# 6. Объекты Kubernetes. YAML. Декларативное управление кластером

В первой главе про Kubernetes мы рассмотрели его основные составляющие, изучили термины и провели первое развертывание и масштабирование своего минимального сервиса, просто вызывая команду kubectl из командной строки. Это не потребовало от нас больших усилий, а результаты получились отличные, по большому счету достаточные для того, чтобы развернуть элементарные сетевые сервисы и приложения на кластере под управлением Kubernetes.

Для одного или двух простых сервисов вызов kubectl из командной строки еще может выполнять свою роль. Однако для приложения, составленного из множества микросервисов, потенциально нескольких десятков, запуск каждого сервиса с помощью командной строки, дополнительные настройки с помощью множества флагов, создание и координация открытых портов для взаимодействия сервисов снова через отдельные вызовы команды kubectl приведет к плохо поддерживаемой и склонной к запутанным ошибкам системе развертывания приложения как готовой системы.

Конечно, можно было бы создать единый скрипт развертывания системы целиком и запускать его на сервере, предназначенном исключительно для этой задачи. Отслеживать изменения в процессе запуска и развертывания проекта можно, храня исходный текст этого скрипта в системе контроля версий. Но скрипты для запуска сложных действий из командной строки имеют склонность становиться чрезмерно сложными и разрастаться до порядочных размеров. К тому же языки командных оболочек, таких как shell, не совсем предназначены для разветвленной логики, и в итоге их сложность становится серьезным препятствием к быстрому изменению.

Kubernetes не стал бы столь популярным без модели управления кластером с помощью объектов своего программного интерфейса, следующего стандарту REST (RESTful API). Желаемое состояние (desired state) кластера Kubernetes в каждый момент описывается набором объектов, являющихся частью программного интерфейса Kubernetes. У каждого из этих объектов есть набор полей, допустимых значений и вложенных объектов, однозначно описывающих, что именно требуется от управляющих компонентов кластера для того, чтобы текущее состояние кластера пришло к желаемому состоянию, описанному объектами. Объекты Kubernetes — это RESTful-ресурсы программного интерфейса Kubernetes, но, как мы увидим, работать с ним напрямую нам не придется, все сделает команда kubectl.

Когда мы создавали свое первое развертывание, вызывая команду kubectl create deployment, за кулисами для нас создали объект Kubernetes под названием Deployment (развертывание). После этого мы открыли порт для доступа и взаимодействия со своим приложением в контейнере (kubectl expose) — то есть создали точку доступа в сервис, и, нетрудно догадаться, это привело к созданию объекта Service.

Основной способ управления кластером Kubernetes — именно описание объектов его программного интерфейса и последующая работа с ними. Вместо запуска хотя и простых и понятных команд kubectl гораздо более гибким и легким для последующих изменений способом управления является прямое описание этих объектов. Для описания обычно используется язык YAML, хотя в итоге перед отправкой на управляющий сервер Kubernetes он преобразуется в JSON. Файлы формата YAML хранятся в системе контроля версий, зачастую рядом с исходным кодом и данными приложения, и изменение способа развертывания приложения в кластере Kubernetes легко обновлять и отслеживать. Объекты Kubernetes, описанные в YAML, часто называют *манифестами* Kubernetes (manifests).

## Несколько слов об «ужасах» YAML

Работа с YAML в Kubernetes стала практически притчей во языцех. Так как мощь и гибкость Kubernetes требуют создания немалого количества файлов YAML и работы с ними, разработчики часто саркастически замечают, что Kubernetes — это и есть редактирование YAML, не самого удобного для человека языка разметки. Как иронично подметил Келси Хайтауэр в своем твиттере (Kelsey Hightower, один из самых заметных проводников и пропагандистов идей Kubernetes компании Google): «Я был на конференции, которая была полностью посвящена только YAML. Это была конференция KubeCon (главная конференция Kubernetes)».

Все это так, и создавать объекты Kubernetes с нуля, мучительно вспоминая все поля и правила записи YAML, действительно непросто. Однако все гениальное просто, и, как правило, эта проблема элементарно решается шаблонами и помощью редактора. Особенно хороши в этом различные расширения и плагины для сред разработки, таких как IntelliJ и VS Code. Ну а в этой главе мы попробуем обойтись без дополнительных средств, используя шаблоны команды kubectl (да, они есть и там!). Список самых удобных на данный момент инструментов для работы с объектами Kubernetes и YAML вы сможете найти на сайте книги, ну и, конечно, простым поиском в Интернете.

