9.3 애플리케이션을 선언적으로 업데이트하기 위한 디플로이먼트 사용하기

생성일	@2021년 5월 2일 오전 12:04
태그	

디플로이먼트 : 로우 레벨로 간주되는 rc/rs를 통해 배포를 수행하는 대신, 앱을 선언적으로 업데이트하기 위한 높은 수준의 리소스

- 생성시 replica set 리소스가 그 아래 생성됨
- replica set은 차세대 rc임
  - 얘도 파드 복제하고 관리
- deployment사용하는 경우 파드는 deployment가 아닌 deployment의 replica set에 의해 생성되고 관리됨

[이렇게 rs위에 deployment를 만든 이유]

- 앱을 업데이트할 때는 추가 rc를 도입하고 두 컨트롤러가 잘 조화되도록 조정해야함
- 전체적으로 통제하는게 필요
- 디플로이먼트 리소스를 통해 원하는 리소스를 정의 → k8s가 나머지를 처리

1. 디플로이먼트 생성

rc와 똑같이 생성하면된다

[디플로이먼트 구성]

- label selector
- 원하는 레플리카 수
- 파드 템플릿
- 디플로이먼트 전략 지정 필드 : 업데이트 수행방법 정의

디플로이먼트 매니페스트 생성

(kubia-deployment-v1.yaml 참고)

기존 rc 예제에서 변경된 부분

- apiVersion
- kind
- name에 버전 포함할 필요 없음 (버전 삭제)

디플로이먼트 리소스 생성

디플로이먼트 만들기 전에 실행 중인 rc와 파드 삭제하고 kubia서비스만 남김

```
kubectl delete rc -all
kubectl create -f kubia-deployment-v1.yaml --record
```

디플로이먼트 롤아웃 상태 출력

디플로이먼트 상태 확인

```
kubectl rollout status deployment kubia
```

디플로이먼트가 레플리카셋을 생성하는 방법과 레플리카셋이 파드를 생성하는 방식 이해

디플로이먼트에서 생성한 파드 세개의 이름 중간에 숫자 값이 추가로 포함되어있음

→ 파드 템플릿의 해시값

→ replica set이 이러한 파드를 관리함을 뜻함

```
kubectl get replicasets
```

위 명령 수행시 레플리카셋의 이름에도 해당 파드 템플릿의 해시값이 포함된것을 확인 가능

- 디플로이먼트는 파드 템플릿의 각 버전마다 하나씩 여러개의 replica set을 만듦

- 파드 템플릿 해시값을 사용하면 디플로이먼트에서 지정된 버전의 파드 템플릿에 관해 항상 동일한 rs 사용가능

서비스로 파드 액세스

이 레플리카셋에 의해 생성된 파드 레플리카 3개 실앻중

→ 새 파드의 레이블이 서비스의 레이블 셀렉터와 일치 → 이전에 생성한 서비스를 사용해 액세스 가능

2. 디플로이먼트 업데이트

디플로이먼트 리소스에 정의된 파드 템플릿을 수정하기만 하면...

→ k8s가 실제 시스템 상태를 리소스에 정의된 상태로 만드는 데 필요한 모든 단계를 수행

사용 가능한 디플로이먼트 전략

이 새로운 상태를 달성하는 방법 : 디플로이먼트에 구성된 디플로이먼트 전략으로 결정됨

- 기본 = RollingUpdate
- 대안 = Recreate (9.1.1 방법과 같음)

데모 목적으로 롤링 업데이트 속도 느리게 하기

업데이트 프로세스를 약간 느려지게 해서 실제 업데이트가 롤링 방식으로 수행되고 있음을 확인할 수 있음

minReadySeconds 속성만 수정하면 됨

```
kubectl patch deployment kubia -p '{"spec": {"minReadySeconds": 10}}'
```

