Red Hat Ansible Automation Platform 2.4

Operator 기반 설치에 대한 Red Hat Ansible Automation Platform 성능 고려 사항

Operator 기반 설치 시 성능 향상을 위해 자동화 컨트롤러 구성

Last Updated: 2024-06-13

Red Hat Ansible Automation Platform 2.4 Operator 기반 설치에 대한 Red Hat Ansible Automation Platform 성능 고려 사항

Operator 기반 설치 시 성능 향상을 위해 자동화 컨트롤러 구성

법적 공지

초록

이 가이드에서는 자동화 컨트롤러 및 컨테이너 그룹 리소스 요청 및 기타 kubernetes 구성 옵션을 구성하여 Operator 기반 자동화 컨트롤러 설치 시 규모에 따라 작업을 보다 효율적으로 실행하는 방법에 대한 권장 사항을 제공합니다.

차례

머리말

Red Hat OpenShift Container Platform과 같은 컨테이너 오케스트레이션 플랫폼에 애플리케이션을 배포하면 운영 관점에서 다양한 이점이 있습니다. 예를 들어 애플리케이션의 기본 이미지 업데이트는 중단 없이 간단한 인플레이스 업그레이드를 통해 수행할 수 있습니다. 기존 가상 머신에 배포된 애플리케이션의 필수 운영 체제를 업그레이드하는 것은 훨씬 더 파괴적이고 위험한 프로세스가 될 수 있습니다.

애플리케이션 및 운영자 개발자는 애플리케이션 배포와 관련하여 OpenShift Container Platform 사용자에게 많은 옵션을 제공할 수 있지만 OpenShift Container Platform 구성에 따라 최종 사용자가 이러한 구성을 제공해야 합니다.

예를 들어 Openshift 클러스터에서 노드 레이블을 사용하면 다른 워크로드가 특정 노드에서 실행되도록 할 수 있습니다. 이러한 유형의 구성은 사용자가 Ansible Automation Platform Operator를 추측할 방법이 없으므로 제공해야 합니다.

RED HAT 문서에 관한 피드백 제공

이 문서를 개선하기 위한 제안이 있거나 오류를 찾을 수 있는 경우 https://access.redhat.com 에서 기술 지원에 문의하여 **docs-product** 구성 요소를 사용하여 Ansible Automation Platform Jira 프로젝트에 문제를 생성하십시오.

1장. POD 사양 수정

1.1. 소개

Pod의 Kubernetes 개념은 하나의 호스트에 함께 배포되는 하나 이상의 컨테이너이며 정의, 배포 또는 관리할 수 있는 최소 컴퓨팅 단위입니다.

Pod는 컨테이너에 대한 머신 인스턴스(실제 또는 가상)와 동일합니다. 각 Pod에는 자체 내부 IP 주소가 할당되므로 해당 Pod가 전체 포트 공간을 소유하고 Pod 내의 컨테이너는 로컬 스토리지와 네트워킹을 공유할 수 있습니다.

Pod에는 라이프사이클이 있습니다. 그런 다음 노드에서 실행되도록 할당된 다음 컨테이너가 종료되거나 다른 이유로 제거될 때까지 실행됩니다. Pod는 정책 및 종료 코드에 따라 종료 후 제거되거나 컨테이너 로그에 대한 액세스를 활성화하기 위해 유지될 수 있습니다.

Red Hat Ansible Automation Platform은 간단한 기본 Pod 사양을 제공하지만 기본 Pod 사양을 재정의하는 사용자 정의 YAML 또는 JSON 문서를 제공할 수 있습니다. 이 사용자 정의 문서에서는 유효한 Pod JSON 또는 YAML로 직렬화할 수 있는 **ImagePullSecrets** 와 같은 사용자 정의 필드를 사용합니다.

전체 옵션 목록은 Openshift Online 설명서에서 확인할 수 있습니다.

장기 실행 서비스를 제공하는 Pod의 예.

