Создайте свою первую диаграмму Helm на платформе Базис. Digital Energy

Знакомство

 Итак, у вас есть кластер Kubernetes, развернутый на технологической платформе ПАО Ростелеком Базис. Digital Energy, запущенный и настроенный Helm v3.x, но как вы запускаете свои приложения на нем? Это руководство проведет вас через процесс создания вашей первой диаграммы, объяснив, что входит в эти пакеты и инструменты, которые вы используете для их разработки. К концу вы должны понять преимущества использования Helm для доставки собственных приложений в кластер.

Для типичного облачного приложения с 3-уровневой архитектурой на приведенной ниже схеме показано, как его можно описать в терминах объектов Kubernetes. В этом примере каждый уровень состоит из объекта [Deployment](http://kubernetes.io/docs/user-guide/deployments/) и [Service](http://kubernetes.io/docs/user-guide/services/) и может дополнительно определять [объекты ConfigMap](http://kubernetes.io/docs/user-guide/configmap/) или [Secret](http://kubernetes.io/docs/user-guide/secrets/). Каждый из этих объектов обычно определяется в отдельных файлах YAML и передается в программу командной строки kubectl.

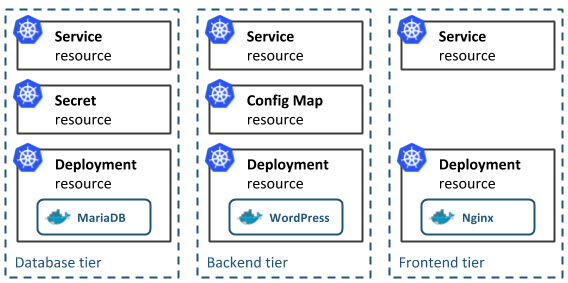


Диаграмма Helm инкапсулирует каждое из этих определений YAML, предоставляет механизм настройки во время развертывания и позволяет определять метаданные и документацию, которые могут быть полезны при совместном использовании пакета. Helm может быть полезен в различных сценариях:

* Поиск и использование популярного программного обеспечения, упакованного в виде диаграмм Kubernetes
* Совместное использование собственных приложений в виде диаграмм Kubernetes
* Создание воспроизводимых сборок приложений Kubernetes
* Интеллектуальное управление определениями объектов Kubernetes
* Управление выпусками пакетов Helm

Давайте рассмотрим второй и третий сценарии, создав нашу первую диаграмму.

Шаг 1: Создайте свою первую диаграмму

Лучший способ начать работу с новой диаграммой — использовать команду helm create для создания примера, на котором мы можем основываться. Используйте эту команду для создания новой диаграммы с именем mychart в новом каталоге:

helm create mychart

Helm создаст новый каталог в вашем проекте под названием mychart со структурой, показанной ниже. Давайте перейдем к нашему новому Chart, чтобы узнать, как это работает.

mychart

|-- Chart.yaml

|-- charts

|-- templates

| |-- NOTES.txt

| |-- \_helpers.tpl

| |-- deployment.yaml

| |-- ingress.yaml

| `-- service.yaml

`-- values.yaml

Шаблоны

Самая важная часть головоломки – это каталог templates/ . Здесь Helm находит определения YAML для ваших служб, развертываний и других объектов Kubernetes. Если у вас уже есть определения для приложения, все, что вам нужно сделать, это заменить созданные файлы YAML на свои собственные. В итоге получается рабочая диаграмма, которую можно развернуть с помощью команды helm install.