→ 파드 템플릿을 변경한것이 아니기 때문에 파드가 업데이트 되지는 않음

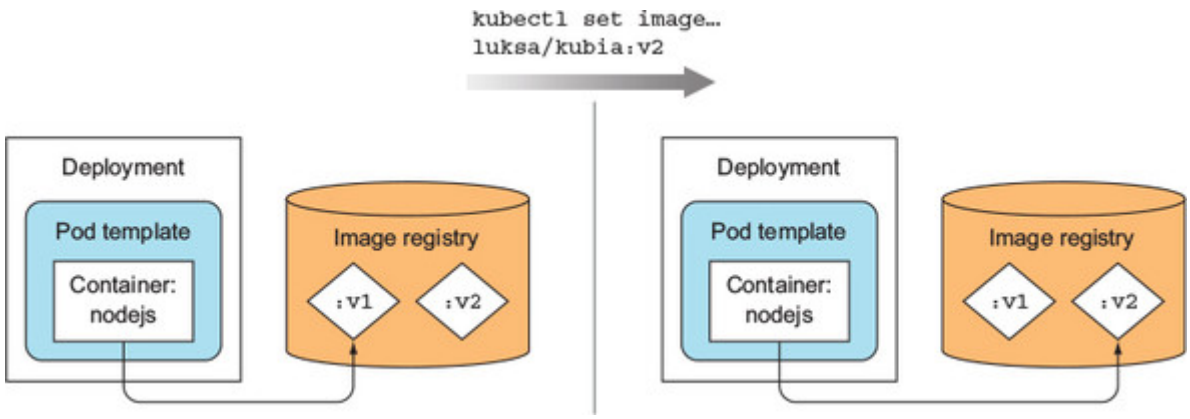
롤링업데이트 시작

롤링 업데이트 프로세스의 진행 사항을 추적 → 다른 터미널에서 curl 요청 다시 실행

```
while true; do curl http://130.211.109.222; done
```

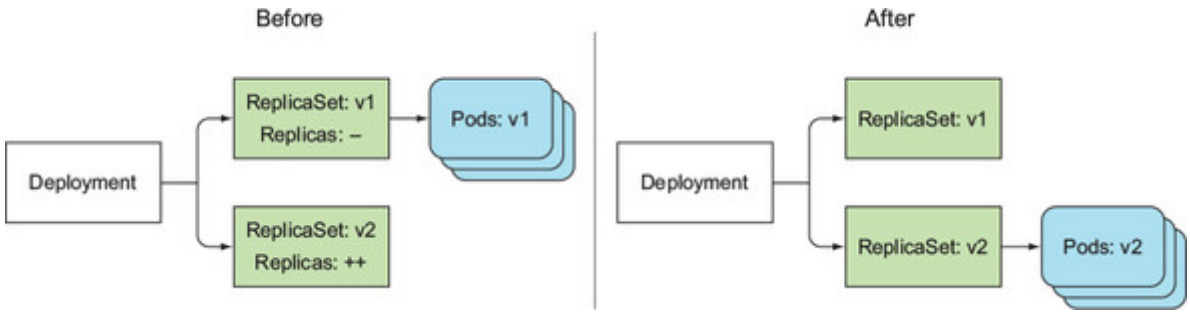
롤아웃을 위해 이미지를 v2로 변경

```
kubect set image deployment kukbia nodeJs=luksa/kubia:v2
```



디플로이먼트의 놀라움

디플로이먼트 리소스에서 파드 템플릿을 변경하는 것만으로 앱을 최신버전으로 업데이트



- kubectl rolling-update를 사용했을 때처럼 Kubectl 클라이언트가 프로세스를 수행하지 않음

  → 쿠버네티스 컨트롤 플레인의 일부로 실행되는 컨트롤러가 업데이트 수행

- 업데이트 하는 동안 디플로이먼트 영역에서 발생한 이벤트는 kubectl rolling-update를 수행하는 동안 발생한 이벤트와 유사 (추가 rs 생성, scaleup, 이전 rs 스케일 다운)

[차이점!!!] : 기존 rs가 삭제되지 않고 존재 (비활성화)