이 예제에서는 Pod의 많은 기능을 보여줍니다. 대부분 다른 주제에서 설명하므로 여기에서 간단히 설명합니다.

```
apiVersion: v1
kind: Pod
metadata:
  annotations: { ... }                    1
  labels:
    deployment: docker-registry-1
    deploymentconfig: docker-registry
    docker-registry: default
  generateName: docker-registry-1-        2
spec:
  containers:                             3
  - env:                                  4
    - name: OPENSHIFT_CA_DATA
      value: ...
    - name: OPENSHIFT_CERT_DATA
      value: ...
    - name: OPENSHIFT_INSECURE
      value: "false"
    - name: OPENSHIFT_KEY_DATA
      value: ...
    - name: OPENSHIFT_MASTER
      value: https://master.example.com:8443
    image: openshift/origin-docker-registry:v0.6.2  5
    imagePullPolicy: IfNotPresent
    name: registry
    ports:                                6
    - containerPort: 5000
      protocol: TCP
```

```
    resources: {}                          7
    securityContext: { ... }               8
    volumeMounts:                          9
    - mountPath: /registry
      name: registry-storage
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-br6yz
      readOnly: true
  dnsPolicy: ClusterFirst
  imagePullSecrets:                        10
  - name: default-dockercfg-at06w
  restartPolicy: Always                    11
  serviceAccount: default                  12
  volumes:                                 13
  - emptyDir: {}
    name: registry-storage
  - name: default-token-br6yz
    secret:
      secretName: default-token-br6yz
```

레이블	설명
주석:	Pod는 단일 작업에서 Pod 그룹을 선택하고 관리하는 데 사용할 수 있는 라벨을 하나 이상 사용하여 "태그를 지정"할 수 있습니다. 레이블은 메타데이터 해시의 key:value 형식으로 저장됩니다. 이 예제의 하나의 레이블은 **docker-registry=default** 입니다.
generateName:	Pod에는 해당 네임스페이스 내에 고유한 이름이 있어야 합니다. Pod 정의는 **generateName** 특성을 사용하여 이름 기반을 지정하고 임의의 문자를 자동으로 추가하여 고유한 이름을 생성할 수 있습니다.
컨테이너:	**컨테이너** 는 컨테이너 정의의 배열을 지정합니다. 이 경우(대부분의 경우와 마찬가지로) 하나의 컨테이너만 정의합니다.
env:	환경 변수는 각 컨테이너에 필요한 값을 전달합니다.
이미지:	Pod의 각 컨테이너는 자체 Docker 형식의 컨테이너 이미지에서 인스턴스화됩니다.
포트:	컨테이너는 Pod의 IP에서 사용할 수 있는 포트에 바인딩할 수 있습니다.
resources:	Pod를 지정할 때 컨테이너에 필요한 각 리소스의 양을 선택적으로 설명할 수 있습니다. 지정할 가장 일반적인 리소스는 CPU 및 메모리(RAM)입니다. 기타 리소스를 사용할 수 있습니다.
securityContext:	OpenShift Online은 권한이 있는 컨테이너로 실행할 수 있는지, 선택한 사용자로 실행할 수 있는지 여부를 지정하는 보안 컨텍스트를 정의합니다. 기본 컨텍스트는 매우 제한적이지만 관리자는 필요에 따라 변경할 수 있습니다.

레이블	설명
volumeMounts:	컨테이너는 컨테이너 내에서 외부 스토리지 볼륨을 마운트해야 하는 위치를 지정합니다. 이 경우 레지스트리 데이터를 저장하기 위한 볼륨이 있으며, 하나는 OpenShift Online API에 대해 요청하는 데 필요한 자격 증명에 대한 액세스용입니다.
ImagePullSecrets	Pod에는 일부 레지스트리에서 가져와야 하는 하나 이상의 컨테이너가 포함될 수 있습니다. 컨테이너가 인증이 필요한 레지스트리에서 제공하는 경우 네임스페이스에 **ImagePullSecrets** 목록을 참조하는 **ImagePullSecrets** 목록을 제공할 수 있습니다. 이러한 지정을 통해 Red Hat OpenShift Container Platform은 이미지를 가져올 때 컨테이너 레지스트리로 인증할 수 있습니다. 자세한 내용은 Kubernetes 문서의 Pod 및 컨테이너에 대한 리소스 관리를 참조하십시오.
restartPolicy:	Pod는 가능한 값 **Always**,**OnFailure** 및 **Never** 를 사용하여 정책을 다시 시작합니다. 기본값은 **Always** 입니다.
serviceAccount:	OpenShift Online API에 대해 요청하는 Pod는 요청 시 Pod에서 인증해야 하는 서비스 계정 사용자를 지정하는 **serviceAccount** 필드가 있는 일반적인 패턴입니다. 따라서 사용자 정의 인프라 구성 요소에 대한 액세스 권한을 세부적으로 제어할 수 있습니다.
volumes:	Pod는 사용할 컨테이너에서 사용할 수 있는 스토리지 볼륨을 정의합니다. 이 경우 레지스트리 스토리지의 임시 볼륨과 서비스 계정 인증 정보가 포함된 시크릿 볼륨을 제공합니다.