## Объект Deployment вместо kubectl create deployment

Как мы уже упомянули, команда kubectl create deployment, примененная нами в предыдущей главе для развертывания и запуска контейнера в кластере Kubernetes, на самом деле создает объект Kubernetes Deployment и отправляет его на управляющий узел.

К нашей радости, все та же самая команда поможет нам получить этот объект в виде YAML. Используем мы ее с сочетанием пары флагов --dry-run=client -o yaml, что означает не что иное, как запуск в холостом режиме без отправки в кластер (client) и вывод созданного объекта Kubernetes на консоль. Формат вывода мы запрашиваем как раз YAML.

Итак, развернем сервис time-service из предыдущего раздела в холостом режиме, используя для простоты образ image из моего репозитория Docker Hub (ivanporty). Вы можете заменить репозиторий на свой собственный.

$ kubectl create deployment time-service --image=ivanporty/time-service:0.1.0 --dry-run=client -o yaml

Мы легко получаем описание объекта Deployment, которое команда kubectl create deployment отправила бы на сервер программного интерфейса Kubernetes apiserver в случае не холостого, а реального вызова:

# Версия программного интерфейса Kubernetes  
apiVersion: apps/v1  
# Тип объекта  
kind: Deployment  
# Метаданные нашего объекта, вложенный объект ObjectMeta  
metadata:  
 creationTimestamp: null  
 # список меток самого объекта Deployment  
 labels:  
 app: time-service  
 name: time-service  
# Описание собственно правил развертывания контейнера  
# Вложенный объект DeploymentSpec  
spec:  
 # количество запущенных отсеков pods для масштабирования   
 replicas: 1  
 selector:  
 matchLabels:  
 app: time-service  
 strategy: {}  
 # описание шаблона для создания новых отсеков  
 template:  
 metadata:  
 creationTimestamp: null  
 # список меток для нового отсека  
 labels:  
 app: time-service  
 # непосредственно описание контейнера в отсеке  
 spec:  
 containers:  
 - image: ivanporty/time-service  
 name: time-service:0.1.0  
 resources: {}  
status: {}

Сразу можно видеть, что информации объект Deployment предоставляет намного больше, чем мы указывали при запуске команды kubectl run/create deployment, но последняя многим полям присваивает разумные значения по умолчанию. Для простоты комментарии с описанием самых важных полей и вложенных объектов добавлены прямо в шаблон YAML.

Для начала мы просто хотим запустить наш контейнер в кластере, поэтому для нас самым важным является способ указать, где находится образ этого контейнера. Этим заведует шаблон, по которому создаются все отсеки и контейнеры внутри них — .template.spec. Еще мы видим загадочный набор меток (labels), повторяющийся три раза, — это уникальные «бирки» на объектах Kubernetes, позволяющие найти нужные объекты в сложном большом кластере. Пока же давайте считать, что для простоты они все должны совпадать.

Используя этот шаблон, создадим свой первый объект Kubernetes, оставив лишь нужные нам поля. Сохраним его в отдельном файле YAML (можно назвать его манифестом), поместим его в папку k8s и назовем k8s-time-service-deploy.yaml. Оставим ту же самую метку app, которая используется командой kubectl create deployment. Это будет просто описание приложения app[lication], довольно популярная метка для простых развертываний.

Вот что у нас получится за объект:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 labels:  
 app: time-service  
 name: time-service  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: time-service  
 template:  
 metadata:  
 labels:  
 app: time-service  
 spec:  
 containers:  
 - image: ivanporty/time-service:0.1.0  
 name: time-service

Остается только создать (create) свое развертывание с помощью объекта Kubernetes. Передать команде kubectl описание объекта из файла YAML нужно с помощью флага -f. (Внимание: если у вас еще остался запущенным сервис и развертывание time-service из предыдущего раздела, не забудьте их удалить с помощью команды kubectl delete!)

$ cd k8s  
$ kubectl create -f k8s-time-deploy.yaml

Как видите, теперь указывать тип объекта и его свойства, такие как образ контейнера, больше нет необходимости — все описано объектом в файле YAML. Убедиться в том, что развертывание создано, можно с помощью уже хорошо знакомых команд:

$ kubectl get deployments  
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE  
time-service 1 1 1 1 9m

В качестве простого упражнения на повтор материала прошлой главы посмотрите детали и события развертывания (describe, events), чтобы убедиться, что оно создано без ошибок.