→ 이 부분의 목적은 차차 설명

3. 디플로이먼트 롤백

먼저 v3을 준비

v3/app.js 참고 → 처음 요청 4개만 제대로 처리하는 buggy한 버전임

```
kubectl set image deployment kubia nodejs=luksa/kubia:v3
kubectl rollout status deployment kubia
```

이제 새 버전이 라이브 상태이지만 오류를 출력하기 시작

```
while true; do curl <새 서비스 IP>; done
```

롤아웃 되돌리기

```
kubectl rollout undo deployment kubia
```

→ 디플로이먼트가 이전 버전으로 롤백함

디플로이먼트 롤아웃 이력 표시

```
kubectl rollout history deployment kubia
```

특정 디플로이먼트 revision으로 롤백

```
kubectl rollout undo deployment kubia --to-revision=1
```

비활성화된 레플리카셋이 남아있는 이유가 이것이다. 이게 남아있지 않으면 디플로이먼트 기록에서 특정버전을 잃어 롤백 불가

→ deployment 리소스의 editionHistoryLimit 속성에 의해 제한됨 (기본값 2)

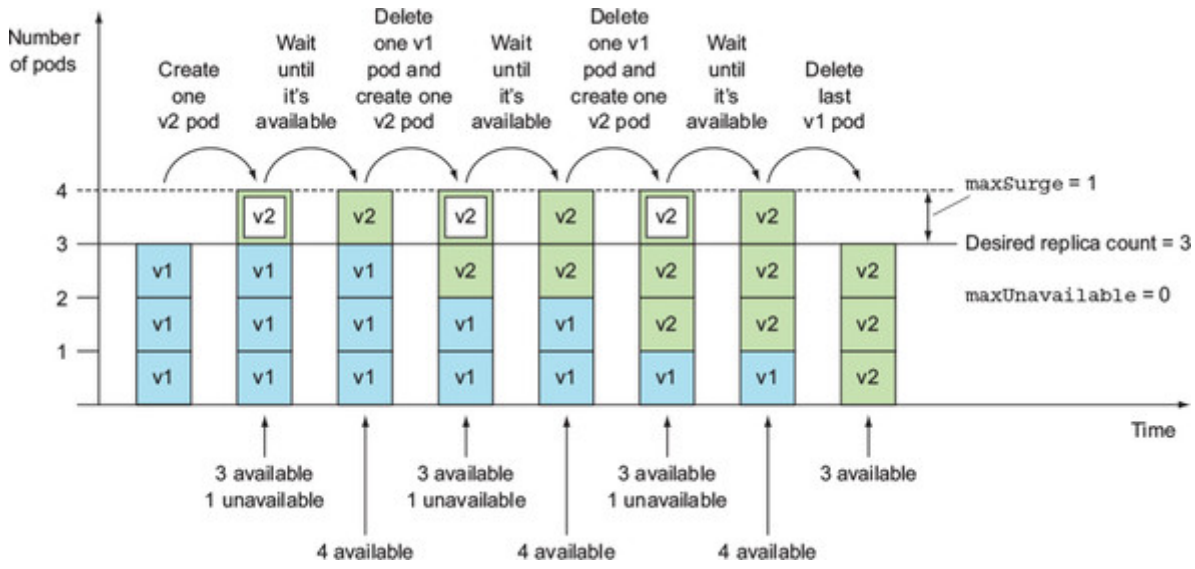