자동화 컨트롤러를 사용하고 자동화 컨트롤러 UI에서 Pod 사양을 편집하여 Kubernetes 기반 클러스터에서 작업을 실행하는 데 사용되는 Pod를 변경할 수 있습니다. 작업을 실행하는 Pod를 생성하는 데 사용되는 Pod 사양은 YAML 형식입니다. Pod 사양 편집에 대한 자세한 내용은 Pod 사양 사용자 지정을 참조하십시오.

1.1.1. Pod 사양 사용자 정의

다음 절차를 사용하여 Pod를 사용자 지정할 수 있습니다.

절차

1. 자동화 컨트롤러 UI에서 **관리 인스턴스 그룹으로이동합니다**.
2. **Pod 사양 사용자 지정을** 확인합니다.
3. **Pod Spec Override** 필드에서 토글을 사용하여 **Pod Spec Override** 필드를 활성화하고 확장하여 네임스페이스를 지정합니다.
4. **저장**을 클릭합니다.
5. 선택 사항: 추가 사용자 지정을 제공하려면 **Expand** 를 클릭하여 전체 사용자 지정 창을 확인합니다.

작업 시작 시 사용되는 이미지는 작업과 연결된 실행 환경에 따라 결정됩니다. 컨테이너 레지스트리 인증 정보가 실행 환경과 연결된 경우 자동화 컨트롤러는 **ImagePullSecret** 을 사용하여 이미지를 가져옵니다. 서비스 계정에 시크릿을 관리할 수 있는 권한을 부여하지 않으려면 **ImagePullSecret** 을 사전 생성하여 Pod 사양에 지정하고, 사용된 실행 환경에서 인증 정보를 생략해야 합니다.

1.1.2. Pod에서 다른 보안 레지스트리의 이미지를 참조하도록 활성화

컨테이너 그룹에서 인증 정보가 필요한 보안 레지스트리의 컨테이너를 사용하는 경우, 컨테이너 레지스트리 인증 정보를 작업 템플릿에 할당된 실행 환경과 연결할 수 있습니다. 자동화 컨트롤러는 이를 사용하여 컨테이너 그룹 작업이 실행되는 OpenShift Container Platform 네임스페이스에서 **ImagePullSecret** 을 생성하고 작업이 완료된 후 정리합니다.

또는 컨테이너 그룹 네임스페이스에 **ImagePullSecret** 이 이미 있는 경우 **ContainerGroup** 의 사용자 정의 Pod 사양에 **ImagePullSecret** 을 지정할 수 있습니다.

컨테이너 그룹에서 실행 중인 작업에서 사용하는 이미지는 항상 작업과 연결된 실행 환경으로 재정의됩니다.

사전 생성된 ImagePullSecrets (Advanced) 사용

이 워크플로를 사용하고 **ImagePullSecret** 을 사전 생성하려는 경우 이전에 보안 컨테이너 레지스트리에 액세스한 시스템의 로컬 **.dockercfg** 파일에서 생성하는 데 필요한 정보를 소싱할 수 있습니다.

절차

최신 Docker 클라이언트의 **.dockercfg 파일** 또는 **$HOME/.docker/config.json** 은 이전에 보안 또는 비보안 레지스트리에 로그인한 경우 정보를 저장하는 Docker 인증 정보 파일입니다.

