Red Hat OpenShift Pipelines 1.15

성능 및 리소스 사용 관리

OpenShift Pipelines에서 리소스 사용량 관리

Last Updated: 2024-09-02

Red Hat OpenShift Pipelines 1.15 성능 및 리소스 사용 관리

OpenShift Pipelines에서 리소스 사용량 관리

법적 공지

초록

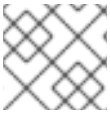
이 문서에서는 OpenShift Pipelines에서 리소스 사용 관리에 대한 정보를 제공합니다.

차례

1장. OPENSHIFT PIPELINES 성능 관리

OpenShift Pipelines 설치가 동시에 많은 작업을 실행하는 경우 성능이 저하될 수 있습니다. 속도 저하 및 파이프라인 실행 실패가 발생할 수 있습니다.

Red Hat 테스트에서는 AWS(Amazon Web Services) **m6a.2xlarge** 노드에서 실행되는 3-노드 OpenShift Container Platform 클러스터에서 최대 60개의 간단한 테스트 파이프라인이 상당한 실패나 지연 없이 동시에 실행되었습니다. 더 많은 파이프라인이 동시에 실행되면 실패한 파이프라인 실행 수, 파이프라인 실행의 평균 기간, Pod 생성 대기 시간, 작업 대기열 깊이 및 보류 중인 Pod 수가 증가했습니다. 이 테스트는 Red Hat OpenShift Pipelines 버전 1.13에서 수행되었으며 버전 1.12에서는 통계적으로 큰 차이가 관찰되지 않았습니다.



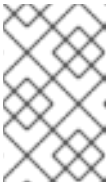
참고

이러한 결과는 테스트 구성에 따라 달라집니다. 구성의 성능 결과는 다를 수 있습니다.

1.1. OPENSHIFT PIPELINES 성능 개선

파이프라인 실행 속도 저하 또는 최신 실패가 발생하는 경우 다음 단계를 수행하여 OpenShift Pipelines의 성능을 개선할 수 있습니다.

- OpenShift Pipelines가 실행되는 OpenShift Container Platform 클러스터에서 노드의 리소스 사용량을 모니터링합니다. 리소스 사용량이 높으면 노드 수를 늘립니다.
- 고가용성 모드를 활성화합니다. 이 모드는 작업 실행 및 파이프라인 실행을 위해 Pod를 생성하고 시작하는 컨트롤러에 영향을 미칩니다. Red Hat 테스트에서 고가용성 모드는 파이프라인 실행 시간과 **TaskRun** 리소스 CR 생성에서 작업 실행을 실행하는 Pod 시작까지 지연을 크게 줄일 수 있었습니다. 고가용성 모드를 활성화하려면 **TektonConfig** CR(사용자 정의 리소스)을 다음과 같이 변경합니다.
  - **pipeline.performance.disable-ha** 사양을 **false** 로 설정합니다.
  - **pipeline.performance.buckets** 사양을 **5** 에서 **10** 사이의 숫자로 설정합니다.
  - **pipeline.performance.replicas** 사양을 **2** 보다 크고 **pipeline.performance.buckets** 설정보다 낮거나 같은 값으로 설정합니다.



참고

버킷 및 복제본에 대해 서로 다른 수를 시도하여 성능에 미치는 영향을 확인할 수 있습니다. 일반적으로 숫자가 클수록 유용합니다. CPU 및 메모리 사용률을 포함하여 노드의 리소스 소모를 모니터링합니다.

1.2. 추가 리소스

- TektonConfig CR을 사용한 성능 튜닝

2장. OPENSHIFT PIPELINES의 리소스 사용량 감소

멀티 테넌트 환경에서 클러스터를 사용하는 경우 각 프로젝트 및 Kubernetes 오브젝트에 대한 CPU, 메모리 및 스토리지 리소스의 사용을 제어해야 합니다. 따라서 하나의 애플리케이션이 너무 많은 리소스를 소비하고 다른 애플리케이션에 영향을 주지 않도록 방지할 수 있습니다.

