Если вы видите что-то необычное, просто сообщите мне.

Как разворачивать MongoDB в Kubernetes

Как развернуть и запустить высокодоступный сервис MongoDB в Kubernetes, используя deployments, secret, configMaps и persistent volumes.

MongoDB это проект с открытым исходным кодом, главной целью которого,документо ориентированная распределенная NOSQL база данных, которая в основном популярна на JavaScript проектах. В этой инструкции вы научитесь разворачивать и запускать MongoDB в Kubernetes.

Содержание

- Изучим безопасное использование ключей с помощью Kubernetes secrets.
- Изучим настроку MongoDB используя ConfigMaps.
- Изучим использование Kubernetes Deployment для высокой доступности.
- Изучим хранение данных с помощью Presistent Volume Claims.

- Изучим бэкапирование базы данных используя Kubernetes CronJobs.

MongoDB Docker образ

К сожалению, нет официального образа для MongoDB. Доступно однако образы сообщества Docker. Как с любым неофициальным образом, рекомендую вам просканировать образы на проблемы и проанализировать Dockerfile.

Dockerfile и скрипты сборки MongoDB могут быть найдены в Docker сообществе.

Настройка MongoDB Docker

Скаченый образ предоставляет несколько переменных используемых для настройки базы данных.

```
MONGO_INITDB_ROOT_USERNAME
MONGO_INITDB_ROOT_PASSWORD
MONGO_INITDB_DATABASE
```

Первые два используются для логина и пароля администратора сервера. В основном для MongoDB достаточно задать два параметра.

Если MONGO_INITDB_ROOT_USERNAME и MONGO_INITDB_ROOT_PASSWORD не заданы значениями по-умолчанию, любой кто подключается к бд имеет полный доступ.

Secrets

Secrets предоставляет хранилище для безопасного хранения чувствительной информации в Kubernetes. Данные хранятся как секреты зашифрованные в `base64`, для сокрытия данных когда отображается на экране, и хранится зашифрованно на ETCD базе данных используя Kubernetes. Два конфигурационных значение которые выдаются как дополнительная защита MONGO_INITDB_ROOT_USERNAME и MONGO_INITDB_ROOT_PASSWORD.

Чтобы создать Kubernetes секрет для MongoDB супер пользователя и пароль, вы должны запустить следующую команду.

```
kubectl create secret generic mongodb \
--from-literal="root" \
--from-literal='my-super-secret-password'
```

Как вариант, Secret манифест(secreat.yaml) может быть создать и применен для создания.

Манифест Secret не должен хранится в git, особенно в той же директории где другие манифесты. Kubernetes хранит секреты как base64 зашифрованная строка. Манифест позволяет выбирать написание значения секрета: как обычная строка или base64 зашифрованная строка. Чтобы предотвратить возможность увидеть пароль через ваше плечё есть предложение всегда кодировать пароль в base64.

Для того. чтобы закодировать пароль в строку используюя base64 используйте следющую команду:

```
echo -n 'my-super-secret-password' | base64
| Output
bXktc3VwZXItc2VjcmV0LXBhc3N3b3Jk
```

Создайте новый файл с именем `mongodb-secrets.yaml` и добавьте в него следующеее содеражние. Секреты может быть сохранен в манифесте как `data` или `stringData`. Все данные в ключе `data` должны быть зашифрованны как base64. Значения лежащие в ключе `stringData` не требуют шифрования base64.

```
apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
data:
  MONGO_INITDB_ROOT_PASSWORD: bXktc3VwZXItc2VjcmV0LXBhc3N3b3Jk
stringData:
  MONGO_INITDB_ROOT_USERNAME: myroot
```

Применяем манифест чтобы создать ресурс в Kubernetes кластере.

```
kubectl apply -f mongodb-secrets.yaml
```

MongoDB Deployment манифест

Deployemnt описывает желаемое состояние которое создаст Deployment Controller. Когда состояние деплойментя отойдет от желаемого, Deployment Controller произведет действия чтобы вернуть желаемое состояние.

Один из примеров управления состоянием это убедится в количестве реплик работающих подов. Если под деплоймента упадет Deployment Controller заменит упавший под.

Обновления и откаты: Когда шаблон пода обновлен в Deployment ресурсе, Deployment Controller разверент обновленные поды прежде чем удалять старые. Новые поды не заменят старые до тех пор, пока не запустятся со здоровым состояниме.