1. 보안 레지스트리에 대한 **.dockercfg** 파일이 이미 있는 경우 다음 명령을 실행하여 해당 파일에서 보안을 생성할 수 있습니다.

```
$ oc create secret generic <pull_secret_name> \
--from-file=.dockercfg=<path/to/.dockercfg> \
--type=kubernetes.io/dockercfg
```

2. 또는 **$HOME/.docker/config.json** 파일이 있는 경우:

```
$ oc create secret generic <pull_secret_name> \
--from-file=.dockerconfigjson=<path/to/.docker/config.json> \
--type=kubernetes.io/dockerconfigjson
```

3. 보안 레지스트리에 대한 Docker 인증 정보 파일이 아직 없는 경우 다음 명령을 실행하여 보안을 생성할 수 있습니다.

```
$ oc create secret docker-registry <pull_secret_name> \
--docker-server=<registry_server> \
--docker-username=<user_name> \
--docker-password=<password> \
--docker-email=<email>
```

4. Pod의 이미지 가져오기에 시크릿을 사용하려면 서비스 계정에 이 시크릿을 추가해야 합니다. 이 예제의 서비스 계정 이름은 Pod에서 사용하는 서비스 계정 이름과 일치해야 합니다. 기본값은 기본 서비스 계정입니다.

```
$ oc secrets link default <pull_secret_name> --for=pull
```

5. 선택 사항: 빌드 이미지를 푸시하고 가져오는 데 보안을 사용하려면 Pod 내에서 보안을 마운트할 수 있어야 합니다. 다음을 실행하여 이 작업을 수행할 수 있습니다.

```
$ oc secrets link builder <pull_secret_name>
```

6. 선택 사항: 빌드의 경우 빌드 구성 내에서 가져오기 보안으로 보안을 참조해야 합니다.

컨테이너 그룹이 성공적으로 생성되면 새로 생성된 컨테이너 그룹의 **세부 정보** 탭이 유지됩니다. 이를 통해 컨테이너 그룹 정보를 검토하고 편집할 수 있습니다. **인스턴스 그룹** 링크에서 **편집** 아이콘을 클릭하면 열리는 메뉴와 같습니다. **인스턴스를 편집하고 이 인스턴스 그룹과 연결된 작업을 검토할 수도 있습니다.**

1.2. POD 및 컨테이너의 리소스 관리

Pod를 지정하면 컨테이너에 필요한 각 리소스의 양을 지정할 수 있습니다. 지정할 가장 일반적인 리소스는 CPU 및 메모리(RAM)입니다.

Pod에서 컨테이너에 대한 리소스 요청을 지정하면 kubernetes-scheduler에서 이 정보를 사용하여 Pod를 배치할 노드를 할당합니다.

컨테이너, *kubelet* 또는 노드 에이전트에 대한 리소스 제한을 지정하면 실행 중인 컨테이너가 설정한 제한보다 더 많은 리소스를 사용할 수 없도록 해당 제한을 적용합니다. kubelet은 특히 해당 컨테이너가 사용할 수 있도록 요청된 시스템 리소스 양을 예약합니다.

1.2.1. 요청 및 제한

Pod가 실행 중인 노드에 사용 가능한 리소스가 충분한 경우 컨테이너에서 해당 리소스에 대한 요청보다 더 많은 리소스를 사용할 수 있습니다. 그러나 컨테이너는 리소스 제한을 초과하여 사용할 수 없습니다.

예를 들어 컨테이너에 256MiB의 메모리 요청을 설정하고 해당 컨테이너가 8GiB 메모리가 있는 노드에 예약된 Pod에 있고 다른 Pod가 없는 경우 컨테이너에서 더 많은 RAM을 사용하려고 시도할 수 있습니다.

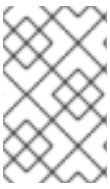
해당 컨테이너에 메모리 제한을 4GiB로 설정하면 kubelet 및 컨테이너 런타임에서 제한을 적용합니다. 런타임에서는 컨테이너가 구성된 리소스 제한보다 더 많은 것을 사용하지 못하도록 합니다.

컨테이너의 프로세스에서 허용된 메모리 양보다 많은 메모리를 사용하려고 하면 시스템 커널은 할당을 시도한 프로세스를 종료하며 OOM(*Out Of Memory*) 오류가 발생합니다.

제한은 다음 두 가지 방법으로 구현할 수 있습니다.

