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

Стратегии развертывания в Kubernetes

В этой статье, мы узнаем, что такое статегии развертывания, во время установки контейнеров используя систему открестрации контейнеров Kubernetes. В конце этой статьи, мы будем знать различные пути установки в кластере Kubernetes.

Обзорное введение в Kubernetes
С популярностью контейнеризации и с революцией в создании, доставке и обслуживании приложений появилась необходимость эффективного обслуживания этих контейнеров. Множество систем оркестраторации контейнеров введены для обслуживания цикла этих контейнеров в больших системах.

Kubernetes один из подобных инструментов оркестрации, который берет на себя заботу о предоставлении и установке, выделении ресурсов, балансировке нагрузки, обнаружении сервисов, предоставлении высокой доступности и другие важные моменты любой системы. С его помощью мы можем разложить наши приложения в на маленькие системы(называемые микросервисами) во время установки. затем мы можем собрать(или оркестрировать) эти системы вместе во время установки.

Применение облачного подхода увеличивает разработку приложений основанных на микросервисной архитектуре. Для таких приложений, наибольший из вызовов это встреча с установкой. Иметь хорошую стратегию - необходимость. В Kubernetes, есть множество путей для выпуска приложений, необходимо выбрать правильную стратегию, чтобы сделать инфраструктуру надежной во время развертывания приложений или обновления. Для примера, в производственной среде, всегда есть требование чтобы пользователи не ощутили время простоя сервиса. В оркестраторе Kubernetes, правильная стратегия - убедиться в верности управления различных версий образов контейнера. В общем, эта статья покрывает различные статегии установки в Kubernetes.

Требования

Чтобы продолжить, нам необходим опыт с кубернетес. Если вы не знакомы с этой платфорой, Пройдите для начала "Step by Step Introduction to Basic Kubernetes Concepts" инструкцию. В ней вы можете найти, всё что нужно для того, чтобы понять, чтоп роисходит в этой иснтрукции. мы также рекомендуем пройти полистать Kubernetes документацию если или когда потребуется.

Кроме того, если нам потребуется kubectl, инструмент командной строки, который позволит нам управлять кластером Kubernetes из терминала. Если у вас нет этого инструмента, проверьт инструкцию по установке kubectl. Так же потребуется базовое понимание Linux и YAML.

Что такое развертывание в Kubernetes?

Развертывание это объект в kubernetes, который определяет желаемое состояние для нашей программы. Развертывание объявляема, это значит, что мы не должны говорить как достигнуть состояния. Вместо этого, мы объявляем желаеме состояние, и позволяем автоматически достигнуть конечного результата наилучшим путём. Развертывание позволяет нам описать жизненный цыкл приложения, к примеру: какой образ использовать для приложения, количество подов, которое необходимо, и способ которым они должны обновляться.

Преимущества использования Kubernetes развертывания.

Процесс ручного обновления контейнеризированных приложением может занимать много времени и быть скучным. Развертывание Kubernetes делает этот процесс автоматическим и повторяемым. Развертывание полностью управляемое с помощью Kubernetes, и полный процесс производится на стороне сервера без взаимодействия клиента.

Более того, контроллер развертывания Kubernetes всегда мониторит здоровье подов и нод. Он заменяет упавшие поды или пропускает недоступные ноды, убеждаясь в непрерывности критических приложений.

Стратегии развертывания

Последовательное развертывание обновлений

Последовательное развертывание это стандартная развертываемае стратегия для Kubernetes. Он заменяет по одному поды прошлых версий на новые версии, при этом создавая ситуацию в которой не возникает времени простоя. Последовательное развертывание медленно заменяет объекты прошлых верси приложения на новые.

Используя стратегии развертывания, есть две различные возможности, которые позволяют настроить процесс обновления:

Максимальный Всплеск(`maxSurge`): количество подов, которое может быть создано над желаемым количеством подов во время обновления. Это может быть абсолютное число в процентах от количества реплик. По умолчанию это 25%.

Максимальная недоступность(`maxUnavailable`): количество подов которое может быть недоступно во время процесса обновления. Это может быть обсолютное количество в процентах от количества реплик, по умолчанию 25%.

