
- Questions

Background on Mux

What is Mux?

- Mux Data: Analytics for Video (2016)
- Mux Video: API for Video (2018)
- Mux Video makes it easy to publish video with a REST API call
- Optimal video encoding settings chosen automatically
- Deployments in AWS and Google Cloud



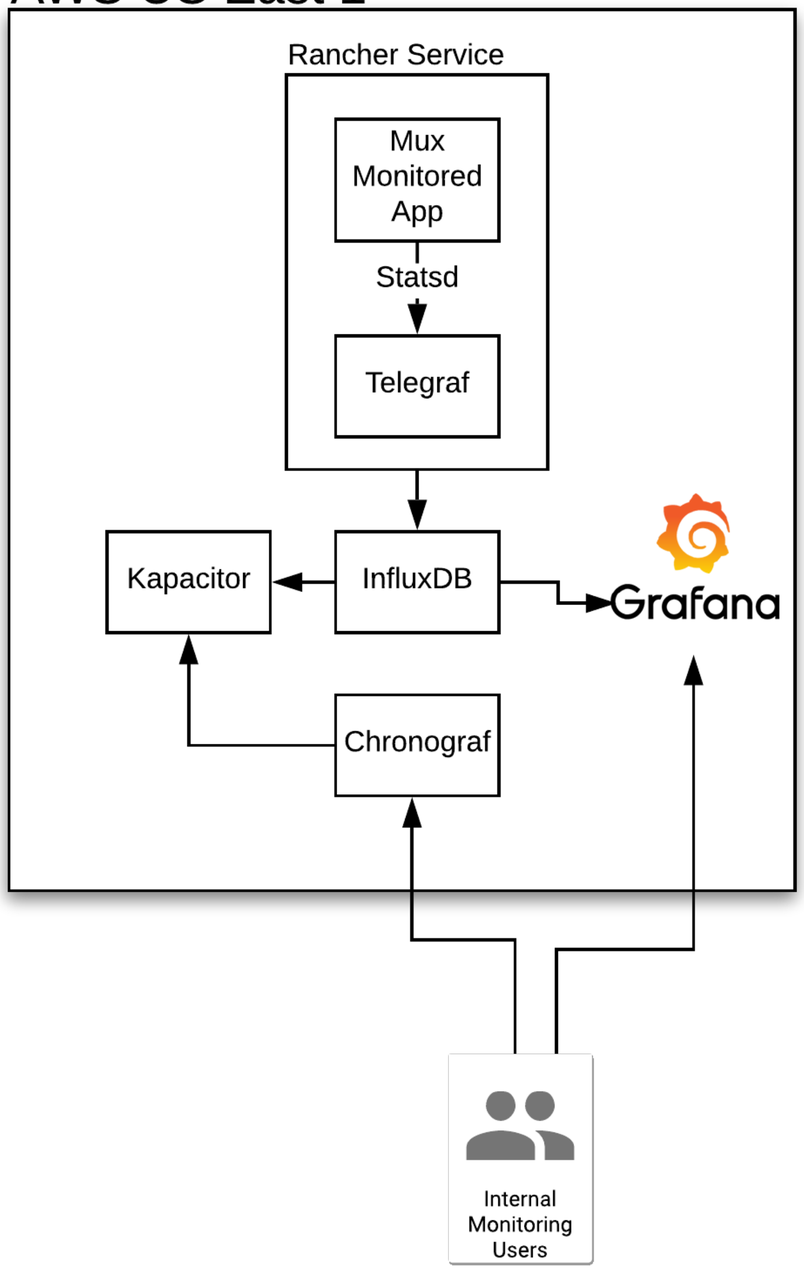
Monitoring for Mux Data

For a Moment, Let's Return to a Simpler Time

- Mux has used Grafana since inception (early 2016)
- Single deployment of Rancher container orchestration system in AWS
- Supported Mux Data, our only product at the time
- Single Grafana instance for all dashboards
- Single InfluxDB instance for application metrics

MUX

AWS US East 1


Rancher Service
Mux Monitored App
Statsd
Telegraf
Kapacitor
InfluxDB
Grafana
Chronograf
Internal Monitoring Users

But in many ways, things were more difficult...

