Инженерный вестник Дона, №3 (2021)
ivdon.ru/ru/magazine/archive/n3y2021/6844

Автоматизация получения доступа к базам данных для компонентов в среде Kubernetes

Н.Б. Лазарева, Н.Н. Ловцова
Тихоокеанский государственный университет, Хабаровск

Аннотация: В данной статье описывается процесс автоматизации процедуры получения критических с точки зрения безопасности данных, необходимых для работы компонентов с базами данных в среде Kubernetes; рассмотрены объекты Kubernetes, логика их работы, а также их взаимодействие с хранилищем Vault. Для управления объектами Kubernetes и автоматизации операций в процессе деплоя используется ПО Helm. В статье приведены необходимые конфигурации для Helm с пояснениями. В качестве примера в статье рассматривается случай взаимодействия компонентов с PostgreSQL и MongoDB как широко распространенными в информационных средах SQL и NO-SQL базами данных.
Ключевые слова: Kubernetes, Vault, Helm, автоматизация, безопасность, база данных, Deployment, Job, PostgreSQL, MongoDB.

Часто в различных информационных средах компоненты должны иметь доступ к различным базам данных. Для этого коду этих компонентов нужно передать логин и пароль, имя базы данных и адрес базы данных. Адрес базы данных и имя базы это параметры, которые могут быть переданы компоненту динамически в момент его деплоя в открытом виде. Но если говорить о таких параметрах, как логин и пароль, то здесь имеет место проблема обеспечения безопасности хранения и передачи этих параметров. Очевидно, что поскольку нужно сообщить эти данные компоненту, необходимо их заранее создать и где-то сохранить. Также нужно иметь в виду, что после передачи этих параметров компоненту сами параметры не должны быть доступны извне, а быть доступны только компоненту.

Для хранения паролей и любых других, важных с точки зрения безопасности, данных, существует распространенное решение Hashicorp Vault [1]. В данной статье опустим процедуру конфигурации и деплоя Vault в Kubernetes, поскольку она довольно тривиальна и описана в официальной документации.

Необходимо продумать и реализовать процесс получения данных из Vault в процессе деплоя компонента в Kubernetes так, чтобы максимально снизить вероятность получения этих данных другими компонентами.

Предположим, компоненту необходим доступ в две базы данных: PostgreSQL и MongoDB. В Vault ответственное лицо поместило ряд ключевых переменных:

- mongodbAdminPassword – пароль администратора
- mongodbUserPassword – уникальный пароль компонента
- postgresqlAdminPassword – пароль администратора
- postgresqlUserPassword– уникальный пароль компонента

Схема использования этих переменных следующая:

1. Во время деплоя компонента непосредственно перед его запуском временно запускает контейнер с клиентским приложением Vault, который забирает из сервера Vault указанные данные и сохраняет их в директории, которая будет доступна на шаге 2. После этого он завершает работу.
2. Запускается второй временный контейнер с кодом, который осуществляет подготовку базы данных для использования компонентом. Непосредственно перед запуском кода происходит публикация сохраненных данных на шаге 1 в виде переменных окружения, доступных коду.
3. В этом контейнере код, используя пароль администратора (или любой другой учетной записи, предварительно созданной в экземпляре базы данных и обладающей правами на создание базы данных, а также на создание пользователя с назначенным паролем и на смену этого пароля), подключается к экземпляру базы данных. Далее в случае PostgreSQL [2] создается пользователь и задается пароль, используя переменную postgresqlUserPassword. Этому пользователю

выдаются права на создание собственной базы данных. После этого происходит переключение с учетной записи администратора на вновь созданную и уже от ее имени создается база данных. В случае MongoDB [3] с использованием учетной записи администратора создается только пользователь и пароль с использованием переменной mongodbUserPassword, база данных будет создана автоматически при первой попытке что-либо записать в нее.

4. Временно запускается контейнер с клиентским приложением Vault, который забирает из сервера Vault указанные данные и сохраняет их в директории, которая будет доступна компоненту. После этого он завершает работу.

5. Запускается контейнер с компонентом, но непосредственно перед запуском кода компонента происходит публикация сохраненных данных на шаге 4 в виде переменных окружения, доступных компоненту. Используя созданные учетные записи и пароли, загруженные из переменных окружения, компонент может выполнить подключение к базам данных.