Однако стоит отметить, что каталог называется templates, и Helm запускает каждый файл в этом каталоге через движок рендеринга [шаблонов Go](https://golang.org/pkg/text/template/). Helm расширяет язык шаблонов, добавляя ряд утилитарных функций для написания Chart’s. Откройте файл service.yaml, чтобы узнать, как это выглядит:

apiVersion: v1

kind: Service

metadata:

name: {{ template "fullname" . }}

labels:

chart: "{{ .Chart.Name }}-{{ .Chart.Version | replace "+" "\_" }}"

spec:

type: {{ .Values.service.type }}

ports:

- port: {{ .Values.service.externalPort }}

targetPort: {{ .Values.service.internalPort }}

protocol: TCP

name: {{ .Values.service.name }}

selector:

app: {{ template "fullname" . }}

Это базовое определение службы с использованием шаблонов. При развертывании диаграммы Helm сгенерирует определение, которое будет больше похоже на допустимую службу. Мы можем выполнить  helm install и включить отладку для проверки сгенерированных определений:

helm install example --dry-run --debug ./mychart

...

# Source: mychart/templates/service.yaml

apiVersion: v1

kind: Service

metadata:

name: pouring-puma-mychart

labels:

chart: "mychart-0.1.0"

spec:

type: ClusterIP

ports:

- port: 80

targetPort: 80

protocol: TCP

name: nginx

selector:

app: pouring-puma-mychart

...

Valurs

Шаблон в service.yaml использует объекты, специфичные для Helm.Chart и .Values. Первый предоставляет метаданные о диаграмме для ваших определений, таких как имя или версия. Последнееобъект.Values является ключевым элементом диаграмм Helm, используемым для предоставления конфигурации, которая может быть задана во время развертывания. Значения по умолчанию для этого объекта определены в файле values.yaml. Попробуйте изменить значение по умолчанию для service.internalPort и выполнить еще один сухой запуск, вы обнаружите, что targetPort в службе и containerPort в развертывании изменяются. Значение service.internalPort используется здесь для обеспечения правильной совместной работы объектов Service и Deployment. Использование шаблонов может значительно уменьшить шаблон и упростить ваши определения.

Если пользователь диаграммы хочет изменить конфигурацию по умолчанию, он может предоставить переопределения непосредственно в командной строке:

helm install --dry-run --debug ./mychart --set service.internalPort=8080

Для более сложной настройки пользователь может указать файл YAML, содержащий переопределения с помощью *--values* парамета.

Помощники и другие функции

Шаблон service.yaml также использует частичные значения, определенные в \_helpers.tpl, а также такие функции, как replace. [Документация Helm](https://helm.sh/docs/chart_template_guide/getting_started/) содержит более глубокое пошаговое руководство по языку шаблонов, объясняющее, как функции, частичные части и управление потоком могут использоваться при разработке диаграммы.

Документация

Другим полезным файлом в каталоге templates/ является файл NOTES.txt. Это шаблонный текстовый файл, который распечатывается после успешного развертывания диаграммы. Как мы увидим, когда мы развернем нашу первую диаграмму, это полезное место для краткого описания следующих шагов по использованию диаграммы. Поскольку NOTES.txt выполняется через модуль шаблонов, вы можете использовать шаблоны для распечатки рабочих команд для получения IP-адреса или получения пароля от объекта Secret.

Метаданные

Как упоминалось ранее, диаграмма Helm состоит из метаданных, которые используются для описания того, что представляет собой приложение, определения ограничений на минимально необходимые Kubernetes и/или версии Helm и управления версией диаграммы. Все эти метаданные находятся в файле Chart.yaml. [Документация Helm](https://helm.sh/docs/) описывает различные поля для этого файла.

Шаг 2. Развертывание первой диаграммы

Диаграмма, созданная на предыдущем шаге, настроена на запуск сервера NGINX, предоставляемого через службу Kubernetes. По умолчанию диаграмма создает службу типа ClusterIP, поэтому NGINX будет предоставляться только внутри кластера. Чтобы получить к нему внешний доступ, мы будем использовать тип NodePort. Мы также можем установить название выпуска Helm, чтобы мы могли легко вернуться к нему. Давайте продолжим и развернем нашу диаграмму NGINX с помощью команды helm install:

helm install example ./mychart --set service.type=NodePort

NAME: example

LAST DEPLOYED: Tue May 2 20:03:27 2017

NAMESPACE: default

STATUS: DEPLOYED

RESOURCES:

==> v1/Service

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

example-mychart 10.0.0.24 <nodes> 80:30630/TCP 0s

==> v1beta1/Deployment

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

example-mychart 1 1 1 0 0s

NOTES:

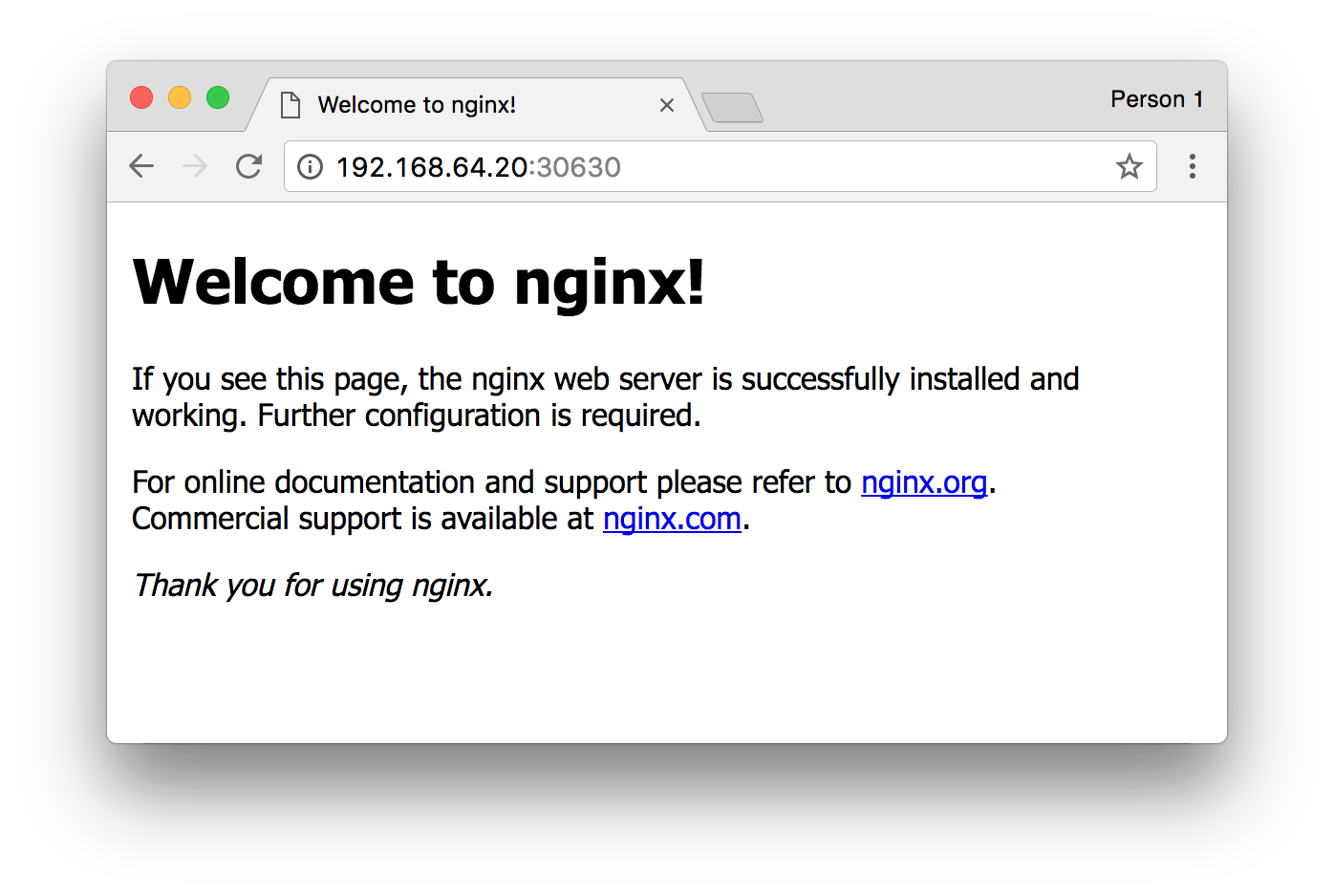
1. Get the application URL by running these commands:

export NODE\_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services example-mychart)

export NODE\_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")

echo http://$NODE\_IP:$NODE\_PORT/

Выходные данные helm install отображают удобную сводку состояния выпуска, какие объекты были созданы, и визуализированный файл NOTES.txt чтобы объяснить, что делать дальше. Выполните команды в выходных данных, чтобы получить URL-адрес для доступа к службе NGINX и получить его в браузере.



Если все прошло хорошо, вы должны увидеть страницу приветствия NGINX, как показано выше. Поздравляю! Вы только что развернули свою самую первую услугу, упакованную в виде диаграммы Helm!

Шаг 3. Изменение диаграммы для развертывания пользовательской службы

Созданная диаграмма создает объект Deployment, предназначенный для запуска образа, предоставляемого значениями по умолчанию. Это означает, что все, что нам нужно сделать для запуска другой службы, это изменить изображение, на которое указывает ссылка в values.yaml.

Мы собираемся обновить диаграмму, чтобы запустить [приложение списка дел,](https://github.com/prydonius/todomvc/tree/master/examples/react) доступное в [Docker Hub](https://hub.docker.com/r/prydonius/todo/). В *файле* values.yaml обновите ключи изображения, чтобы они ссылались на изображение списка дел:

image:

repository: prydonius/todo

tag: 1.0.0

pullPolicy: IfNotPresent

При разработке диаграммы рекомендуется запускать ее через линтер, чтобы убедиться, что вы следуете рекомендациям и что ваши шаблоны хорошо сформированы. Выполните команду helm lint, чтобы увидеть линтер в действии:

helm lint ./mychart

==> Linting ./mychart

[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, no failures

Линтер не обнаружил какие-либо серьезные проблемы с Chart, так что мы готовы к работе. Однако, в качестве примера, вот что может вывести линтер, если вам удалось что-то сделать не так:

echo "malformed" > mychart/values.yaml

helm lint ./mychart

==> Linting mychart

[INFO] Chart.yaml: icon is recommended

[ERROR] values.yaml: unable to parse YAML

error converting YAML to JSON: yaml: line 34: could not find expected ':'

Error: 1 chart(s) linted, 1 chart(s) failed

На этот раз линтер сообщает нам, что не удалось правильно проанализировать мой файл values.yaml. С подсказкой номера строки мы можем легко найти исправление ошибки, которую мы ввели.

Теперь, когда Chart снова действительна, запуститеhelm install еще раз, чтобы развернуть приложение списка дел:

helm install example2 ./mychart --set service.type=NodePort

NAME: example2

LAST DEPLOYED: Wed May 3 12:10:03 2017

NAMESPACE: default

STATUS: DEPLOYED

RESOURCES:

==> v1/Service

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

example2-mychart 10.0.0.78 <nodes> 80:31381/TCP 0s

==> apps/v1/Deployment

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

example2-mychart 1 1 1 0 0s

NOTES:

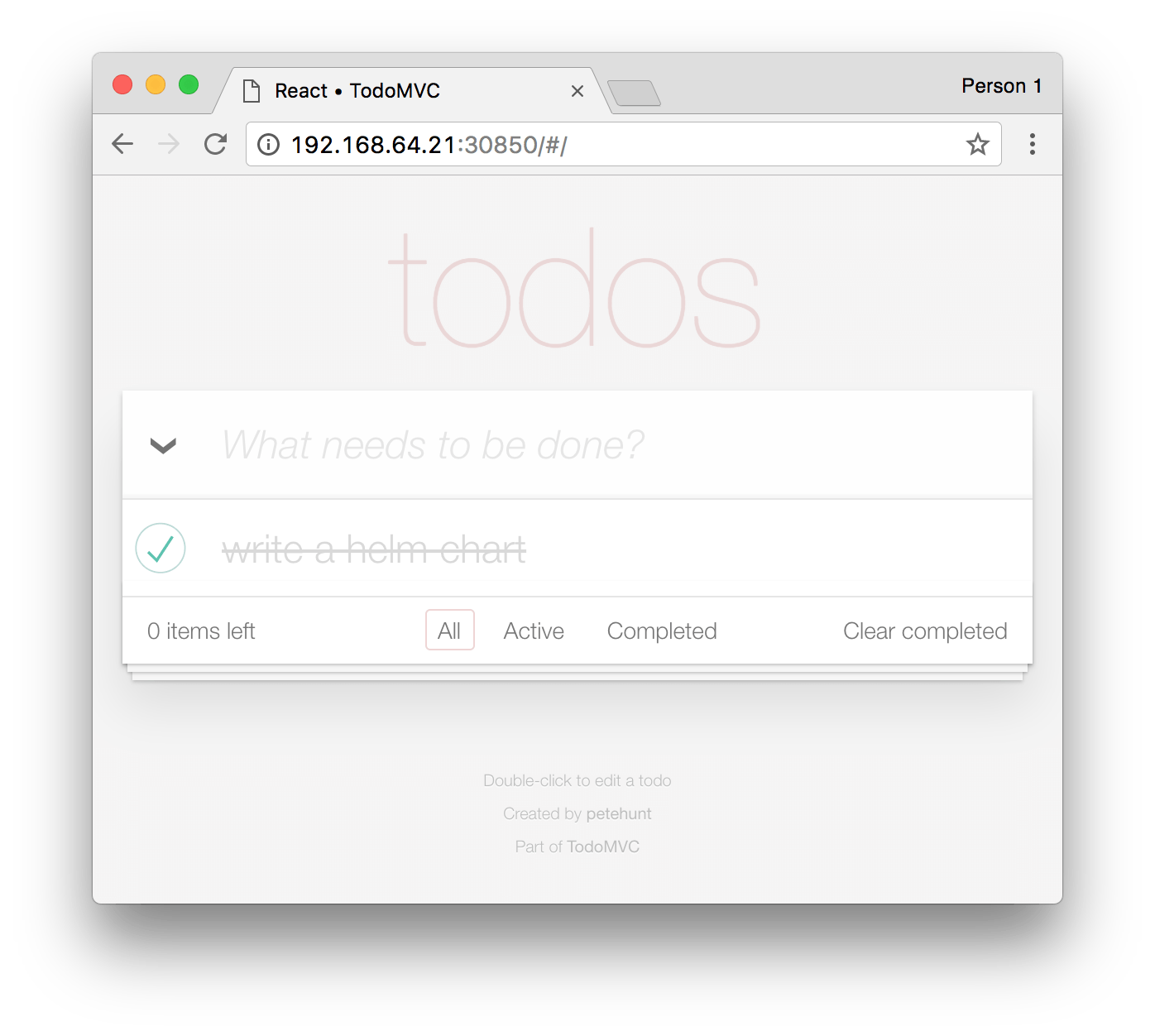
1. Get the application URL by running these commands:

export NODE\_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services example2-mychart)

export NODE\_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")

echo http://$NODE\_IP:$NODE\_PORT/

Еще раз, мы можем выполнить команды в NOTES, чтобы получить URL-адрес для доступа к нашему приложению.



Если контейнеры для приложений уже созданы, их можно запустить вместе с chart, обновив values по умолчанию или шаблон Deployment.

