Supercronic

Le projet

https://github.com/aptible/supercronic

Extrait:

> *Supercronic's goal is to behave exactly how you would expect* `cron` *running in a container to behave:*
>
> - *Your environment variables are available in jobs*
> - *Job output is logged to* `stdout` / `stderr`
> - `SIGTERM` *triggers a graceful shutdown (and so does* `SIGINT`*, which you can deliver via CTRL+C when used interactively)*
> - *Job return codes and schedules are logged to* `stdout` / `stderr`
> - `SIGUSR2` *triggers a graceful shutdown and reloads the crontab configuration*
> - `SIGQUIT` *triggers a graceful shutdown*

Dockerfile

Dockerfile:

```
FROM debian:stable-slim

RUN apt-get update \
  && apt-get -y dist-upgrade \
  && apt-get -y install curl \
  && apt-get clean \
  && apt-get autoclean

ENV PORT="12345"
ENV LISTENURL="0.0.0.0"

# Latest releases available at https://github.com/aptible/supercronic/releases
ARG SUPERCRONIC_URL=https://github.com/aptible/supercronic/releases/download/v0.2.33/supercronic-linux-amd64
ARG SUPERCRONIC_SHA1SUM=71b0d58cc53f6bd72cf2f293e09e294b79c666d8
ARG SUPERCRONIC=supercronic-linux-amd64

RUN curl -fsSLO "$SUPERCRONIC_URL" \
  && echo "${SUPERCRONIC_SHA1SUM}  ${SUPERCRONIC}" | sha1sum -c - \
  && chmod +x "$SUPERCRONIC" \
  && mv "$SUPERCRONIC" "/usr/local/bin/${SUPERCRONIC}" \
  && ln -s "/usr/local/bin/${SUPERCRONIC}" /usr/local/bin/supercronic

RUN groupadd -r supercronic && useradd -m -r -s "/bin/sh" -g supercronic supercronic

USER supercronic

CMD  ["sh", "-c", "/usr/local/bin/supercronic -json -prometheus-listen-address ${LISTENURL}:${PORT} /opt/crontab"]
```

Commande de build & push:

```
docker build -t docker-registry.ocb-corp.caascad.com/dev/supercronic-yme:v0.2.33 .
docker push docker-registry.ocb-corp.caascad.com/dev/supercronic-yme:v0.2.33
```

Deploiement

Namespace:

```
kubectl create ns supercronic-yme
```

Secret pour la registry:

```
export VAULT_ADDR=https://vault.infra-stg.caascad.com
vault read -field="value-json" secret/concourse-infra/global/docker-regi
kubectl create -f secret.json --namespace supercronic-yme
```

Deployment & Configmap

```
kubectl apply -f deployment.yml
```

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: supercronic-yme
  namespace: supercronic-yme
spec:
  replicas: 1
  selector:
    matchLabels:
      app: supercronic-yme
  template:
    metadata:
      labels:
        app: supercronic-yme
    spec:
      imagePullSecrets:
        - name: docker-registry-external-fe
      containers:
        - image: docker-registry.caascad.com/dev/supercronic-yme:v0.2.33
          name: supercronic-yme
          command:
            [ "sh", "-c",
              "/usr/local/bin/supercronic -json -prometheus-listen-address 0.0.0.0:12345 /etc/configmap/crontab", ]
          ports:
            - name: metrics
              containerPort: 12345
          volumeMounts:
            - name: crontab
              mountPath: /etc/configmap
      volumes:
        - name: crontab
          configMap:
            name: supercronic-yme-crontab
```

Configmap

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: supercronic-yme-crontab
  namespace: supercronic-yme
data:
  crontab: |
    * * * * * date
    * * * * * false
    * * * * * unecommandequinexistepas
```

Ce qu'il faut voir

- Logs :
  - c'est en JSON
  - les logs sont explicites
- Port-forward sur le port 12345 : on voit les métriques
  - `supercronic_currently_running` : permet de savoir quelles commandes sont en cours d'exécution. Permet aussi de connaître leur durée dans le temps
  - `supercronic_executions` : nombre de fois que la commande a été exécutée. Permet d'identifier des plages où elle n'est pas exécutée (quand `increase()` n'augmente pas)
  - `supercronic_failed_executions` : permet d'identifier les commandes ayant échoué (idem : c'est un compteur)
  - `supercronic_successful_executions` : permet de s'assurer qu'une commande a été exécutée avec succès (idem : c'est un compteur)
  - `supercronic_cron_execution_time_seconds_xxx` c'est un histogramme sur le temps d'exécution des commandes

Métriques

Extrait :

```
# HELP supercronic_currently_running count of currently running cron executions
# TYPE supercronic_currently_running gauge
supercronic_currently_running{command="date",position="0",schedule="* * * * *"} 0
supercronic_currently_running{command="false",position="1",schedule="* * * * *"} 0
supercronic_currently_running{command="unecommandequinexistepas",position="2",schedule="* * * * *"} 0
# HELP supercronic_executions count of cron executions
# TYPE supercronic_executions counter
supercronic_executions{command="date",position="0",schedule="* * * * *"} 1
supercronic_executions{command="false",position="1",schedule="* * * * *"} 1
supercronic_executions{command="unecommandequinexistepas",position="2",schedule="* * * * *"} 1
# HELP supercronic_failed_executions count of failed cron executions
# TYPE supercronic_failed_executions counter
supercronic_failed_executions{command="false",position="1",schedule="* * * * *"} 1
supercronic_failed_executions{command="unecommandequinexistepas",position="2",schedule="* * * * *"} 1
# HELP supercronic_successful_executions count of successul cron executions
# TYPE supercronic_successful_executions counter
supercronic_successful_executions{command="date",position="0",schedule="* * * * *"} 1
```

Noter la cardinalité faible par rapport aux Cronjobs de Kubernetes

Cronjobs

Les cronJobs de Kubernetes génèrent entre autres la création d'un pod à chaque exécution.

Pour chaque pod créé, des métriques de kubelet et kube-state-metrics sont mises à disposition avec des labels différents. Cela augmente sérieusement la cardinalité des métriques liées au pod.

La création d'un pod est une opération lourde (télécharger l'image, l'instancier, créer le container, puis faire le ménage a posteriori)

Les pods sont différents à chaque exécution

- les logs doivent être consultées avec un outil comme Loki/Grafana
- les métriques sur l'exécution de chaque pod sont consultables avec des expressions promQL complexes.

Cependant, pendant la non-exécution des jobs, les ressources Kubernetes ne sont pas utilisées.

Utilisation de Supercronic

Supercronic s’utilise plutôt

- en l’injectant dans un pod existant (avec un init-job et un volume éphémère comme `emptyDir` par exemple)
- en surchargeant la `command` du container dont on doit lancer la commande en boucle

Cela évite de devoir générer des images spécifiques.

Résumé

- Alternative aux `cronJobs`, mais avec une faible cardinalité des métriques et des logs faciles d’exploitation
- Facile à mettre en place
- Présente des métriques d’exécution des jobs
- Kubernetes-friendly