MUX

Problems began to surface

- Management of alerting rules was performed in Chronograf
- Ran a second visualization tool just to administer alerting rules
- No versioned history of alerting rules
- Rules were often disabled during a deploy or maintenance, and then people would forget to re-enable them, leading to undetected incidents
- Unclear why alerts were disabled, and whether it’s safe to delete

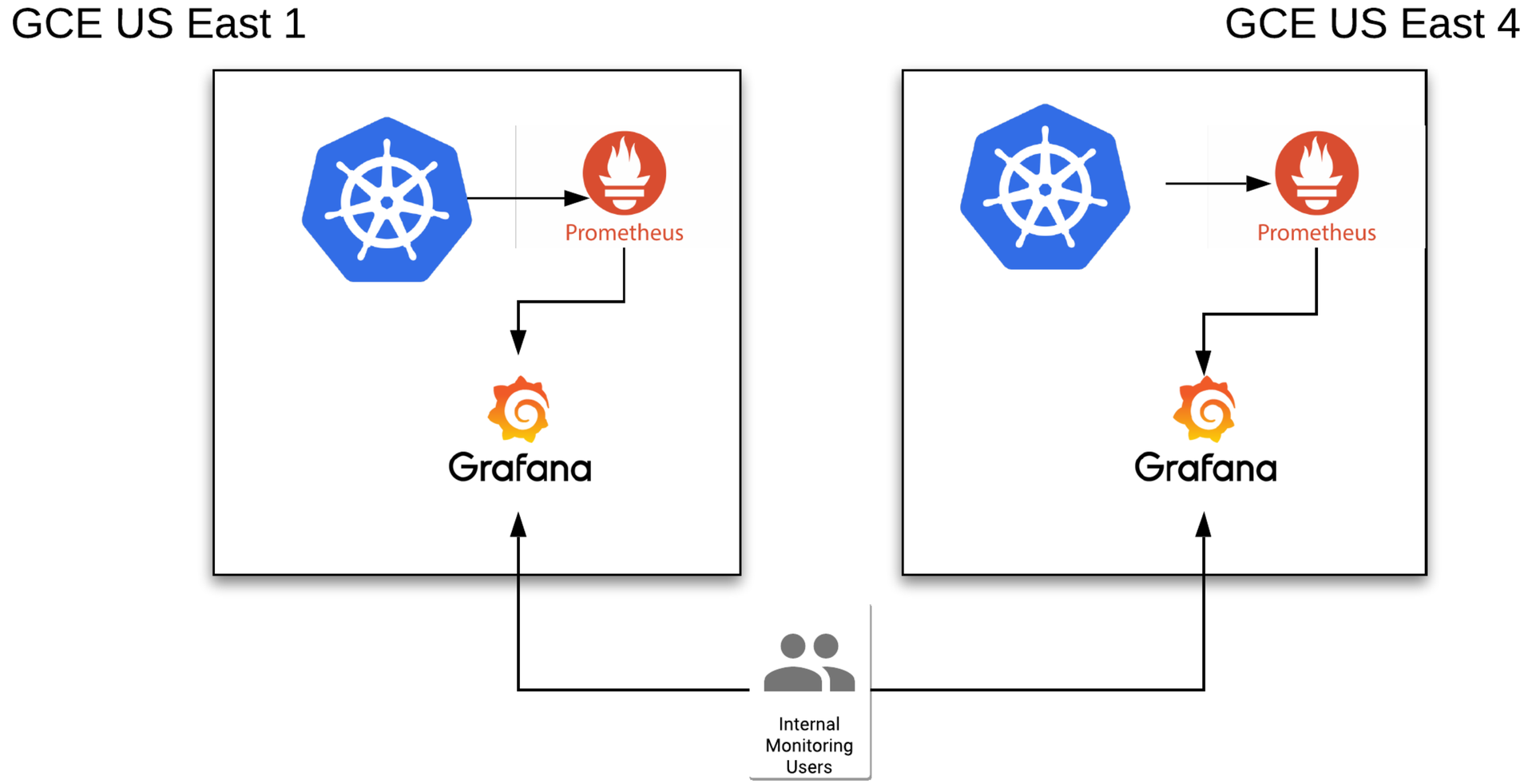
MUX

Greenfield Monitoring Opportunities with Mux Video

Mux Video Development

- Late 2017 we began developing Mux Video
- We had already run some proof-of-concept Kubernetes cluster with Mux Data
- Decided to run all services in Kubernetes and monitor with Prometheus and Grafana