Шаг 4: Упакуйте все это, чтобы поделиться

До сих пор в этом учебнике мы использовали команду helm install для установки локальной распакованной диаграммы. Однако, если вы хотите поделиться своими диаграммами со своей командой или сообществом, ваши потребители обычно устанавливают диаграммы из пакета tar. Мы можем использовать helm package для создания пакета tar:

helm package ./mychart

Helm создаст пакет mychart-0.1.0.tgz в нашем рабочем каталоге, используя имя и версию из метаданных, определенных в файле Chart.yaml. Пользователь может выполнить установку из этого пакета вместо локального каталога, передав пакет в качестве параметра *для* helm install.

helm install example3 mychart-0.1.0.tgz --set service.type=NodePort

Репозитории

Чтобы упростить обмен пакетами, Helm имеет встроенную поддержку установки пакетов с HTTP-сервера. Helm считывает индекс репозитория, размещенный на сервере, который описывает, какие пакеты диаграмм доступны и где они расположены.

Мы можем использовать команду helm serve для запуска локального репозитория для обслуживания нашего chart.

helm serve

Regenerating index. This may take a moment.

Now serving you on 127.0.0.1:8879

Теперь в отдельном окне терминала вы сможете увидеть свой Chart в локальном репозитории и установить его оттуда:

helm search local

NAME VERSION DESCRIPTION

local/mychart 0.1.0 A Helm chart **for** Kubernetes

helm install example4 local/mychart --set service.type=NodePort

Чтобы настроить удаленный репозиторий, вы можете следовать руководству в [документации Helm](https://github.com/kubernetes/helm/blob/master/docs/chart_repository.md).

Зависимости

По мере того, как приложения, упакованные в виде диаграмм, становятся все более сложными, может оказаться, что вам нужно извлечь зависимость, такую как база данных. Helm позволяет указать sub-charts, которые будут созданы в рамках того же релиза. Чтобы определить зависимость, создайте файл requirements.yaml в корневом каталоге диаграммы:

cat > ./mychart/requirements.yaml <<EOF

dependencies:

- name: mariadb

version: 0.6.0

repository: https://charts.helm.sh/stable

EOF

Подобно файлу зависимостей языка среды выполнения (например*, требованиям Python.txt*), файл requirements.yaml позволяет управлять зависимостями Chart’s и их версиями. При обновлении зависимостей создается файл блокировки, чтобы последующая выборка зависимостей использовала известную рабочую версию. Выполните следующую команду, чтобы получить определенную нами зависимость MariaDB:

helm dep update ./mychart

Hang tight **while** we grab the latest from your chart repositories...

...Unable to get an update from the "local" chart repository (http://127.0.0.1:8879/charts):

Get http://127.0.0.1:8879/charts/index.yaml: dial tcp 127.0.0.1:8879: getsockopt: connection refused

...Successfully got an update from the "bitnami" chart repository

...Successfully got an update from the "incubator" chart repository

Update Complete. \*Happy Helming!\*

Saving 1 charts

Downloading mariadb from repo

$ ls ./mychart/charts

mariadb-0.6.0.tgz

Helm нашел подходящую версию в репозитории bitnami и извлек ее в каталог sub-chart моего chart. Теперь, когда мы установим chart, мы увидим, что объекты MariaDB также созданы:

helm install example5 ./mychart --set service.type=NodePort

NAME: example5

LAST DEPLOYED: Wed May 3 16:28:18 2017

NAMESPACE: default

STATUS: DEPLOYED

RESOURCES:

==> v1/Secret

NAME TYPE DATA AGE

example5-mariadb Opaque 2 1s

==> v1/ConfigMap

NAME DATA AGE

example5-mariadb 1 1s

==> v1/PersistentVolumeClaim

NAME STATUS VOLUME CAPACITY ACCESSMODES AGE

example5-mariadb Bound pvc-229f9ed6-3015-11e7-945a-66fc987ccf32 8Gi RWO 1s

==> v1/Service

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

example5-mychart 10.0.0.144 <nodes> 80:30896/TCP 1s

example5-mariadb 10.0.0.108 <none> 3306/TCP 1s

==> apps/v1/Deployment

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

example5-mariadb 1 1 1 0 1s

example5-mychart 1 1 1 0 1s

NOTES:

1. Get the application URL by running these commands:

export NODE\_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services example5-mychart)

export NODE\_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")

echo http://$NODE\_IP:$NODE\_PORT/