Чем сложнее и более распределена по небольшим микросервисам будет ваша система и приложение в целом, тем больше преимуществ предоставит управление кластером с помощью объектов, а не вызовов команды kubect xyz ...:

* объекты хранятся в системе контроля версий рядом с исходным кодом системы. Вы видите, какие изменения были сделаны с состоянием кластера, просто по изменениям YAML и комментариям в Git;
* элементарно добавить новые метки, изменить количество экземпляров, передать контейнерам переменные окружения, все в одном описании Deployment, и, опять же, легко следить за этим в системе контроля версий;
* любой оператор кластера, включая разработчиков, сможет мгновенно восстановить состояние кластера в определенный момент, просто воссоздав объекты Kubernetes из системы контроля версий. Сделать это вызовами команды kubectl будет очень сложно и путанно.

## Объект Service вместо kubectl expose

При знакомстве с Kubernetes мы использовали команду kubectl expose для открытия сетевого порта и доступа к работающему контейнеру и этой же командой в холостом режиме сможем получить отличный шаблон для объекта сервиса Kubernetes Service:

$ kubectl expose deployment time-service --port=8080 --type=NodePort --dry-run=client -o yaml

Вот что у нас получится в результате. Структура объекта будет очень похожа на развертывание Deployment, и для краткости здесь удалены лишние поля, а метки совпадают с теми, что были использованы в шаблоне развертывания (app):

apiVersion: v1  
kind: Service  
metadata:  
 labels:  
 app: time-service  
 name: time-service  
spec:  
 # список портов. Дополнительно можно указать протокол  
 ports:  
 - port: 8080  
 # по этим меткам идет поиск отсеков, куда отправляются запросы  
 selector:  
 app: time-service  
 # тип сервиса. В облаке можно использовать LoadBalancer  
 type: NodePort

Базовое описание полей сервиса есть прямо в файле YAML, ну а полностью мы рассмотрим все детали и возможности сервисов Kubernetes в отдельной главе.

Назовем файл по аналогии с развертыванием, намекнув в его названии, что он содержит описание сервиса: k8s-time-svc.yaml. Создадим новый сервис (если у вас запущен предыдущий вариант сервиса, лучше будет его удалить):

$ kubectl create -f k8s-time-svc.yaml

И уже знакомыми командами узнаем, что сервис успешно создан, и увидим порт внутри кластера, по которому к нему можно обратиться:

$ kubectl get services  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 17d  
time-service NodePort 10.98.92.168 <none> 8080:30721/TCP 5s

В прошлом разделе мы посмотрели, как в локальных кластерах Minikube или Docker обратиться к сервисам типа NodePort, это будет хорошее повторение материала.

Если вы пользуетесь публичным полноценным кластером, таким как Google Cloud Kubernetes Engine, вы сможете получить доступ к сервису из Интернета, поменяв в описании объекта тип сервиса на LoadBalancer, а затем получив его открытый IP-адрес из описания сервиса, полученного такой же командой kubectl get services.

## Императивное управление кластером Kubernetes — create и delete

После описания объектов Kubernetes в формате YAML остается их создать (kubectl create -f), а потом удалить (kubectl delete -f). Подобная работа с объектами в программировании называется императивной (imperative), или процедурной — мы напрямую создаем объекты и управляем их жизненным циклом в «приказном порядке», при необходимости обновляем, в конце концов удаляем и затем создаем заново с обновленными настройками. Именно так мы пока и поступали с созданными нами развертыванием и сервисом для своего приложения time-service. Система лишь исполняет наши инструкции и в управлении непосредственно объектами не участвует — она лишь использует их для определения, каким должно быть «желаемое состояние» (desired state) нашего кластера.

Интересно (и удобно), что инструмент kubectl поддерживает создание и удаление объектов не только из одного файла (а в одном файле может содержаться множество объектов разного типа, например развертывание и сервисы вместе), а из множества файлов YAML, находящихся в директории. Учитывая, что мы поместили наши декларации Kubernetes (два файла с объектами развертывания и сервиса) в директорию k8s, создать и удалить все необходимое для микросервиса time-service можно и так:

$ kubectl delete -f k8s/  
deployment.apps "time-service" deleted  
service "time-service" deleted  
  
$ kubectl create -f k8s/  
deployment.apps "time-service" created  
service "time-service" created

Если после предыдущих примеров развертывание и сервис уже существуют, сначала удалите их, а затем попробуйте эти команды. Для множества микросервисов вероятно появление и множества деклараций с объектами кластера Kubernetes, и такая команда становится весьма кстати, если вы развертываете все сервисы разом.

Что же дальше? Управлять созданием и удалением объектов кластера и менять таким образом его желаемое состояние понятно и прямолинейно. Однако дополнительные преимущества Kubernetes использовать будет непросто — например, что, если вы хотите создать автоматически масштабируемое приложение? Или хотите обновить контейнеры только на части отсеков (pods), или меняете версию контейнера только для одного отсека? Удалять и создавать подмножества объектов отдельными командами будет неудобно, так же как очень тяжело отследить историю внесенных изменений. Для больших кластеров, сложных приложений и сервисов почти всегда применяется *декларативное* управление.