MUX


GCE US East 1
GCE US East 4
Prometheus
Grafana
Prometheus
Grafana
Internal
Monitoring
Users

Goals for Monitoring

MUX

Goals

1. Easily configure which services are scraped by Prometheus
2. Run policy checks on alert rules with each build
3. Store the dashboards and alert rules alongside code
4. Automatically deploy dashboards and alert rules to Kubernetes clusters each time we ship code

MUX

Goal #1

Easily configure which services are scraped by Prometheus

MUX

Prometheus Monitoring in Kubernetes

- Using the Prometheus Operator to configure Prometheus and Alertmanager
- https://github.com/coreos/prometheus-operator
- Uses Kubernetes label metadata to target which services to scrape and on which port

MUX

Prometheus: Kubernetes Service Monitor

1) Examine services in all Kubernetes namespaces

2) Match on services with a “monitoring: core” label

3) Scrape whatever port is named “metrics”

```
apiVersion:
monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: core-servers
  namespace: monitoring
  labels:
    k8s-app: core-servers
spec:
  jobLabel: core-servers
  namespaceSelector:
    any: true
  selector:
    matchLabels:
      monitoring: core
  endpoints:
  - port: metrics
    interval: 10s
    honorLabels: true
```


MUX

Prometheus: Monitored Service

```
apiVersion: v1
kind: Service
metadata:
   name: kafka
   namespace: default
   labels:
      app: kafka
      monitoring: core
```

1) Simply add the “monitoring: core” label to a server

MUX

Services Scraped

Prometheus Alerts Graph Status Help

Targets

accesslogs-kafka (3/3 up)

Endpoint	State	Labels	Last Scrape	Error
http://100.96.55.84:5556/metrics	UP	endpoint="metrics" instance="100.96.55.84:5556" namespace="gce-us-east1-production" pod="accesslogs-kafka-broker-0" service="accesslogs-kafka"	3.3s ago	
http://100.96.58.20:5556/metrics	UP	endpoint="metrics" instance="100.96.58.20:5556" namespace="gce-us-east1-production" pod="accesslogs-kafka-broker-1" service="accesslogs-kafka"	4.169s ago	
http://100.96.59.22:5556/metrics	UP	endpoint="metrics" instance="100.96.59.22:5556" namespace="gce-us-east1-production" pod="accesslogs-kafka-broker-2" service="accesslogs-kafka"	9.769s ago	

apiserver (3/3 up)

Endpoint	State	Labels	Last Scrape	Error
https://10.142.0.10:443/metrics	UP	endpoint="https" instance="10.142.0.10:443" namespace="default" service="kubernetes"	8.029s ago	
https://10.142.0.3:443/metrics	UP	endpoint="https" instance="10.142.0.3:443" namespace="default" service="kubernetes"	10.574s ago	
https://10.142.0.9:443/metrics	UP	endpoint="https" instance="10.142.0.9:443" namespace="default" service="kubernetes"	2.951s ago	

MUX

Goal #2

Run policy checks on alert rules with each build

MUX

Prometheus: Automated Policy Check

✔ 🙏 Policy Check docker-compose -f .buildki… Ran in 43s Waited 6s buildkite-dind-m76bq

Log | Artifacts | Timeline | Environment

+ Expand groups − Collapse groups Delete Download Follow

```
1  ▸ Preparing build folder                                    1s
17 ▾ Running build script                                      42s
18 $ docker-compose -f .buildkite/docker-compose.yaml run --rm policycheck ./policy-
   check.sh
19 Checking ./core/golang/kafka/monitoring/kafka-consumer.rules.yaml
20   SUCCESS: 1 rules found
21
22 Checking ./core/servers/consul/monitoring/consul.rules.yaml
23   SUCCESS: 4 rules found
24
25 Checking ./core/servers/hadoop/monitoring/hadoop.rules.yaml
26   SUCCESS: 3 rules found
27
28 Checking ./core/servers/zookeeper/monitoring/zookeeper.rules.yaml
29   SUCCESS: 3 rules found
30
31 Checking ./data/internal/servers/chproxy/monitoring/chproxy.rules.yaml
32   SUCCESS: 7 rules found
```