Создадим новый файл mongodb-deployment.yaml и добавим следующее содержание.

```
apiVersion: app/v1
kind: Deployment
metadata:
  name: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongodb
        image: mongodb:3.6.19-xenial
        ports:
          containerPort: 27017
```

Пример выше развернет одну реплику MongoDB сущности. Базовый образ использует Docket Community MongoDB v3.6.19 основанные на Ubuntu Xenial.

Применим манифест чтобы создать деплоймент ресуср в Kubernetes.

```
kubectl apply -f mongodb-deployment.yaml
```

Свой MongoDB конфиг файл

Наш MongoDB под использует стандартные настройки для вновь созданного сервера. Только 2 параметра сконфигурированны это логин и пароль.

В MongoDB сервере есть множество настроеке. Однако, эти настройки задаются в `mongodb.conf` файле.

Создаем ConfigMap

Чтобы использовать `mongodb.conf` файл в нашем экземпляре в Kubernetes вы должны создать файл и сохранить его как `configMap` сущность.

Возьмем за пример файл `mongodb.conf`:

```
systemLog:
   destination: file
   path: "/var/log/mongodb/mongod.log"
   logAppend: true
storage:
   journal:
      enabled: true
processManagement:
   fork: true
net:
   bindIp: 127.0.0.1
   port: 27017
setParameter:
   enableLocalhostAuthBypass: false
```

Чтобы создать `configMap` сущность из `mongodb.conf` файла, запустите Kubernetes команду. Пример ниже создаст `configMap` с именем `mongodb`, если он еще не существует, и добавить `mongodb.conf` в сщуность под названием `conf`.

```
kubectl create configMap mongodb-config-file --from-file=conf=mongodb.conf
```

Монтирование mongodb.conf файла

Самый простой способ использования конфигурационного файла в MongoDB под это монтированием файла в качестве `volume`. После этого конфигурационный файл будет доступен как файл MongoDB сервиса когда запустится контейнер.

Для монтирования файла в качестве `volulme` нам нужно обновить наш Deployemtn манифест.

```
apiVersion: app/v1
kind: Deployment
metadata:
  name: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongodb
        image: mongodb:3.6.19-xenial
        ports:
          containerPort: 27017
        volumeMounts:
        - name: mongodb-configuration-file
          mountPath: /etc/mongod.conf
          readOnly: true
    volumes:
    - name: mongodb-configuration-file
      configMap:
        name: mongodb-config-file
```

Постоянное хранилище `Persistent Volumes`

Контейнеры недолговечны по определению. Любое состояние контенейра будет потеряно после перезапуска. Для баз данных как MongoDB, это значит что вся ваша база данных будет стерта.

`Persistent Volumes` может быть смонтированно в Pod позволяя сохранять данные вне зависимости от наличия контенера. Чтобы добавить `Persistent Volumes` в контейнер в Kubernetes нужно создать `Persistent Volumes Claim` и затем смонтировать `volume` к `Deployment`.

Создадим файл под названием mongodb-pvc.yaml и добавим в него:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pv-claim
  labels:
    app: mongodb
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

`Volume claim` создаст хранилие в которое можно будет что-то писать размером в 1GB.

Применим его к нашему Kuberentes кластеру.

```
kubectl apply -f mongodb-pvc.yaml
```

Чтобы смонтировать раздел нам нужно обновить наш deployemtn манифест. `volume` нужно боавить в `template` ключ нашего манифеста, а `volumeMount` в ключ контенера, чтобы смонтировать раздел.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongodb
        image: mongo:4.4.0-bionic
        ports:
          - containerPort: 27017
        envFrom:
        - secretRef:
            name: mongodb-secret
        volumeMounts:
        - name: mongodb-configuration-file
          mountPath: /etc/mongod.conf
          readOnly: true
        - name: mongodb-persistent-storage
          mountPath: /data/db
      volumes:
      - name: mongodb-persistent-storage
        persistentVolumeClaim:
          claimName: mongodb-pv-claim
      - name: mongodb-configuration-file
        configMap:
          name: mongodb-config-file
```

Применим новый деплоймент manifest чтобы обновить сущействующее или создать новое развертывание.