4. 롤아웃 속도 제어

롤링 업데이트 전략의 두가지 추가 속성을 통해 새 파드를 만들고 기존 파드를 삭제하는 방법 구상 가능

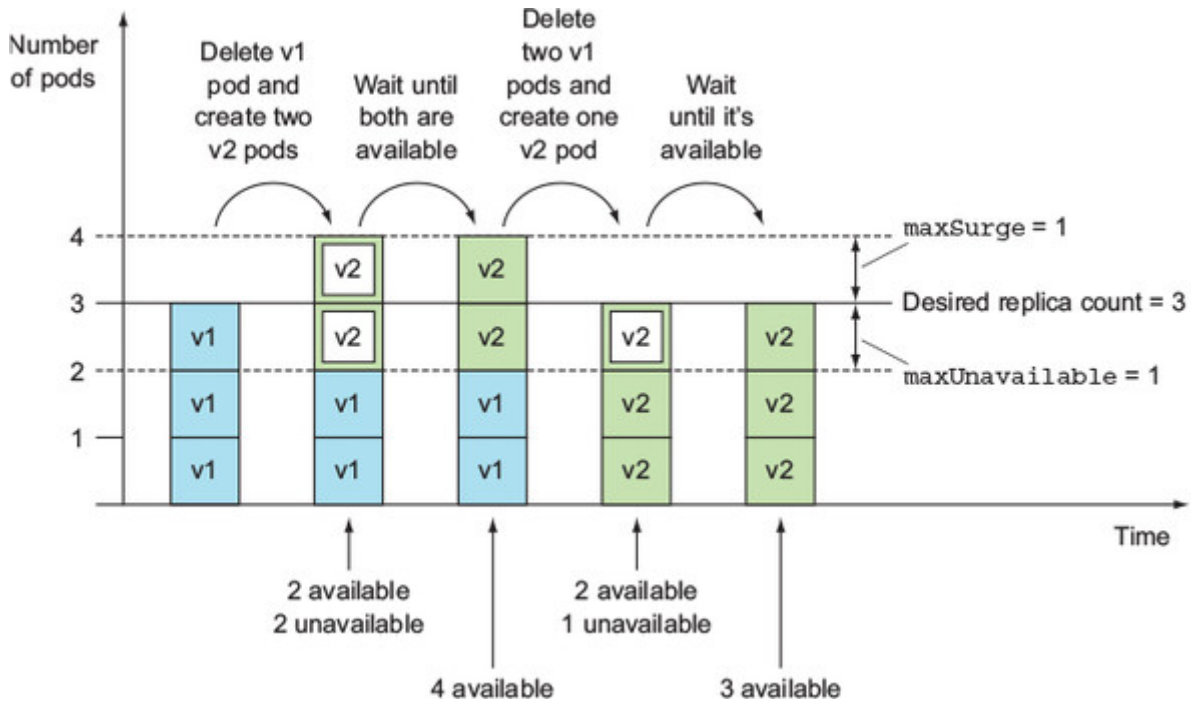
롤링 업데이트 전략의 maxSurge와 maxUnavailable 속성 소개

롤링업데이트 중 한번에 몇개의 파드를 교체할지 결정

- maxSurge: 디플로이먼트가 의도하는 레플리카 수보다 얼마나 많은 파드 인스턴스 수를 허용할 수 있는지 결정 (기본 25%로 설정)
- maxUnavailable : 업데이트 중 의도하는 레플리카 수를 기준으로 사용할 수 없는 파드 인스턴스 수를 결정 (기본 25% → 사용가능한 파드 인스턴스 수는 의도하는 레플리카 수의 75%이하로 떨어지지 않음)



레플리카 3개와 기본 maxSurge/maxUnavailable이 있는 디플로이먼트의 롤링 업데이트



maxSurge=1, maxUnavailable=1인 디플로이먼트 롤링 업데이트

5. 롤아웃 프로세스

롤아웃 일시 중지

버그픽스를 한 v4 이미지로 변경해 롤아웃 시작한 즉시 이를 일시 중지한다

```
kubcetl set image deployment kubia nodejs=luksa/kubia:v4
kubectl rollout pause deployment kubia
```