MUX

Prometheus: Automated Policy Check

1) Use promtool to validate alert rules files

2) Verify that all files end with a new-line to allow for concatenation

```
MONITORING_YAML_FILES=$(find $searchdir -type f -name
"*.rules.yaml" | sort)
for f in $MONITORING_YAML_FILES; do
  promtool check rules $f
  rc=$?
  if [[ $rc != 0 ]]; then
    echo "$f is not a valid Prometheus alert rule
YAML file."
    echo ""
    ERRORS="yes"
  fi
  LAST_CHAR=$(cat $f | tr '\n' '#' | tail -c 1)
  if [[ $LAST_CHAR != "#" ]]; then
    echo "$f does not end with a new line."
    echo ""
    ERRORS="yes"
  fi
done
```


MUX

Goal #3

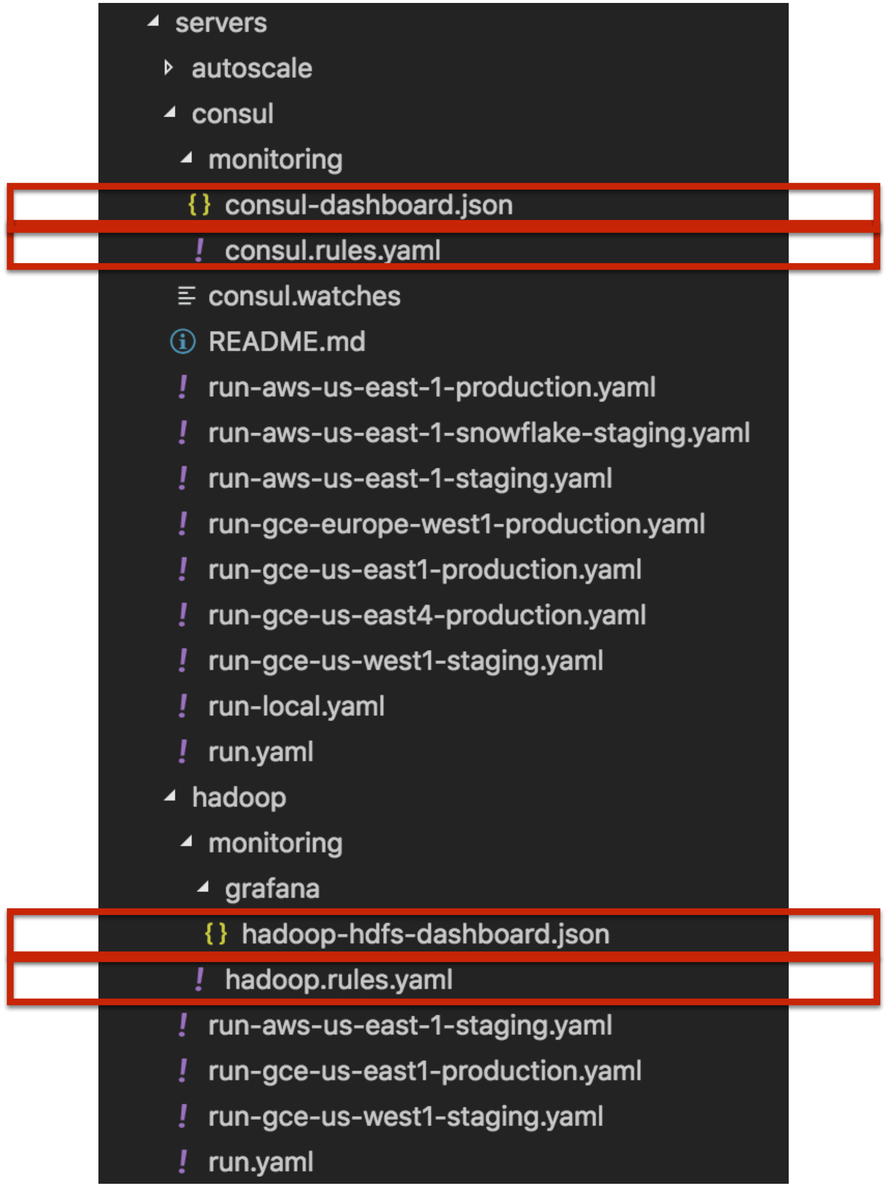
Store the dashboards and alert rules alongside code

MUX

Code Organization

1) Dashboards are named “*-dashboard.json”, and stored in a “monitoring/grafana” directory for the associated component

2) Alert rules are named “*.rules.yaml” and kept in a “monitoring” directory



Goal #4

Automatically deploy dashboards and alert rules to Kubernetes clusters each time we ship code