결과 pod에 설정된 최종 리소스 제한을 정의하기 위해 Red Hat OpenShift Pipelines는 리소스 할당량 제한 및 해당 Pod가 실행되는 프로젝트의 제한 범위를 사용합니다.

프로젝트의 리소스 사용을 제한하려면 다음을 수행할 수 있습니다.

- 리소스 할당량을 설정하고 관리하여 집계 리소스 사용을 제한합니다.
- Pod, 이미지, 이미지 스트림, 영구 볼륨 클레임과 같은 특정 오브젝트에 대한 제한 범위를 사용하여 리소스 사용을 제한합니다.

2.1. 파이프라인에서 리소스 사용 이해

각 작업은 **Task** 리소스의 **steps** 필드에 정의된 특정 순서로 실행하는 데 필요한 여러 단계로 구성됩니다. 모든 작업은 Pod로 실행되고 각 단계는 해당 Pod 내에서 컨테이너로 실행됩니다.

단계는 한 번에 하나씩 실행됩니다. 작업을 실행하는 Pod는 한 번에 작업에서 단일 컨테이너 이미지(단계)를 실행하기에 충분한 리소스만 요청하므로 작업의 모든 단계에 대한 리소스를 저장하지 않습니다.

spec 단계의 **Resources** 필드는 리소스 소비에 대한 제한을 지정합니다. 기본적으로 CPU, 메모리, 임시 스토리지에 대한 리소스 요청은 **BestEffort** (zero) 값으로 설정되거나 해당 프로젝트에서 제한 범위를 통해 설정된 최소값으로 설정됩니다.

리소스 요청 구성 및 단계 제한의 예

```
spec:
  steps:
  - name: <step_name>
    computeResources:
      requests:
        memory: 2Gi
        cpu: 600m
      limits:
        memory: 4Gi
        cpu: 900m
```

LimitRange 매개변수 및 컨테이너 리소스 요청에 대한 최소 값이 Pipeline 및 작업을 실행하는 프로젝트에 지정되면 Red Hat OpenShift Pipelines는 프로젝트의 모든 **LimitRange** 값을 살펴보고 0 대신 최소 값을 사용합니다.

프로젝트 수준에서 제한 범위 매개변수 구성 예

```
apiVersion: v1
kind: LimitRange
metadata:
  name: <limit_container_resource>
spec:
  limits:
  - max:
      cpu: "600m"
```

```
      memory: "2Gi"
    min:
      cpu: "200m"
      memory: "100Mi"
    default:
      cpu: "500m"
      memory: "800Mi"
    defaultRequest:
      cpu: "100m"
      memory: "100Mi"
    type: Container
...
```

2.2. 파이프라인에서 추가 리소스 소비 완화

Pod의 컨테이너에 리소스 제한이 설정된 경우 OpenShift Container Platform은 모든 컨테이너가 동시에 실행될 때 요청된 리소스 제한을 합계합니다.

호출된 작업에서 한 번에 한 단계를 실행하는 데 필요한 최소 리소스 양을 사용하기 위해 Red Hat OpenShift Pipelines는 가장 많은 리소스 양이 필요한 단계에 지정된 대로 최대 CPU, 메모리 및 임시 스토리지를 요청합니다. 이렇게 하면 모든 단계의 리소스 요구 사항이 충족됩니다. 최대 값 이외의 요청은 0으로 설정됩니다.

그러나 이 동작으로 인해 리소스 사용량이 필요 이상으로 증가할 수 있습니다. 리소스 할당량을 사용하는 경우 Pod를 예약할 수 없게 될 수도 있습니다.

예를 들어 스크립트를 사용하고 리소스 제한과 요청을 정의하지 않는 두 단계로 이루어진 작업을 살펴보겠습니다. 결과 pod에는 두 개의 init 컨테이너(한 개는 진입점 복사용, 다른 하나는 스크립트 작성용)와 두 개의 컨테이너(단계 당 하나씩)가 있습니다.