```
kubectl apply -f mongodb-deployment.yaml
```

Выставление сервисов

Мы развернули один под MongoDB. Выставление пода в виде сервиса настоятельно не рекомендуется вне тестирования или разработки. Поды недолговечны и как только они остановятся их состояние потеряется, в том числе назначенный ip адрес.

Внустренний сервис.

Внутренний сервис это сервис который доступен только внутри кластера Kubernetes. Это стандартное повдение сервиса. Для базы данных сервер и других системы это лучший вариант настройки.

Создадим новый сервис для MongoDB.

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb
spec:
  selector:
    app: mongodb
  ports:
  - protocol: TCP
    port: 27017
```

Применим манифест:

```
kubectl apply -f mongodb-service.yaml
```

Backing Up MongoDB

Всё что было сказано выше совершенно не содержит шагов отвечающих за безопасность системы. Есть несколько методов для бэкапирования MongoDB в Kubernetes, но эта инструкция сфокусируется на создании расписания в Kubernetes в качествет CronJob.

CronJob

CronJob это расписание контейнеризированных работ.

CronJob бэка для MongoDB будет делать следующее:

- Запустит MongoDB контейнер
- Смонтирует раздел используемый MongoDB
- Выполнит команду `mongodump`
- Скопирует dump в корзину хранилища, к примеру Google Cloud Storage Bucket.

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: mongodb-backup
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: mongodb-backup
            image: mongo:4.4.0-bionic
            args:
            - "/bin/sh"
            - "-c"
            - "/usr/bin/mongodump -u $MONGO_INITDB_ROOT_USERNAME -p
$MONGO_INITDB_ROOT_PASSWORD -o /tmp/backup -h mongodb"
            - "tar cvzf mongodb-backup.tar.gz /tmp/backup"
            #- gsutil cp mongodb-backup.tar.gz gs://my-project/backups/mongodb-backup.tar.gz
            envFrom:
            - secretRef:
                name: mongodb-secret
            volumeMounts:
            - name: mongodb-persistent-storage
              mountPath: /data/db
          restartPolicy: OnFailure
          volumes:
          - name: mongodb-persistent-storage
            persistentVolumeClaim:
              claimName: mongodb-pv-claim
```

Управлвение MongoDB in Kubernetes

Так как MongoDB не должно быть доступно вне нашего кластера. Оператор до сих пор может подключиться к нему.

Перенапралвение портов.

Перенаправление портов с помощью kubectl позволяет нам создать проксированное подключение к вашей машине в Kubernetes сервисе. Для MongoDB то значит мы можем создать mongodb подключение к нашему серверу.

```
kubectl port-forward mongodb-service 27017 &
| Output

Forwarding from 127.0.0.1:27017 -> 27017
Forwarding from [::1]:27017 -> 27017
```

С помощью полученного порта MongoDB на нашей локальной машине, мы можем использовать mongo клиента для подключения.

```
mongo -u <root-username> -p <root-password>
| Output

MongoDB shell version v4.2.0
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Handling connection for 27017
Implicit session: session { "id" : UUID("feb9a859-43eb-4cb3-bdd9-ef76690cbb92") }
MongoDB server version: 3.6.19
WARNING: shell and server versions do not match
Server has startup warnings:
2020-08-24T03:21:23.861+0000 I STORAGE  [initandlisten]
2020-08-24T03:21:23.861+0000 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem
is strongly recommended with the WiredTiger storage engine
2020-08-24T03:21:23.861+0000 I STORAGE  [initandlisten] **          See
http://dochub.mongodb.org/core/prodnotes-filesystem
>
```

Интерактивная консоль в контейнере

Как замена, можно открыть интерактивную консоль внутри рабочего MongoDB контейнера. Для этого нужно название MongoDB пода чтобы зайти в него.

```
kubectl get pods
| Output

NAME                       READY   STATUS    RESTARTS   AGE
mongodb-7cfbc6f555-t97c4   1/1     Running   0          12m
```

Так как у нас только одна реплика и её имя mongodb-7cfbc6f555-t97c4. Теперь зная имя нашего пода можно подключиться к нему.

```
kubectl exec -it mongodb-7cfbc6f555-t97c4 /bin/bash
| Output

root@mongodb-7cfbc6f555-t97c4:/#
```

Когда запустится shell в контейнере мы можем управлять MongoDB ипользую его внутренную консоль.

```
mongo -u $MONGO_INITDB_ROOT_USERNAME -p $MONGO_INITDB_ROOT_PASSWORD
```

В завершении

В этой инструкции мы обсудили как разворачивать MongoDB в Kubernetes и управлять им. Научились как безопасно хранить данные для доступа к вашей бд. Как настроить сервис для доступности его внутри кластера.

Одна из важных тем это бэкапирование, которую мы развернули в качесте `CronJob`.

Revision #7
Created 2021-10-31 16:10:09 UTC by gasick
Updated 2023-04-16 19:36:18 UTC by gasick