MUX

Automatic Deployment of Dashboards and Alert Rules

- Our Buildkite builds automatically generate Kubernetes manifest for servers across all target environments
- Also generate Kubernetes ConfigMaps with Grafana dashboards and Prometheus alert rules
- Buildkite deploy plan applies Kubernetes manifests and ConfigMaps to each Kubernetes cluster
- Grafana and Prometheus ConfigMaps automatically reloaded

MUX

Gather Alerting Rules

1) Find all alert rules files conforming to naming pattern

```
#!/bin/bash

set -e
RULES_DIR=$1
mkdir -p $OUTPUT_DIR
rm -r $OUTPUT_DIR/* || true

for searchdir in "${@:2}"; do
      RULES_FILES=$(find $searchdir
-type f -name "*.rules.yaml")
      for file in $RULES_FILES; do
            cp $file $RULES_DIR
      done
done
```


MUX

Generate Kubernetes ConfigMap with Alert Rules

1) Begin rendering a Kubernetes ConfigMap manifest

2) Concatenate contents of each alert rule file to the ConfigMap

```
set -e
NAMESPACE=$1
OUTPUT_FILE=$2
RULES_DIR=$3

mkdir -p $(dirname $OUTPUT_FILE)

cat <<-EOF > $OUTPUT_FILE
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-k8s-rules
  namespace: $NAMESPACE
  labels:
    role: prometheus-rulefiles
    prometheus: k8s
data:
EOF

for f in $(find $RULES_DIR -type f -name
"*.rules.yaml")
do
  echo "  $(basename $f): |+" >> $OUTPUT_FILE
  cat $f | sed "s/^/    /g" >> $OUTPUT_FILE
done
```


MUX

Automatic Deployment of Prometheus Alert Rules

- Prometheus Operator includes a config reloader that monitors the ConfigMap for changes
- Sends web hook to Prometheus instructing it to reload its config

MUX

Gather Grafana Dashboards and Datasources

1) Find all Grafana dashboard and datasource files conforming to naming pattern

```
#!/bin/bash
set -e



OUTPUT_DIR=$1
mkdir -p $OUTPUT_DIR
rm -r $OUTPUT_DIR/* || true

for searchdir in "${@:2}"; do
      DASHBOARD_FILES=$(find $searchdir -type
f -name "*-dashboard.json" -o -name "*-
datasource.json" | sort)
      for file in $DASHBOARD_FILES; do
        echo "FILE: $file"
            cp $file $OUTPUT_DIR
        done
done
```


MUX

Render ConfigMap with Grafana Dashboards

Have been using the `grafana-dashboards-configmap-generator` script at https://github.com/eedugon/grafana-dashboards-configmap-generator

```
monitoring/grafana/grafana-dashboards-configmap-generator/bin/grafana_dashboards_generate.sh \
  -n ${MONITORING_NAMESPACE} \
  -s 200000 \
  -o run/k8s ${NAMESPACE}/monitoring/grafana/config-map.yaml \
  -g run/k8s/${NAMESPACE}/monitoring/grafana/run.yaml \
  -i run/monitoring/grafana \
  --hostname ${GRAFANA_HOSTNAME}
```


MUX

Grafana Watcher to reload Dashboards

```
- name: grafana-watcher
  image: quay.io/coreos/grafana-watcher:v0.0.8
  args:
    - '--watch-dir=/var/grafana-dashboards-0'
    - '--grafana-url=http://localhost:3000'
  env:
  - name: GRAFANA_USER
    valueFrom:
      secretKeyRef:
        name: grafana-credentials
        key: user
  - name: GRAFANA_PASSWORD
    valueFrom:
      secretKeyRef:
        name: grafana-credentials
        key: password
  volumeMounts:
  - name: grafana-dashboards-0
    mountPath: /var/grafana-dashboards-0
volumes:
- name: grafana-storage
  emptyDir: {}
- name: grafana-dashboards-0
  configMap:
    name: grafana-dashboards-0
```

1) Use ‘grafana-watcher’ container to reload Grafana dashboards supplied in ConfigMap volume

Next Steps

- Replace ‘grafana-watcher’ pod with Grafana provider config that automatically reloads dashboards from ConfigMap volume path
- Control over which dashboards are deployed; some Grafana instances have dashboards that are unused or point to non-existent servers

MUX

Credit to the Mux Team

Adam Brown

Matt Ward

Thank You!

MUX