- **Reactively: 시스템이 위반을 감지하면 개입합니다.**
- **적용: 시스템에서 컨테이너가 제한을 초과하지 않도록 합니다.**

런타임마다 동일한 제한을 구현하는 다양한 방법이 있을 수 있습니다.



참고

리소스에 대한 제한을 지정하지만 요청을 지정하지 않고 승인 시간 메커니즘에서 해당 리소스에 대한 기본 요청을 적용하지 않은 경우 Kubernetes는 지정한 제한을 복사하고 해당 리소스에 대해 요청된 값으로 사용합니다.

1.2.2. 리소스 유형

CPU와 메모리는 모두 리소스 유형입니다. 리소스 유형에는 기본 단위가 있습니다. CPU는 컴퓨팅 처리를 나타내며 Kubernetes CPU 단위로 지정됩니다. 메모리는 바이트 단위로 지정됩니다.

CPU 및 메모리는 전체적으로 컴퓨팅 리소스 또는 리소스라고 합니다. 컴퓨팅 리소스는 요청, 할당 및 소비할 수 있는 측정 가능한 양입니다. API 리소스와 다릅니다. Pod 및 서비스와 같은 API 리소스는 Kubernetes API 서버를 통해 읽고 수정할 수 있는 오브젝트입니다.

1.2.3. Pod 및 컨테이너에 대한 리소스 요청 및 제한 지정

각 컨테이너에 다음을 포함하여 리소스 제한 및 요청을 지정할 수 있습니다.

```
spec.containers[].resources.limits.cpu
spec.containers[].resources.limits.memory
spec.containers[].resources.requests.cpu
spec.containers[].resources.requests.memory
```

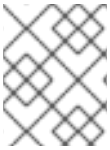
개별 컨테이너에 대한 요청 및 제한만 지정할 수 있지만 Pod의 전체 리소스 요청 및 제한을 생각하는 것도 유용합니다. 특정 리소스의 경우 Pod 리소스 요청 또는 제한은 Pod의 각 컨테이너에 대한 리소스 요청 또는 제한의 합계입니다.

1.2.4. Kubernetes의 리소스 단위

CPU 리소스 단위

CPU 리소스에 대한 제한 및 요청은 CPU 단위로 측정됩니다. Kubernetes에서 하나의 CPU 단위는 노드가 물리적 호스트인지 아니면 물리적 머신 내에서 실행되는 가상 머신인지에 따라 하나의 물리적 프로세서 코어 또는 하나의 가상 코어와 동일합니다.

소수 요청은 허용됩니다. **spec.containers[].resources.requests.cpu** 를 **0.5** 로 설정하여 컨테이너를 정의할 때 1.0 CPU를 요청한 경우와 비교하여 절반의 CPU를 요청합니다. CPU 리소스 단위의 경우 양 표현식 0.1은 100m 식과 동일합니다. 100m은 100m로 읽을 수 있으며 100밀리코어 또는 100밀리코어로 읽을 수 있습니다. millicpu 및 millicores는 동일한 것을 의미합니다. CPU 리소스는 항상 상대 양이 아닌 절대적인 리소스로 지정됩니다. 예를 들어 500m CPU는 컨테이너가 단일 코어, 듀얼 코어 또는 48코어 시스템에서 실행되는지와 동일한 양의 컴퓨팅 성능을 나타냅니다.



참고

CPU 단위가 1.0 또는 1000m 미만을 지정하려면 milliCPU 형식을 사용해야 합니다. 예를 들어 0.005 CPU가 아닌 5m을 사용합니다.

메모리 리소스 단위

메모리에 대한 제한 및 요청은 바이트 단위로 측정됩니다. E, P, T, G, M, k 중 하나를 사용하여 메모리를 일반 정수 또는 고정 소수점 숫자로 나타낼 수 있습니다. Ei, Pi, Ti, Gi, Mi, Ki와 같은 Power-of-two를 사용할 수도 있습니다. 예를 들어 다음은 대략 동일한 값을 나타냅니다.

```
128974848, 129e6, 129M,  128974848000m, 123Mi
```

접미사가 있는 경우 주의하십시오. 400m의 메모리를 요청하는 경우 400 메가바이트 (400Mi) 또는 400 메가바이트 (400M)가 아닌 0.4 바이트에 대한 요청입니다.