OpenShift Container Platform은 프로젝트에 필요한 리소스 요청 및 제한을 계산하기 위해 설정된 제한 범위를 사용합니다. 이 예에서는 프로젝트에서 다음 제한 범위를 설정합니다.

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-min-max-demo-lr
spec:
  limits:
  - max:
      memory: 1Gi
    min:
      memory: 500Mi
    type: Container
```

이 시나리오에서 각 init 컨테이너는 1Gi의 요청 메모리 (제한 범위의 최대 제한)를 사용하고 각 컨테이너는 500Mi의 요청 메모리를 사용합니다. 따라서 Pod의 총 메모리 요청은 2Gi입니다.

10단계의 작업에 동일한 제한 범위를 사용하는 경우 최종 메모리 요청은 5Gi로 각 단계에서 실제로 필요한 것보다 높은 500Mi입니다 (각 단계가 차례로 실행되기 때문).

따라서 리소스의 리소스 사용량을 줄이기 위해 다음을 수행할 수 있습니다.

- 스크립트 기능 및 동일한 이미지를 사용하여 서로 다른 단계를 한 단계로 그룹화하여 지정된 작업의 단계 수를 줄입니다. 이렇게 하면 요청된 최소 리소스가 줄어듭니다.

- 서로 상대적으로 독립적이며 자체적으로 실행할 수 있는 단계를 단일 작업 대신 여러 작업에 분산합니다. 이렇게 하면 각 작업의 단계 수가 줄어들어 각 작업에 대한 요청이 줄어들며, 리소스가 사용 가능할 때 스케줄러가 해당 단계를 실행할 수 있습니다.

2.3. 추가 리소스

- OpenShift Pipelines의 컴퓨팅 리소스 할당량 설정
- 프로젝트당 리소스 할당량
- 제한 범위를 사용하여 리소스 사용 제한
- Kubernetes에서 리소스 요청 및 제한

3장. OPENSHIFT PIPELINES의 컴퓨팅 리소스 할당량 설정

Red Hat OpenShift Pipelines의 **ResourceQuota** 오브젝트는 네임스페이스당 총 리소스 소비를 제어합니다. 오브젝트 유형에 따라 네임스페이스에서 생성된 오브젝트 수량을 제한하는 데 사용할 수 있습니다. 또한 네임스페이스에서 사용하는 총 컴퓨팅 리소스 양을 제한하기 위해 컴퓨팅 리소스 할당량을 지정할 수 있습니다.

그러나 전체 네임스페이스에 대한 할당량을 설정하는 대신 파이프라인 실행으로 인해 Pod에서 사용하는 컴퓨팅 리소스의 양을 제한할 수 있습니다. 현재 Red Hat OpenShift Pipelines에서는 파이프라인의 컴퓨팅 리소스 할당량을 직접 지정할 수 없습니다.

3.1. OPENSHIFT PIPELINES에서 컴퓨팅 리소스 사용을 제한하는 대체 방법

파이프라인에서 컴퓨팅 리소스 사용에 대한 어느 정도의 제어 권한을 얻으려면 다음과 같은 대체 방법을 고려하십시오.

- 작업의 각 단계에 대한 리소스 요청 및 제한을 설정합니다.

  예: 작업의 각 단계에 대한 리소스 요청 및 제한을 설정합니다.

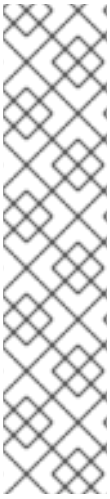
  ```
  ...
  spec:
    steps:
      - name: step-with-limts
        computeResources:
          requests:
            memory: 1Gi
            cpu: 500m
          limits:
            memory: 2Gi
            cpu: 800m
  ...
  ```

- **LimitRange** 오브젝트의 값을 지정하여 리소스 제한을 설정합니다. **LimitRange** 에 대한 자세한 내용은 제한 범위를 사용하여 리소스 소비 제한을 참조하십시오.