Первое мы создадим шаблон нашего последовательного развертывания. В шаблон ниже, мы указали `maxSurge` = 2 и `maxUnavailable` = 1.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rollingupdate-strategy
  version: nanoserver-1709
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2
      maxUnavailable: 1
  selector:
    matchLabels:
      app: web-app-rollingupdate-strategy
      version: nanoserver-1709
  replicas: 3
  template:
    metadata:
      labels:
        app: web-app-rollingupdate-strategy
        version: nanoserver-1709
    spec:
      containers:
        - name: web-app-rollingupdate-strategy
          image: hello-world:nanoserver-1709
```

Мы можем создать развертывание используя `kubectl` команду.

kubectl apply -f rollingupdate.yaml

Как только мы получили шаблон развертывания, мы можем предоставить путь для доступа объекта развертывания с помощью создания сервиса. Отметим, что мы развертывая образ `hello-world` с помощью nanoserver-1709. В этом случае мы имеем два заглавия, `name= web-app-rollingupdate-strategy` и `version=nanoserver-1709`. Мы укажем их как названия для сервисов ниже. и сохраним файл `service.yml`

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: web-app-rollingupdate-strategy
  labels:
    name: web-app-rollingupdate-strategy
    version: nanoserver-1709
spec:
  ports:
    - name: http
      port: 80
      targetPort: 80
  selector:
    name: web-app-rollingupdate-strategy
    version: nanoserver-1709
  type: LoadBalancer
```

Создание сервиса добавить балансировщик нагрузки который будет доступен снаружи кластера.

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

Чтобы проверить наличие деплойментов запускаем команду:

```
$ kubectl get deployments
```

Если деплоймент всё еще создается то будет будет следующий ответ:

```
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
rollingupdate-strategy   0/3     0            0           1s
```

Если еще раз запустить команду `kubectl get deployments` чуть позже. Вывод будет выглядить следующийм образом:

```
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
rollingupdate-strategy   3/3     0            0           7s
```

Чтобы увидеть количество реплик созданых деплойментом, запустите:

```
$ kubectl get rs
```

Ответ будет выглядеть следующим образом:

```
NAME                                    DESIRED   CURRENT   READY   AGE
rollingupdate-strategy-87875f5897   3         3         3       18s
```

Чтобы увидеть 3 пода запущенных для деплоймента запустите:

```
$ kubectl get pods
```

Созданные ReplicaSet(набор реплик)проверять что запущенно 3 рабочих пода. А вывод будет следующим:

```
NAME                                    READY     STATUS    RESTARTS   AGE
rollingupdate-strategy-87875f5897-55i7o   1/1       Running   0          12s
rollingupdate-strategy-87875f5897-abszs   1/1       Running   0          12s
rollingupdate-strategy-87875f5897-qazrt   1/1       Running   0          12s
```

Давайте обновим `rollingupdate.yaml` шаблон деплоймента чтобы использовать образ `hello-world:nanoserver-1809` вместо образа `hello-world:nanoserver-1709`. Затем обновим образ существующего запущенного деплоймента используя команду `kubectl`.

```
$ kubectl set image deployment/rollingupdate-strategy web-app-rollingupdate-strategy=hello-world:nanoserver-1809 --record
```

Вывод будет похожим на:

```
deployment.apps/rollingupdate-strategy image updated
```

Теперь мы развертывает образ `hello-world` с версией `nanoserver-1809`. В данном случае мы обновили `lable` в `service.yaml`. `label` будет обновлет на `version=nanoserver-1809`. Еще раз запускаем команду ниже, для обновления сервиса который подберет новый рабочий под, с новой версией образа.

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

Чтобы увидеть статус выкатывания запустите команду ниже:

```
$ kubectl rollout status deployment/rollingupdate-strategy

Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
```

Запустим еще раз чтобы убедиться что обновление прошло успешно:

```
$ kubectl rollout status deployment/rollingupdate-strategy

deployment "rollingupdate-strategy" successfully rolled out
```

После успешного обновления, мы можем посмотреть на деплоймент командой `kubectl get deployments`:

```
$ kubectl get deployments

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
rollingupdate-strategy   3/3     0            0           7s
```

Выполните `kubectl get rs`, чтобы увидеть что Deployment обновился. Новые поды созданы в новом ReplicaSet и запущено 3 копии. Старый ReplicaSet больше не содержит рабочих копий.

```
$ kubectl get rs

NAME                                   DESIRED   CURRENT   READY   AGE
rollingupdate-strategy-87875f5897   3         3         3       55s
rollingupdate-strategy-89999f7895   0         0         0       12s
```