CPU 및 메모리 사양의 예

다음 클러스터에는 전용 100m CPU 및 250Mi로 작업 Pod를 예약할 수 있는 충분한 여유 리소스가 있습니다. 클러스터는 또한 2000m CPU 및 2Gi 메모리 전용 용도로 버스트에 견딜 수 있습니다.

```
spec:
  task_resource_requirements:
    requests:
      cpu: 100m
      memory: 250Mi
    limits:
      cpu: 2000m
      memory: 2Gi
```

자동화 컨트롤러는 제한 세트보다 더 많은 리소스를 사용하는 작업을 예약하지 않습니다. 작업 Pod에서 제한 세트보다 많은 리소스를 사용하는 경우 컨테이너는 Kubernetes에서 OOMKilled하고 재시작됩니다.

1.2.5. 리소스 요청에 대한 크기 권장 사항

컨테이너 그룹을 사용하는 모든 작업은 동일한 Pod 사양을 사용합니다. Pod 사양에는 작업을 실행하는 Pod에 대한 리소스 요청이 포함됩니다.

모든 작업은 동일한 리소스 요청을 사용합니다. Pod 사양에서 특정 작업에 대해 지정된 리소스 요청은 Kubernetes가 작업자 노드에서 사용 가능한 리소스를 기반으로 작업 Pod를 예약하는 방법에 영향을 미칩니다. 이는 기본값입니다.

- 하나의 포크에는 일반적으로 100Mb의 메모리가 필요합니다. **system_task_forks_mem** 을 사용하여 설정됩니다. 작업에 포크 5개가 있는 경우 작업 Pod 사양에서 500Mb의 메모리를 요청해야 합니다.
- 특히 높은 포크 값이 있거나 대규모 리소스 요청이 필요한 작업 템플릿의 경우 더 큰 리소스 요청을 나타내는 다른 Pod 사양으로 별도의 컨테이너 그룹을 생성해야 합니다. 그런 다음 작업 템플릿에 할당할 수 있습니다. 예를 들어 포크 값이 50인 작업 템플릿은 5GB 메모리를 요청하는 컨테이너 그룹과 페어링되어야 합니다.
- 단일 Pod가 작업을 포함할 수 없을 만큼 작업의 포크 값이 높은 경우 작업 분할 기능을 사용합니다. 이렇게 하면 개별 작업 "슬라이스"가 컨테이너 그룹에서 프로비저닝한 자동화 Pod에 적합하도록 인벤토리가 분할됩니다.

2장. 컨트롤 플레인 조정

컨트롤 플레인은 사용자 인터페이스를 제공하고 작업 스케줄링 및 시작을 처리하는 웹 및 작업 컨테이너가 포함된 자동화 컨트롤러 Pod를 나타냅니다. 자동화 컨트롤러 사용자 정의 리소스에서 *복제본* 수가 자동화 컨트롤러 배포의 자동화 컨트롤러 Pod 수를 결정합니다.

2.1. 작업 컨테이너에 대한 요청 및 제한

작업 컨테이너의 CPU 및 메모리 리소스 제한 값을 설정해야 합니다. 실행 노드에서 실행되는 각 작업에 대해 해당 작업에 대한 콜백 이벤트를 예약, 열기, 수신하려면 컨트롤 플레인에서 처리가 수행되어야 합니다.

자동화 컨트롤러의 Operator 배포의 경우 이 컨트롤 플레인 용량 사용은 컨트롤 플레인 인스턴스 그룹에서 추적됩니다. 사용 가능한 용량은 자동화 컨트롤러 사양의 task_resource_requirements 필드를 사용하거나 자동화 컨트롤러를 생성할 때 작업 컨테이너에 설정된 사용자 집합을 기준으로 결정됩니다.

클러스터에 적합한 메모리 및 CPU 리소스 제한을 설정할 수도 있습니다.

2.2. 컨테이너 리소스 요구 사항

하위 끝(요청)과 상한(제한) 모두에서 작업 및 웹 컨테이너의 리소스 요구 사항을 구성할 수 있습니다. 실행 환경 컨트롤 플레인은 프로젝트 업데이트에 사용되지만 일반적으로 작업의 기본 실행 환경과 동일합니다.