- 파이프라인 리소스 사용을 줄입니다.

- 프로젝트당 리소스 할당량을 설정하고 관리합니다.

- 파이프라인의 컴퓨팅 리소스 할당량은 파이프라인 실행에서 동시에 실행되는 Pod에서 사용하는 총 컴퓨팅 리소스 양과 동일해야 합니다. 그러나 작업을 실행하는 Pod는 사용 사례에 따라 컴퓨팅 리소스를 사용합니다. 예를 들어 Maven 빌드 작업에는 빌드하는 애플리케이션에 대해 서로 다른 컴퓨팅 리소스가 필요할 수 있습니다. 따라서 일반 파이프라인의 작업에 대한 컴퓨팅 리소스 할당량을 사전 결정할 수 없습니다. 컴퓨팅 리소스 사용을 예측할 수 있고 제어하려면 다양한 애플리케이션에 대해 사용자 지정된 파이프라인을 사용합니다.



참고

ResourceQuota 오브젝트로 구성된 네임스페이스에 Red Hat OpenShift Pipelines를 사용하는 경우 작업 실행 및 파이프라인 실행으로 인한 Pod가 **실패한 할당량: <quota name>과 같은 오류와 함께 실패할 수 있습니다. <quota name>은 cpu, memory를 지정해야 합니다**.

이 오류를 방지하려면 다음 중 하나를 수행하십시오.

- (권장) 네임스페이스에 대한 제한 범위를 지정합니다.
- 모든 컨테이너에 대한 요청 및 제한을 명시적으로 정의합니다.

자세한 내용은 문제 및 해결 방법을 참조하십시오.

이러한 접근 방식에서 사용 사례를 다루지 않는 경우 우선순위 클래스에 리소스 할당량을 사용하여 해결 방법을 구현할 수 있습니다.

3.2. 우선순위 클래스를 사용하여 파이프라인 리소스 할당량 지정

PriorityClass 오브젝트는 우선순위 클래스 이름을 상대 우선순위를 나타내는 정수 값에 매핑합니다. 값이 클수록 클래스의 우선 순위가 증가합니다. 우선순위 클래스를 생성한 후 사양에 우선순위 클래스 이름을 지정하는 Pod를 생성할 수 있습니다. 또한 Pod의 우선 순위에 따라 Pod의 시스템 리소스 사용을 제어할 수 있습니다.

파이프라인의 리소스 할당량을 지정하는 것은 파이프라인 실행으로 생성된 Pod 서브 세트에 대한 리소스 할당량을 설정하는 것과 유사합니다. 다음 단계에서는 우선순위 클래스를 기반으로 리소스 할당량을 지정하여 해결 방법의 예를 제공합니다.

프로세스

1. 파이프라인의 우선순위 클래스를 생성합니다.

   예: 파이프라인의 우선 순위 클래스

   ```
   apiVersion: scheduling.k8s.io/v1
   kind: PriorityClass
   metadata:
     name: pipeline1-pc
   value: 1000000
   description: "Priority class for pipeline1"
   ```

2. 파이프라인에 대한 리소스 할당량을 생성합니다.

   예: 파이프라인의 리소스 할당량

   ```
   apiVersion: v1
   kind: ResourceQuota
   metadata:
     name: pipeline1-rq
   spec:
     hard:
       cpu: "1000"
       memory: 200Gi
       pods: "10"
   ```

```
  scopeSelector:
    matchExpressions:
    - operator : In
      scopeName: PriorityClass
      values: ["pipeline1-pc"]
```

3. 파이프라인의 리소스 할당량 사용량을 확인합니다.

예: 파이프라인의 리소스 할당량 사용 확인