## Декларативное управление кластером. Команда apply

Декларативное описание объектов Kubernetes не требует от нас прямого вмешательства в дела кластера и оставляет управление объектами, включая их создание и удаление, в руках системы управления кластером. Девиз декларативного описания — неважно, в каком порядке, в течение какого времени и как часто система получает описание объектов кластера. Управление кластера обязуется преобразовать простое, понятное и легко отслеживаемое описание объекта в *желаемое состояние кластера* (desired state).

Если мы передали системе объект с развертыванием Deployment и такого развертывания в кластере еще нет, он будет создан, как если бы была вызвана команда create. Если мы передали обновленный сервис Service, в котором, например, добавили еще один порт, система динамически обновит соответствующий объект (или же удалит и создаст заново) и приведет кластер в его желаемое состояние.

Получить описание объектов и сделать на их основе необходимые действия позволяет команда kubectl apply. Учитывая, что основы объектов развертывания и сервисов мы уже знаем и создали файлы YAML с их описанием, передача объектов в управление кластера теперь происходит в одну строчку:

$ kubectl apply -f k8s/  
deployment.apps "time-service" created  
service "time-service" created

Как мы видим, при отсутствии объектов в системе они будут созданы. Если бы они там уже существовали, изменений в системе не было бы.

Предположим, что мы получили первых пользователей для микросервиса и одного экземпляра сервиса нам мало, а использовать автоматическое масштабирование мы пока не хотим. Отредактируем объект развертывания и укажем, что нам требуется два экземпляра (replicas) отсека и контейнера в нем при развертывании:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 labels:  
 app: time-service  
 name: time-service  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: time-service  
 template:  
 metadata:  
 labels:  
 app: time-service  
 spec:  
 containers:  
 - image: ivanporty/time-service  
 name: time-service

Декларативное описание позволит не волноваться о том, удалено ли старое развертывание, и не проверять, в скольких экземплярах оно находится сейчас (а может быть, оно вообще еще не создано). Передадим объекты в управление кластера и получим свое желаемое состояние без дополнительных усилий:

$ kubectl apply -f k8s/  
deployment.apps "time-service" configured  
service "time-service" unchanged

Как мы видим, изменений в сервисе не было, а вот конфигурация развертывания была изменена — причем команда apply описывает, что именно было сделано с объектами: создание, удаление или переконфигурация.

Проверим, в каком состоянии и в скольких экземплярах работает развертывание:

$ kubectl get deploy  
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE  
time-service 2 2 2 2 2m

Состояние кластера верно. Декларативное описание развертываний и сервисов с помощью объектов Kubernetes помогло нам достичь своего желаемого состояния без дополнительных проверок командой get, создания и удаления объектов, масштабирования scale и множества других «ручных» команд.

Если мы управляем кластером исключительно декларативным способом, то в случае отсутствия ошибок всегда можем быть уверены в том, что его состояние совпадает с тем, что описано нашими объектами Kubernetes. Управление сложными развертываниями, их обновлениями и масштабированием становится намного проще и понятнее.

## Резюме

* Простой запуск контейнеров, открытие портов и масштабирование в Kubernetes возможны просто с использованием команды kubectl. На этой основе можно создать простые скрипты для управления кластером. Однако это императивное управление, зависящее от предыдущего и текущего состояния кластера, и поэтому оно быстро становится сложным и запутанным.
* Программный интерфейс Kubernetes API, доступный через apiserver, полностью следует концепции REST. Все концепции и возможности Kubernetes описываются как RESTful-ресурсы, называемые *объектами Kubernetes*. Обычно их описывают в формате YAML и отслеживают в системе контроля версий.
* Развернуть и запустить контейнер в кластере Kubernetes и открыть его порты позволяют объекты Deployment и Service. Их можно напрямую (императивно) создавать и удалять командами create и delete.
* Самым удобным и максимально уменьшающим ошибки способом управления состоянием кластера является *декларативный*. Мы передаем управляющей системе кластера набор объектов командой kubectl apply и говорим, что хотим видеть кластер соответствующим переданным объектам (желаемое состояние). Управляющий цикл (control loop) Kubernetes постоянно проверяет состояние объектов и приводит кластер к желаемому состоянию.
* Декларативный способ значительно уменьшает сложность администрирования и поддержки серверной системы. Состояние сложной распределенной системы теперь описано простыми объектами YAML, легко воспроизводится и имеет полную историю изменений.