두 가지 모두 정의된 컨테이너에 더 높은 QoS 클래스가 제공되므로 리소스 요청 및 제한을 설정하는 것이 가장 좋습니다. *즉, 기본 노드가 제한되고 실행 중인 메모리 또는 기타 실패를 방지하기 위해 클러스터가 Pod를 다시 시도해야 하는 경우 컨트롤 플레인 Pod를 회수할 가능성이 적습니다.*

이러한 요청 및 제한은 자동화 컨트롤러의 제어 Pod에 적용되며 제한이 설정된 경우 인스턴스의 용량을 결정합니다. 기본적으로 작업 제어에는 하나의 용량 단위가 사용됩니다. 작업 컨테이너의 메모리 및 CPU 제한은 컨트롤 노드의 용량을 결정하는 데 사용됩니다. 이를 계산하는 방법에 대한 자세한 내용은 용량 알고리즘에 대한 결정 검색을 참조하십시오.

작업자 노드에서 예약된작업도 참조하십시오.

이름	설명	Default
web_resource_requirements	웹 컨테이너 리소스 요구사항	requests: {CPU: 100m, memory: 128Mi}
task_resource_requirements	작업 컨테이너 리소스 요구사항	requests: {CPU: 100m, memory: 128Mi}
ee_resource_requirements	EE 컨트롤 플레인 컨테이너 리소스 요구 사항	requests: {CPU: 100m, memory: 128Mi}
redis_resource_requirements	Redis 컨트롤 플레인 컨테이너 리소스 요구 사항	requests: {CPU:100m, memory: 128Mi}

***topology_spread_constraints** 를 사용하여 컨트롤 노드를 별도의 기본 Kubernetes 작업자 노드에 배포하는 것이 좋습니다. 적절한 요청 및 제한 세트는 노드의 실제 리소스와 동일한 합계가 있는 제한입니다. 제한만 설정하면 요청이 자동으로 제한과 동일하게 설정됩니다. 그러나 제어 Pod의 컨테이너 간 리소스 사용량*

의 일부 변형이 허용되므로 요청을 더 낮은 양으로 설정할 수 있습니다(예: 노드에서 사용 가능한 리소스의 25%). 작업자 노드에 4개의 CPU와 16GB의 RAM이 있는 클러스터의 컨테이너 사용자 지정의 예는 다음과 같습니다.

```
spec:
  ...
  web_resource_requirements:
    requests:
      cpu: 250m
      memory: 1Gi
    limits:
      cpu: 1000m
      memory: 4Gi
  task_resource_requirements:
    requests:
      cpu: 250m
      memory: 1Gi
    limits:
      cpu: 2000m
      memory: 4Gi
  redis_resource_requirements
    requests:
      cpu: 250m
      memory: 1Gi
    limits:
      cpu: 1000m
      memory: 4Gi
  ee_resource_requirements:
    requests:
      cpu: 250m
      memory: 1Gi
    limits:
      cpu: 1000m
      memory: 4Gi
```

2.3. 자동화 컨트롤러 설정을 통한 대체 용량 제한

OpenShift의 제어 노드의 용량은 메모리 및 CPU 제한에 따라 결정됩니다. 그러나 이러한 용량이 설정되지 않은 경우 실제로 기본 Kubernetes 노드의 CPU 및 메모리인 파일 시스템의 Pod에서 탐지한 메모리 및 CPU에 따라 용량을 결정합니다.

이로 인해 자동화 컨트롤러 Pod가 해당 노드의 유일한 Pod가 아닌 경우 기본 Kubernetes Pod에 문제가 발생할 수 있습니다. 작업 컨테이너에 제한을 직접 설정하지 않으려면 **extra_settings** 를 사용할 수 있습니다. 다음 구성 방법은 *사용자 정의 Pod 시간 초과* 섹션을 참조하십시오.

```
SYSTEM_TASK_ABS_MEM = 3gi
SYSTEM_TASK_ABS_CPU = 750m
```

이는 애플리케이션 내에서 자동화 컨트롤러에서 실행하려고 하는 작업 양을 제어하거나 Kubernetes 자체에서 CPU 제한 위험을 감수하지 않거나 메모리 사용량이 필요한 제한을 초과할 경우 다시 얻을 수 있는 소프트 제한으로 작동합니다. 이러한 설정은 Kubernetes 리소스 정의의 리소스 요청 및 제한에서 허용되는 동일한 형식을 허용합니다.