Реализовать подобную схему можно с помощью пакетного менеджера Helm [4]. Helm позволяет упростить и автоматизировать процесс деплоя компонентов в Kubernetes, являясь широко распространенным ПО. Кроме того, Helm несколько расширяет возможности деплоя компонентов, добавляя определенный функционал. В данном случае используются Helm-хуки, а также возможность генерации переменных на лету при помощи механизма helpers. Рассмотрим компонент Job [5], необходимый для запуска кода, ответственного за подготовку баз данных (рис. 1):

```
apiVersion: batch/v1
kind: Job
metadata:
  name: {{ .Values.name }}-createdb
  labels:
    app: {{ .Values.name }}
    chart: {{ .Values.name }}
    release: {{ .Values.name }}
  annotations:
    "helm.sh/hook": pre-install, pre-upgrade
    "helm.sh/hook-delete-policy": hook-succeeded,before-hook-creation
    "helm.sh/hook-weight": "10"
spec:
  template:
    metadata:
{{- if .Values.createdb.annotations }}
      annotations:
{{ toYaml .Values.createdb.annotations | indent 8 }}
{{- end }}
      name: {{ .Values.name }}-createdb
    spec:
      restartPolicy: Never
      containers:
      - name: createdb
        image: "{{ .Values.docker.registry }}/{{ .Values.docker.db.image }}:{{ .Values.docker.db.tag }}"
        command:
        - /bin/bash
        - -c
        - |-
          source /vault/secrets/config
          export PGUSER={{ .Values.createdb.postgresql.adminuser | quote }}
          export PGPASSWORD=$PGADMINPASSWORD
          if echo -n > /dev/tcp/$PGHOST/$PGPORT; then
            if ! psql --dbname={{ .Values.createdb.postgresql.adminuser }} -qAtc \
              "SELECT usename from pg_user"|grep -qw {{ .Values.createdb.postgresql.user }}; then
              psql --dbname={{ .Values.createdb.postgresql.adminuser }} --command=\
              "CREATE USER {{ .Values.createdb.postgresql.user }} PASSWORD '$PGUSERPASSWORD'";
            fi
            psql --dbname={{ .Values.createdb.postgresql.adminuser }} --command=\
            "ALTER USER {{ .Values.createdb.postgresql.user }} WITH CREATEDB";
            psql --dbname={{ .Values.createdb.postgresql.adminuser }} --command=\
            "ALTER USER {{ .Values.createdb.postgresql.user }} WITH PASSWORD '$PGUSERPASSWORD'";
            if ! psql --dbname=postgres -lqt | cut -d \| -f 1 | grep -qw {{ .Values.createdb.postgresql.dbname }}; then
              export PGUSER={{ .Values.createdb.postgresql.user }};
              export PGPASSWORD=$PGUSERPASSWORD;
              createdb -O {{ .Values.createdb.postgresql.user }} {{ .Values.createdb.postgresql.dbname }};
            fi
          else
            exit 1;
          fi
          if echo -n > /dev/tcp/{{ .Values.createdb.mongodb.host }}/{{ .Values.createdb.mongodb.port }}; then
            if ! mongo {{ .Values.createdb.mongodb.host }}:{{ .Values.createdb.mongodb.port }}/admin -u \
            {{ .Values.createdb.mongodb.adminuser }} -p $mongodbAdminPassword --eval \
            {{ include "mongodb_check_user" $ | quote }} | grep '\"user\" : \"{{ .Values.name }}\"'; then
              mongo {{ .Values.createdb.mongodb.host }}:{{ .Values.createdb.mongodb.port }}/admin -u \
              {{ .Values.createdb.mongodb.adminuser }} -p $mongodbAdminPassword --eval \
              {{ include "mongodb_create_user" $ | quote }};
            fi
          else
            exit 1;
          fi
        env:
        - name: PGHOST
          value: {{ .Values.createdb.postgresql.pghost | quote }}
        - name: PGPORT
          value: {{ .Values.createdb.postgresql.pgport | quote }}
        - name: PGDATABASE
          value: {{ .Values.createdb.postgresql.dbname | quote }}
```

Рис. 1. – Пример компонента Job

Как видно из данного примера, необходимо сообщить Helm о том, что нужно выполнить этот код до запуска основного кода компонента. Это регулируется

блоком.metadata.annotations [6]. Далее блок.spec.template.metadata.annotations подгружает специальный блок аннотаций, который позволяет использовать переменные, полученные на шаге 1 после выгрузки из Vault. Данный блок хранится в файле values.yaml [7], который будет рассмотрен позднее.

Блок spec.template.metadata.annotations.spec.containers описывает шаги 2 и 3, в случае PostgreSQL, используя переменную окружения PGPASSWORD, получающую свое значение из данных, полученных на шаге 1, а также переменных, значения которых могут быть переданы в открытом виде, такие, как PGHOST (адрес базы данных), PGPORT (порт базы данных), PGDATABASE (имя базы данных), PGUSER (пользователь, от имени которого выполняются команды). В случае с MongoDB используются данные, полученные на шаге 1 напрямую, в виде переменных mongodbAdminPassword и mongodbUserPassword без создания переменных окружения, так как нельзя использовать переменные окружения для работы с MongoDB. Для работы с MongoDB используются специально подготовленные переменные mongodb_check_user, mongodb_create_user, по существу являющиеся консольными командами. Подготовка этих команд осуществляется в файле _helpers.tpl [8] (рис. 2):

```
{{- define "mongodb_create_user" -}}
db = db.getSiblingDB('{{ .Values.createdb.mongodb.dbname }}');
db.createUser({user: '{{ .Values.name }}', pwd: "$mongodbUserPassword",
roles: [ { role: 'readWrite', db: '{{ .Values.createdb.mongodb.dbname }}' } ]})
{{- end -}}

{{- define "mongodb_check_user" -}}
db = db.getSiblingDB('{{ .Values.createdb.mongodb.dbname }}'); db.getUsers()
{{- end -}}
```

Рис. 2 – Содержимое файла _helpers.tpl

Для того, чтобы данный объект Job смог использовать данные, полученные на шаге 1, нужно добавить специальные аннотации в блок .spec.template.metadata.annotations. Для этого в файле Values.yaml нужно добавить блок аннотаций (рис. 3):

```
createdb:
  annotations:
    vault.hashicorp.com/agent-inject: "true"
    vault.hashicorp.com/role: "all"
    vault.hashicorp.com/agent-inject-secret-config: "secret/component-name/config"
    vault.hashicorp.com/agent-pre-populate-only: "true"
    vault.hashicorp.com/agent-inject-template-config: |
      {{ with secret "secret/component-name/config" -}}
        export mongodbAdminPassword="{{ .Data.data.mongodbAdminPassword }}"
        export mongodbUserPassword="{{ .Data.data.mongodbUserPassword }}"
        export PGADMINPASSWORD="{{ .Data.data.postgresqlAdminPassword }}"
        export PGUSERPASSWORD="{{ .Data.data.postgresqlUserPassword }}"
      {{- end }}
```

Рис. 3 – Блок аннотаций для объекта Job

Кроме этого, нужно передать другие переменные, необходимые для работы Job (рис. 4):

```
name: "component-name"

docker:
  db:
    image: "docker-image-name"
    tag: "docker-tag"

createdb:
  postgresql:
    pghost: &pghost "postgresql-name"
    pgport: &pgport "5432"
    adminuser: "postgres"
    user: "component-name"
    dbname: "component-name"
  mongodb:
    host: &mhost "mongodb-name"
    port: &mport "27017"
    adminuser: "root"
    dbname: &mdbname "component-name"
    user: &muser "component-name"
```

Рис. 4 – Дополнительные переменные для объекта Job

С наличием данной конфигурации, при деплое будет запущен контейнер с кодом внутри, который подготовит базы к использованию. Останется лишь подготовить контейнер с кодом компонента. В случае, если контейнер компонента запускается путем создания объектов Pod [9] и

Deployment [10], нужно добавить блок аннотаций (рис. 5) по аналогии с блоком аннотаций для объекта Job:

```
annotations:
  vault.hashicorp.com/agent-init-first: "true"
  vault.hashicorp.com/agent-inject: "true"
  vault.hashicorp.com/role: "all"
  vault.hashicorp.com/agent-inject-secret-config: "secret/component-name/config"
  vault.hashicorp.com/agent-pre-populate-only: "true"
  vault.hashicorp.com/agent-inject-template-config: |
    {{ with secret "secret/component-name/config" -}}
      export component_name_postgresql_password="{{ .Data.data.mongodbUserPassword }}"
      export component_name_mongodb_password="{{ .Data.data.postgresqlUserPassword }}"
    {{- end }}
```

Рис. 5 – Блок аннотаций для объекта Deployment

Отличие данного блока аннотаций от похожего блока для Job состоит в том, что компоненту передаются только пароли от вновь созданных учетных записей. Таким образом, компонент не может использовать пароли администратора. Подключить данный блок аннотаций нужно в блоке Deployment .spec.template.metadata.annotations.

Также нужно сообщить контейнеру, что перед непосредственным запуском кода компонента нужно выполнить загрузку данных, полученных на шаге 4. Для этого необходимо модифицировать команду запуска контейнера в объекте Deployment:

service_cmd: "[\"/bin/bash\", \"-c\", \"source /vault/secrets/config && npm start\"]"

вставив её в блок . spec.template.spec.containers.command.

Теперь после запуска кода компонент сможет выполнить подключение к базам данных, используя свои уникальные пары логин/пароль. Данные, выгруженные на шаге 1, после запуска компонента более недоступны. Данные, выгруженные на шаге 4, являются доступными только данному компоненту, другие компоненты к ним доступа не имеют.

Таким образом, используя Helm и Vault, удалось автоматизировать процесс получения данных, необходимых для подключения к базам данных, обеспечив высокий уровень безопасности процесса.

Литература

1. Официальная документация Vault // vaultproject.io. URL: vaultproject.io/docs/platform/k8s (дата обращения: 14/02/2021).
2. PostgreSQL документация // postgresql.org. URL: postgresql.org/docs/ (дата обращения: 14/02/2021).
3. MongoDB документация // mongodb.com. URL: docs.mongodb.com/.
4. Официальная документация Helm // helm.sh URL: helm.sh/docs/intro/.
5. Job resource for Kubernetes // kubernetes.io. URL: kubernetes.io/docs/concepts/workloads/controllers/job/ (дата обращения: 14/02/2021).
6. Helm hooks // helm.sh. URL: helm.sh/docs/topics/charts_hooks/ (дата обращения: 14/02/2021).
7. Файлы значений Helm // helm.sh. URL: helm.sh/docs /chart_ template_ guide/values_files/ (дата обращения: 14/02/2021).
8. Именованные шаблоны Helm // helm.sh. URL: helm.sh/docs/ chart_ template_guide/named_templates/.
9. Описание ресурса Pod для Kubernetes // kubernetes.io. URL: kubernetes.io/ docs/concepts/workloads/pods/ (дата обращения: 14/02/2021).
10. Описание ресурса Deployment для Kubernetes // kubernetes.io. URL: kubernetes.io/docs/concepts/workloads/controllers/deployment/ (дата обращения: 14/02/2021).

References

1. Oficial`naya dokumentaciya Vault [Vault documentation]. URL: vaultproject.io/docs/platform/k8s (data obrashheniya: 14/02/2021).

2. PostgreSQL dokumentaciya. URL: postgresql.org/docs/ (data obrashheniya: 14/02/2021).
3. MongoDB dokumentaciya. URL: docs.mongodb.com/.
4. Oficial`naya dokumentaciya Helm [Helm documentation]. URL: helm.sh/docs/intro/.
5. Job resource for Kubernetes. URL: kubernetes.io/docs/concepts/ workloads/controllers/job/ (data obrashheniya: 14/02/2021).
6. Helm hooks. URL: helm.sh/docs/topics/charts_hooks/ (data obrashheniya: 14/02/2021).
7. Fajly` znachenij Helm [Helm values files]. URL: helm.sh/docs/chart_template_guide/values_files/ (data obrashheniya: 14/02/2021).
8. Imenovanny`e shablony` Helm [Helm named templates]. URL: helm.sh/docs/chart_template_guide/named_templates/.
9. Opisanie resursa Pod dlya Kubernetes [Pod resource for Kubernetes]. URL: kubernetes.io/docs/concepts/workloads/pods/ (data obrashheniya: 14/02/2021).
10. Opisanie resursa Deployment dlya Kubernetes [Deployment resource for Kubernetes]. URL: kubernetes.io/docs/concepts/workloads/ controllers/deployment/ (data obrashheniya: 14/02/2021).