Запустите `kubectl get pods` теперь должны быть только новые поды из новой ReplicaSet.

```
$ kubectl get pods

NAME                                    READY     STATUS    RESTARTS   AGE
rollingupdate-strategy-89999f7895-55i7o   1/1       Running   0          12s
rollingupdate-strategy-89999f7895-abszs   1/1       Running   0          12s
rollingupdate-strategy-89999f7895-qazrt   1/1       Running   0          12s
```

Очень полезна команда `rollout` в данномо случае. Мы можем использовать её чтобы проверить что делает наш deployment. Команда, по-умолчанию, ждет то тех пор пока deplyment не запустит успешно все поды. Когда deployment успешно отработает, команда

вернет 0 код в качестве указателя на успех. Если deployment упадет, команда завершиться с ненулевым кодом.

```
$ kubectl rollout status deployment rollingupdate-strategy

Waiting for deployment "rollingupdate-strategy" rollout to finish: 0 of 3 updated replicas are available…
Waiting for deployment "rollingupdate-strategy" rollout to finish: 1 of 3 updated replicas are available…
Waiting for deployment "rollingupdate-strategy" rollout to finish: 2 of 3 updated replicas are available…

deployment "rollingupdate-strategy" successfully rolled out
```

Если деплоймент упадет в Kubernetes, процесс deployment остановится, но поды из упавшего deployment остаются. При падении deployment, наше окружение может содержать подыд из двух старых и новых deploymentов. Чтобы вернуться в стабильное, рабочее состояние, мы можем использовать `rollout undo` команду, чтобы вернуть назад рабочие поды и очистить упавший деплоймент.

```
$ kubectl rollout undo deployment rollingupdate-strategy

deployment.extensions/rollingupdate-strategy
```

Затем проверяем статус deployment еще раз.

```
$ kubectl rollout status deployment rollingupdate-strategy

deployment "rollingupdate-strategy" successfully rolled out
```

Чтобы указать Kubernetes что приложение готово, нам необходима помощь от приложения. Kubernetes использует проверку готовности для того чтобы знать, что делает приложение. Как только экземпляр начинает отвечать на проверку готовности позитивно, экземпляр считается готовым для использования. Проверка готовности говорит Kubernetes когда приложение готово, но не когда приложение будет всегда готово. Если приложение продалжает падат, оно сможет никогда не ответить позитивно Kubernetes.

`Rolling deployment` обычно ждет когда новые поды будут готовы через проверку готовности прежде чем опустить старые компоненты. Если возникла существенная проблема, `rolling deployment` может быть отменен. Если возникает проблема, выкатывание обновлений или развертывания может быть прервано без отключения всего кластера.

Развертывание восстановления

При развертывании восстановления, мы полностью отключаем текущее приложение прежде чем выкатываем новое. На картинке ниже, версия 1 отображает текущую версию приложения, а 2 отражает версию нового приложения. Когда обновление текущей версии приложения, сначала убираем существующие рабочие копии версии 1, затем одновременно развертываем копии с новой версией.

Шаблон ниже показывает развертывание используя стратегию восстановления: Сначала, создаем наш деплоймент и помещаем его в файл `recreate.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: recreate-strategy
spec:
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: web-app-recreate-strategy
      version: nanoserver-1809
  replicas: 3
  template:
    metadata:
      labels:
        app: web-app-recreate-strategy
    spec:
      containers:
        - name: web-app-recreate-strategy
          image: hello-world:nanoserver-1809
```

Затем мы можем создать развертывание используя команду `kubectl`

```
$ kubectl apply -f recreate.yaml
```

Как только у нас будет шаблон развертывания мы можем предоставить способ для доступа в экземпляры развертывания создавая Service. Отметим, что развертывание образа `hello-world` с версией `nanoserver-1809`. В этом случае мы можем иметь два заголовка: `name= web-app-recreate-strategy` и `version=nanoserver-1809`. Мы назначим этим загловки для сервиса ниже и сохраним в service.yml.

```
apiVersion: v1
kind: Service
metadata:
  name: web-app-recreate-strategy
  labels:
    name: web-app-recreate-strategy
    version: nanoserver-1809
spec:
  ports:
    - name: http
      port: 80
      targetPort: 80
  selector:
    name: web-app-recreate-strategy
    version: nanoserver-1809
  type: LoadBalancer
```

Теперь создание этого сервиса создаст балансировщик нагрузки, который доступен вне кластера.

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

Метод создания требует некоторое вермя во время процесса обновления. Время простоя не проблема, если приложение может использовать окно обслуживания или сбой. Однако, если работа приложения критична и имеет высокий уровень SLA и требования доступности, использование различных стратегий развертывания будет правильным применением. Развертывание восстановлния в общем используется для целей разработки, так как легко

настраивается и прилоежние полностью обновляется на новую версию. Еще, то что нам не нужно обслуживать больше чем одну версию приложения в паралели, и поэтому мы можем избежать проблем обратной совестимости для данных и приложения.

Blue-Green Развертывание

В blue/green стратегия развертывания(иногда называемая red/black), `blue` - отражает текущую версию приложения, а `green` - новую версию приложения. Тут, только 1 версия живет. Траффик марштутизируется в `blue` развертвание пока `green` развертывание создается и тестируется. После конца тестирования, направляем траффик на новую версию.

После успешного развертывания, мы можем как сохранить `blue` развертывание для возможного отката или вывода из эксплуатации. Другая возножность, это развертывание новой версии приложения на этих экземплярах. В этом случае текущая `blue` окружение обслуживается как подгтотовительное для следующего релиза.

Эта техника может устранить время просто которое мы встретили в развертывании восстановления. Так же, `bluegreen` развертывание сокращает риск: если что-то необычное случиться с нашей `green` версией, мы тут же сможем откатиться на прошлую версию просто переключившись на `blue` версию. Есть постоянная возможность выкатитьоткатиться. Мы так же можем избежать проблем с версией, состояние приложения меняется одним развертыванием.

`BlueGreen` развертывание очень дорого, так как требует двойные ресурсы. Полноценное тестирование всей платформы должно быть выполнено до релиза его в производствао. Даже больше, обслуживание незименяемого приложения сложно.

Для начала мы создадим наше `blue` развертывание сохранив `blue.yaml` file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blue-deployment
spec:
  selector:
```

```
    matchLabels:
      app: blue-deployment
      version: nanoserver-1709
  replicas: 3
  template:
    metadata:
      labels:
        app: blue-deployment
        version: nanoserver-1709
    spec:
      containers:
        - name: blue-deployment
          image: hello-world:nanoserver-1709
```

Мы затем создадим развертывание используя `kubectl` команду

```
$ kubectl apply -f blue.yaml
```

Как только у нас есть шаблон развертывания, мы можем предоставить способ к доступу экземпляра развертывания создав сервис. Отметим, что наше развертывание образа `hello-world` с версией `nanoserver-1809`. В этом случае у нас есть два загловка `name= blue-deployment` и `version=nanoserver-1709`. Мы укажем этим загловки в селекторе сервиса и сохраним в файл `service.yaml`.

```
apiVersion: v1
kind: Service
metadata:
  name: blue-green-service
  labels:
    name: blue-deployment
    version: nanoserver-1709
spec:
  ports:
    - name: http
      port: 80
      targetPort: 80
  selector:
    name: blue-deployment
    version: nanoserver-1709
```

```
  type: LoadBalancer
```

Теперь создание сервиса, создаст балансировщик нагрузки который доступен вне кластера.

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

Наши настроки готовы.

Для `green` развертывания мы развернем новое развертывание рядом с `blue` развертыванием. Шаблон ниже содержит код ниже, сохраним как `green.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: green-deployment
spec:
  selector:
    matchLabels:
      app: green-deployment
      version: nanoserver-1809
  replicas: 3
  template:
    metadata:
      labels:
        app: green-deployment
        version: nanoserver-1809
    spec:
      containers:
        - name: green-deployment
          image: hello-world:nanoserver-1809
```

Заметим что образ hello-world:nanoserver-1809 изменен. Что значит, что мы сделали отдельное развертывание с двумя загловками, `name=green-deployment` и `version=nanoserver-1809`.

```
$ kubectl apply -f green.yaml
```

Обрезать `green` развертывание, мы обновим селектор для существующего сервиса. Изменим `service.yaml` и заменим селектор версии на вторую и назовем `green-deployment`. Это будет

совпадат с подами `green` развертывания.

```
apiVersion: v1
kind: Service
metadata:
  name: blue-green-service
  labels:
    name: green-deployment
    version: nanoserver-1809
spec:
  ports:
    - name: http
      port: 80
      targetPort: 80
  selector:
    name: green-deployment
    version: nanoserver-1809
  type: LoadBalancer
```

Мы создадим сервис еще раз используя команду `kubectl`:

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

Завершая, можно оценить что `bluegreen` развертывание есть всё или ничего. В отличии от развертывания выкатывания обновления, где мы не можем постеменно выктить нашу новую версию. Все пользователи получать обновления в одно и то же время, так же существующим версиям будет позволенно завершить их работу в старом экземпляре. Оно так же требует использования больше серверных ресурсов так как нам нужно запустить две копии каждого пода.

К счастью процедура отката проста: Нам просто нужно перевести выключатель обратно, и предыдущая версия будет возвращена. Это потому, что старая версия все еще крутиться на старых подах. Просто трафик не переводится на них. Когда мы убеждаемся что новая версия полноценно работает, мы должны удалить старые поды.

Канареечное развертывание

Стратегия канареечного обновления - частичный процесс который позвляет нам тестировать новые версии программ на реальных пользвоателях без обазтельного полного выкатывания. Похоже на `bluegreen` развертывания, но более подконтрольно, и они используют более прогрессивную доставку где развертывание применяется пофазово. Есть множество стратегий которые падают под зонтик канареечного развертывания, включая черновой запуск или AB тестирование

В канареечном развертывании новые версии приложения пошагово развертываются в кластер Kubernetes пока не достигнут небольшого количества живого трафика(небольшое количество живых пользователей подключаеются к новой версии пока остаток пользователей всё ещё используют старую версию)ю В этом применении, у нас есть два почти одинаковых сервера: один который используется всем текущим активным пользователям и другой - с новой версией который расчитан на небольшое количество пользователей. Когда становится понятно, что проблем с новой версией нет новая версия пошагово выкатывается на оставшуюся инфраструктуру. В конце, весь живой трафик идет на новую (канареечную) версию и затем становится основной версией производства.

Картинка ниже отражает самый прямой и простой путь произвести канарейчное развертывание. Новая версия развертывает небольшое количество серверов.

Пока это происходит, мы смотрим как обновленные машины работаеют. Мы проверяем ошибки и проблемы производительность, слушаем пользовательские отзывы. С тем как растет увереность в релизе, мы продолжаем устанавливать на оставшихся машинах до тех пор пока они все не обновятся на последний релиз.

Мы должны обратить вниманние на различные вещи когда планируем канареечное развертывание.

- Шаги: Как много пользователей будут переведены на канарейку в начале и как много шагов.
- Длительность: как долго мы планируем запускать канарейку? Канареечный релиз отличаются, так как мы должны дождаться достаточное количество клиентов для обновления, прежде чем мы сможем оценить результаты. Это может происходить в течении нескольких дней и даже недель.
- Метрики: какие местри небходимо записывать для анализа, включая производительность приложения и отчеты ошибок? Хорошо выбранные параметры

ведут к успеху канареечного развертывания. Для примера, очень простой способ измерить развертывание с помощью статус кодов HTTP. Мы можем иметь простой ping сервис который возвращает 200 когда развертывание успешно. Он вернет сервер пятисотую ошибку если возникнет проблема с развертыванием.

- Оценка: какие критерии мы будем использовать для определения успешности канареичного релиза.
  Канарейка используется в сценарии где мы должны тестировать новый функционал, обычно на бэкенде нашего приложения. Канареечное развертывание должно быть использованно когда мы на 100% не уверенны в новом приложении, мы предсказываем возможные проблемы с маленьким процентом падения. Эта стратегия обычно используется когда у нас идет мажорное обновление, такое, как добавление нового функционала или экспериментального функционала.

Резюме K8s стратегий развертывания

В конце можно сказать, если несколько способов развертывания приложения, когда работаем с devstage окружение, развертывание восстановления или ускоренное, обычно хороший выбор. Когда нужно вылить все на производство, то ускоренное или `bluegreen` развертывание хороший выбор, но тестирование обазятельно в этом случае. Если мы не уверены в стабильности платформы которая может влиять на выпуск новой версии ПО, тогда канареечный релиз должен быть тем самым путём.

Revision #1
Created 2022-10-13 14:04:31 UTC by gasick
Updated 2023-04-16 19:36:18 UTC by gasick