```
$ oc describe quota
```

샘플 출력

```
Name:       pipeline1-rq
Namespace:  default
Resource    Used  Hard
--------    ----  ----
cpu         0     1k
memory      0     200Gi
pods        0     10
```

Pod가 실행 중이 아니므로 할당량이 사용되지 않습니다.

4. 파이프라인 및 작업을 생성합니다.

예: 파이프라인의 YAML

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: maven-build
spec:
  params:
  - name: GIT_URL
  workspaces:
  - name: local-maven-repo
  - name: source
  tasks:
  - name: git-clone
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: git-clone
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: output
      workspace: source
    params:
    - name: URL
```

```
      value: $(params.GIT_URL)
  - name: build
    taskRef:
      name: mvn
    runAfter: ["git-clone"]
    params:
    - name: GOALS
      value: ["package"]
    workspaces:
    - name: maven-repo
      workspace: local-maven-repo
    - name: source
      workspace: source
  - name: int-test
    taskRef:
      name: mvn
    runAfter: ["build"]
    params:
    - name: GOALS
      value: ["verify"]
    workspaces:
    - name: maven-repo
      workspace: local-maven-repo
    - name: source
      workspace: source
  - name: gen-report
    taskRef:
      name: mvn
    runAfter: ["build"]
    params:
    - name: GOALS
      value: ["site"]
    workspaces:
    - name: maven-repo
      workspace: local-maven-repo
    - name: source
      workspace: source
```

예: 파이프라인의 작업의 YAML

```
apiVersion: tekton.dev/v1
kind: Task
metadata:
  name: mvn
spec:
  workspaces:
  - name: maven-repo
  - name: source
  params:
  - name: GOALS
    description: The Maven goals to run
    type: array
    default: ["package"]
  steps:
    - name: mvn
      image: gcr.io/cloud-builders/mvn
```

```
    workingDir: $(workspaces.source.path)
    command: ["/usr/bin/mvn"]
    args:
      - -Dmaven.repo.local=$(workspaces.maven-repo.path)
      - "$(params.GOALS)"
```

5. 파이프라인 실행을 생성하고 시작합니다.

예: 파이프라인 실행을 위한 YAML

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  generateName: petclinic-run-
spec:
  pipelineRef:
    name: maven-build
  params:
  - name: GIT_URL
    value: https://github.com/spring-projects/spring-petclinic
  taskRunTemplate:
    podTemplate:
      priorityClassName: pipeline1-pc
  workspaces:
  - name: local-maven-repo
    emptyDir: {}
  - name: source
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 200M
```

참고

파이프라인 실행이 **실패한 할당량으로 실패할 수 있습니다. <quota name>은 cpu, memory를 지정해야 합니다**.

이 오류를 방지하려면 네임스페이스의 제한 범위를 설정합니다. 여기서 **LimitRange** 오브젝트의 기본값이 빌드 프로세스 중에 생성된 Pod에 적용됩니다.

제한 범위 설정에 대한 자세한 내용은 *추가 리소스 섹션의 제한 범위를 사용하여 리소스 소비 제한을* 참조하십시오.

6. Pod가 생성되면 파이프라인 실행에 대한 리소스 할당량 사용량을 확인합니다.

예: 파이프라인의 리소스 할당량 사용 확인

```
$ oc describe quota
```

샘플 출력

```
Name:       pipeline1-rq
Namespace:  default
Resource    Used  Hard
--------    ----  ----
cpu         500m  1k
memory      10Gi  200Gi
pods        1     10
```

출력은 우선순위 클래스당 리소스 할당량을 지정하여 우선순위 클래스에 속하는 동시 실행 중인 모든 Pod에 대해 결합된 리소스 할당량을 관리할 수 있음을 나타냅니다.

3.3. 추가 리소스

- 제한 범위를 사용하여 리소스 사용 제한
- Kubernetes의 리소스 할당량
- Kubernetes의 제한 범위
- Kubernetes에서 리소스 요청 및 제한