이렇게 하면...! 카나리 릴리스를 효과적으로 실행할 수 있당

카나리 릴리스: 잘못된 버전의 앱이 롤아웃돼 모든 사용자에게 영향을 주는 위험을 최소화하는 기술

- 새 버전을 모든 사람에게 롤아웃 하는 대신 하나/적은 수의 이전 파드만 새 버전으로 바꾼다
- 소수의 이용자만 초기에 새 버전을 이용하게 되는것
- 새 버전 정상동작 확인 후 나머지 모든 파드를 통해 롤아웃 지속해나감

롤아웃 재개

```
kubectl rollout resume deployment kubia
```

6. 잘못된 버전의 롤아웃 방지

minReadySeconds의 기능을 이용하면됨

- 오작동 버전의 배포를 방지하는 기능
- 롤아웃 속도를 늦춰 롤링 업데이트 과정을 직접 볼 수 있고, 모든 파드를 한번에 교체 안함

minReadySeconds의 적용 가능성 이해

파드를 사용 가능한 것으로 취급하기 전에 새로 만든 파드를 준비할 시간 지정

1. 모든 파드의 레디니스 프로브가 성공하면 파드가 준비됨
2. minReadySecons가 지나기 전에 새 파드가 작동하지 않음
3. 레디니스 프로브가 실패하기 시작 → 새 버전의 롤아웃이 효과적으로 차단됨

→ 에어백 역할

버전 v3가 완전히 롤아웃되는 것을 방지하기 위한 레디니스 프로브 정의

v3를 다시 배포하지만, 이번에는 파드에 적절한 레디니스 프로브가 정의되어있어야함

+) 현재 버전=v4이므로 시작전에 v2로 롤백

- 이미지를 변경하고 레디니스 프로브를 한 번에 추가하려면 kubectl apply 사용
- kubia-deployment-v3-with-readinesscheck.yaml 참고
- minReadySeconds(10)와 maxUnavailable(0), readinessProbe의 periodSeconds(1) 설정

kubectl apply를 통한 디플로이먼트 업데이트

```
kubectl apply -f kubia-deployment-v3-with-readinesscheck.yaml
kubectl rollout status deployment kubia
whiel true; do curl <새 IP주소>; done
```

근데 v3 파드에 접근할 수 없음 → 파드가 아직 준비되지 않았기 때문

레디니스 프로브가 잘못된 버전으로 롤아웃되는 것을 방지하는 법

새 파드가 시작되자마자 레디니스 프로브가 매초마다 시작됨

- 앱이 5번째 요청부터 http 500 반환 → 이때부터 레디니스 프로브가 실패하기 시작
- 이 파드는 서비스의 엔드포인트에서 제거됨
- curl요청에서 서비스를 시작할때까지 파드는 준비되지 않은것으로 표시됨 → 실패의 이유
- 클라이언트가 제대로 동작하지 않는 파드에 접근하지 못하게 하는것!@!!!!

rollout status 명령어 결과를 보면 하나의 새 레플리카만 시작됐음을 보여줌

→ 새 파드를 사용할 수 없으므로 롤아웃 프로세스가 계속되지 않음

사용가능한 것으로 간주되려면 10초 이상 준비되어야 함!!(레디니스 프로브가 10초 뒤에 검증)

→ 사용 가능할때까지 롤아웃 프로세스는 새 파드를 만들지 않음

→ maxUnavailable 속성이 0이므로 원래 파드도 제거하지 않음

> 💡 롤아웃이 모든 파드를 결함이 있는 v3로 교체하지 않았기 때문에 사용자에게 부정적인 영향이 발생하지 않았음 (레디니스 프로브 사용했기 때문)

롤아웃 데드라인 설정

kubectl describe로 디플로이먼트 조건확인

```
kubectl describe deploy kubia
```

Reason에 ProgressDeadlineExceeded로 나올것임

디플로이먼트가 실패한것으로 간주되는 시간은 디플로이먼트 스펙의 progressDeadlineSeconds 속성으로 설정 가능

잘못된 롤아웃 중지

롤아웃이 계속 진행되지 않기 때문에 ,,, 롤아웃을 취소해서 중단시키자

```
kubectl rollout undo deployment kubia
```