Job Scheduling Using Kubernetes Scheduler

DESCRIPTION

Use Kubernetes Scheduler to automate the alert allocation for an analyst on a regular interval of time.

**Background of the problem statement:**  
A cybersecurity company whose product development is working on Real-time Threat Detection and Incident Reporting has several analysts who are responsible for doing the analysis on a daily basis and report the incidents. The incidents that get generated are numerous. Being a  
Kubernetes expert how would you automate the process of allocating these alerts to an analyst periodically?

**Features to be included:**

* Auto Scheduling
* Dynamic analysis of Jobs
* Scheduling/Rescheduling

**You must use the following:**

* GKE Service Account with Valid Project: to authenticate to Google Cloud Platform services
* Google Cloud Shell: a shell environment for managing resources hosted on Google Cloud Platform (GCP)
* kubectl: Kubernetes command-line tool, that allows you to run commands against Kubernetes clusters

**Following requirements should be met:**

* System should able to schedule the alerts/jobs every second/minute based upon the number of people available in a shift.
* Dynamic analysis of no. of people/analyst working in a shift.
* Scheduling/Rescheduling of tasks automatically.
* Document the step-by-step process involved in completing this task.

This section will guide you to:

* Understand how Kubernetes helps you design a system that should able to schedule pods on a node.
* Create dynamic resource scheduler for the created clusters.
* Implement scheduling/rescheduling.

**Step 1: Create a Kubernetes Cluster**

**Step 2: Creation of job/tasks (submit CronJobs)**

Here we are going to use CronJobs to run tasks at a specific time or interval and to execute or assign the tasks automatically at regular intervals of time. CronJobs are a good choice for automatic tasks such as backups, reporting, sending emails, or cleanup tasks (job scheduling).

CronJobs use job objects to complete their tasks. A CronJob creates a job object each time it runs. CronJobs are created, managed, scaled, and deleted in the same way as jobs. For more information about jobs, refer to the section-end project *Running a Job*.

First Step: Create a CronJob from the manifest file as shown below. This CronJob prints the current time and a string once per minute:

# cronjob.yaml

apiVersion: batch/v1beta1

kind: CronJob

metadata:

name: hello

spec:

schedule: "\*/1 \* \* \* \*"

jobTemplate:

spec:

template:

spec:

containers:

- name: hello

image: busybox

args:

- /bin/sh

- -c

- date; echo "Hello, World!"

restartPolicy: OnFailure

To create this CronJob, you can save the yaml manifest to a file and apply it to the cluster using: kubectl apply -f [FILENAME].

The subsequent steps provide more details about specifying when the CronJob runs and what actually runs.

**Step 3: Scheduling of CronJobs**

**Specifying when the CronJob runs**

The spec.schedule field defines when and how often the CronJob runs, using Unix standard crontab format. All CronJob times are in UTC. There are 5 fields separated by spaces. These fields represent the following:

* Minutes (between 0 and 59)
* Hours (between 0 and 23)
* Day of the month (between 1 and 31)
* Month (between 1 and 12)
* Day of the week (between 0 and 6)

You can use the following special characters in any of the spec.schedule fields:

* ? is a wildcard value that matches a single character.
* \* is a wildcard value that matches zero or more characters.
* /, allows you to specify an interval for a field. For instance, if the first field (the minutes field) has a value of \*/5, it means "every 5 minutes". If the fifth field (the day-of-week field) is set to 0/5, it means "every fifth Sunday."

**Specifying what the CronJob runs**

The spec.jobTemplate describes what the CronJob does, including its container images, the commands the containers execute, and the restart policy for the CronJob.

**Specifying a deadline**

The optional startingDeadlineSeconds field indicates the maximum number of seconds the CronJob can take to start if it misses its scheduled time for any reason. Missed CronJobs are considered failures.

To specify a deadline, add the startingDeadlineSeconds value to the CronJob's spec field in the manifest file. For example, the following manifest specifies that the CronJob has 100 seconds to begin:

apiVersion: batch/v1beta1

kind: CronJob

metadata:

name: hello

spec:

schedule: "\*/1 \* \* \* \*"

startingDeadlineSeconds: 100

jobTemplate:

spec:

...

If you do not specify a startingDeadlineSeconds value, the CronJob never times out. This could lead to the same CronJob running multiple times simultaneously.

**Inspecting a CronJob**

To check a CronJob's configuration, use kubectl describe:

kubectl describe cronjob [CRON\_JOB]

**Step 4: Viewing CronJob history/monitoring CronJob**

A CronJob runs within a pod. By default, Kubernetes preserves the logs for the terminated pods, representing the last three successful runs of a CronJob and the single last failed job. You can change or disable these defaults.

To view a CronJob's history, first list all the pods. The completed CronJobs are shown with a status of Completed, and failed jobs have a status of RunContainerError, CrashLoopBackOff, or another status indicating a failure.

kubectl get pods

You can verify the running CronJob by using kubectl get cronjob <jobname> and also see the scheduled time it needs to run after the defined interval as shown below: