KUB 29: Основы пакетного менеджера Helm

Описание:

В процессе работы и доставки кода в инфраструктуру kubernetes инженеры сталкиваются с типовым набором проблем, которые необходимо решить, чтобы указанные процессы были надежны и вариативны. Вы уже представляете себе, как задеплоить приложение в k8s, но давайте попробуем представить, какие проблемы нам ещё потребуется решить, когда мы начнем обслуживать приложение, которое разместили в k8s:

1. Возможность отката. Самый первый и острый вопрос, который мы должны рассмотреть. Если мы настраиваем инструмент для доставки кода, то вопрос возможности максимально быстрого отката до определенного состояния возникает сам собой. Да, конечно, в kubectl есть возможность откатить состояние, допустим, deployment, но попробуйте это автоматизировать и сами все поймете.
2. Вариативность. Как следствие первого пункта и как отдельная задача. В случае внесения изменений в функционирование контейнера с приложением, изменения переменных окружения или каких-либо иных аспектов, связанных с работой нашего кода, мы должны иметь дружелюбный интерфейс, с помощью которого можно произвести эти изменения. Самый базовый вопрос — версионирование тега image контейнера с кодом из container registry. Sed'ы?
3. Удобство использования. Само собой, можно выкрутиться и kubectl apply -f a lot of files, но чем больше элементов, с которыми мы работаем, тем больше потенциальных возможностей для ошибок.

Helm позволяет решить указанные проблемы. Мы можем собрать требуемое описание ресурсов в одном месте, шаблонизировать необходимые аспекты для удобной работы (совсем как ansible templates) и работать с полученной группой ресурсов как с единым элементом.

Как уже было сказано, Helm — это пакетный менеджер для чартов (Charts). Но что же такое чарты?

В простейшем описании чарт представляет из себя структуру файлов, которые описывают требуемое состояние ресурсов. Эти файлы можно условно разделить на обязательные и необязательные. Давайте посмотрим подробнее (подобную структуру можно сгенерировать при помощи helm create):

```
/...
/templates/              — можно сказать, сердце чарта,
директория, которая содержит шаблоны манифестов ресурсов
Kubernetes.
/templates/NOTES.txt     — необязательный файл с примечанием,
которое будет выводиться пользователю при работе с чартом.
```

```
/charts/                  — каталог с вложенными чартами
(необязателен).
/Chart.yaml               — файл с общей информацией о чарте.
/LICENSE                  — файл с лицензией чарта
(необязателен).
/README.md                — файл с описанием/документацией чарта
(необязателен).
/requirements.yaml        — файл со списком зависимостей (иных
чартов, требуемых для работы текущего, необязателен).
/values.yaml              — файл со значениями переменных для
шаблонов.
```

В целом, в данном случае можно отбросить все неважные элементы для начала работы с Helm. При создании чарта после описания ряда информационных файлов вашей задачей, по сути, будет верное оформление темплейтов (шаблонов манифестов). В качестве шаблонизатора Helm использует дополненный Golang-шаблонизатор.
{{ }}
Фигурные скобки являются указателем на то, что внутри содержится шаблонизированное значение. Все, что не ставится в фигурные скобки, при рендере чарта остается неизменным.
.
Точка — это переменная контекста, можно сказать, что это указатель для подстановки значений, содержащихся в структуре данных чарта.
Даже в общих чертах ознакомление со структурой шаблонизатора — достаточно обширная тема, и в рамках саммари проще рассмотреть практические примеры:

```
{{- if .Capabilities.APIVersions.Has "apps/v1beta2" }}
apiVersion: apps/v1beta2
{{- else }}
apiVersion: extensions/v1beta1
{{- end }}
metadata:
  labels:
    chart: "{{ .Chart.Name }}"
    release: "{{ .Release.Name }}"
containers:
- name: nginx
  image: "nginx:{{ .Values.ImageTag }}"
...
  data:
    nginx.conf: |
{{ .Files.Get "configs/nginx.conf" | indent 6 }}
```

Итак, .Chart — это структура данных, содержит информацию о чарте и описывается в файле Chart.yaml. Release содержит данные о релизе, инсталляции и обновлении, такие как имя релиза, неймспейс, в который производится выкладка, и т.д. По сути, это метаинформация о нашем релизе. И наконец, .Values — это параметры, определенные в values.yaml опциями --set — именно тут параметризуется «пользовательская» информация. Capabilites — это доступная информация о кластере, в который производится выкатка приложения. Files — структура, содержащая информацию о файлах, которые хранятся в директории чарта.
В процессе работы с шаблонизатором вам может потребоваться множество раз в разных файлах описывать одну и ту же логику. К примеру, вы хотите описать какую-либо сложную составную структуру для формирования значения одного из labels, которыми будут отмечены все экземпляры ваших манифестов. В целом, описать это несложно, однако при внесении изменений вам необходимо будет править все манифесты темплейтов, что увеличивает шанс человеческой ошибки при правках. Также злоупотребление такими моментами делает структуры темплейтов нечитаемыми, становится сложнее распределять работу между сотрудниками, а новым людям потребуется неоправданно много времени для того, чтобы понять логику происходящего в темплейтах.
Такие экземпляры кода можно выносить во вспомогательные файлы, которые находятся в каталоге /templates/ и имеют расширение .tpl. Добавляя промежуточные переменные, содержащие сложную, составную логику, описанную во вспомогательных файлах, вы делаете темплейты намного более читаемыми и уменьшаете вероятность человеческих ошибок при правках.
В корневой директории чарта мы можем хранить файл с названием values.yaml. В нем будет содержаться массив данных, которые мы можем использовать как переменные в ходе работы с нашим helm-chart.
Переменные для helm-chart могут быть заданы:

1. Файл values.yaml в корневой директории чарта.
2. Если это вложенный чарт, файл values.yaml родительского чарта.
3. Дополнительные параметры values, содержащиеся в yaml-файле, могут быть переданы в чарт с помощью команды helm install/upgrade с флагом -f (helm install chart_name -f myvals.yaml ./mychart).
4. Отдельные параметры передаются с помощью --set (например, helm install chart_name --set foo = bar ./mychart).

Приведенный выше список имеет определенную специфику: данные, содержащиеся в родительском values.yaml, являются параметрами по умолчанию, в то же время они могут быть дополнены или переопределены с помощью дополнительных файлов, переданных в команде установки с ключом -f, либо же могут быть переопределены конкретные переменные с помощью флага --set.
Файлы значений — это простые YAML-файлы.
Также переменные, хранимые в values.yaml, можно структурировать, например:

```yaml
favorite:
  drink: coffee
  food: pizza
```

При указанной выше форме записи обращения к конкретным ключам будут выглядеть следующим образом:

```
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink }}
  food: {{ .Values.favorite.food }}
```

Полезные ссылки:

- Helm Docs

Задание:

1. Напишите helm-чарт для сервиса, который будет:
- создавать один deployment nginx-dp;
- создавать один сервис nginx-svc, который смотрит на этот деплоймент;
- создавать ingress nginx-ingress, который смотрит на этот сервис.
2. Задеплойте данный чарт с именем релиза local-chart в namespace helm.
3. Сохраните чарт локально - он понадобится в следующем задании.
4. Отправьте задание на проверку.