3장. 전용 노드 지정

Kubernetes 클러스터는 많은 가상 머신 또는 노드(일반적으로 2~20개 노드)에서 실행됩니다. Pod는 이러한 노드에서 예약할 수 있습니다. 새 Pod를 생성하거나 예약할 때 **topology_spread_constraints** 설정을 사용하여 예약 또는 생성될 때 새 Pod가 기본 노드에 배포되는 방법을 구성합니다.

해당 노드가 실패하면 해당 Pod에서 제공하는 서비스도 실패하므로 단일 노드에 Pod를 예약하지 마십시오.

자동화 작업 Pod에 대해 다른 노드에서 실행되도록 컨트롤 플레인 노드를 예약합니다. 컨트롤 플레인 Pod가 작업 Pod와 노드를 공유하는 경우 컨트롤 플레인이 리소스가 부족하여 전체 애플리케이션의 성능이 저하될 수 있습니다.

3.1. 특정 노드에 POD 할당

Operator에서 생성한 자동화 컨트롤러 Pod를 특정 노드 하위 집합에서 실행하도록 제한할 수 있습니다.

- **node_selector** 및 **postgres_selector** 는 지정된 모든 키 또는 값, 쌍과 일치하는 노드에서만 실행되도록 자동화 컨트롤러 Pod를 제한합니다.
- 허용 오차 및 **postgres_tolerations** 를 사용하면 자동화 컨트롤러 Pod를 일치하는 테인트가 있는 노드에 예약할 수 있습니다. 자세한 내용은 Kubernetes 문서 의 테인트 및 허용 오차를 참조하십시오.

다음 표는 YAML의 자동화 컨트롤러 사양 섹션에 설정할 수 있는 설정 및 필드(또는 OpenShift UI 양식 사용)를 보여줍니다.

이름	설명	Default
postgres_image	가져올 이미지의 경로	Postgres
postgres_image_version	가져올 이미지 버전	13
node_selector	AutomationController Pod의 nodeSelector	""
topology_spread_constraints	AutomationController Pod의 topologySpreadConstraints	""
허용 오차	AutomationController Pod의 허용 오차	""
annotations	AutomationController Pod의 주석	""
postgres_selector	Postgres Pod의 nodeSelector	""
postgres_tolerations	Postgres Pod의 허용 오차	""

topology_spread_constraints 를 사용하면 노드 선택기와 일치하는 컴퓨팅 노드에서 컨트롤 플레인 Pod를 분배하는 데 도움이 될 수 있습니다. 예를 들어 이 옵션의 **maxSkew** 매개변수가 **100** 으로 설정된 경우 사용 가능한 노드에 최대 분배를 의미합니다. 따라서 일치하는 컴퓨팅 노드와 세 개의 포드가 있는 경우 하나의 포드가 각 계산 노드에 할당됩니다. 이 매개변수를 사용하면 컨트롤 플레인 Pod가 서로 리소스에 대해 경쟁하지 못하도록 합니다.

컨트롤러 Pod를 특정 노드로 제한하는 사용자 정의 구성의 예

```
spec:
  ...
  node_selector: |
    disktype: ssd
    kubernetes.io/arch: amd64
    kubernetes.io/os: linux
  topology_spread_constraints: |
    - maxSkew: 100
      topologyKey: "topology.kubernetes.io/zone"
      whenUnsatisfiable: "ScheduleAnyway"
      labelSelector:
        matchLabels:
          app.kubernetes.io/name: "<resourcename>"
  tolerations: |
    - key: "dedicated"
      operator: "Equal"
      value: "AutomationController"
      effect: "NoSchedule"
  postgres_selector: |
    disktype: ssd
    kubernetes.io/arch: amd64
    kubernetes.io/os: linux
  postgres_tolerations: |
    - key: "dedicated"
      operator: "Equal"
      value: "AutomationController"
      effect: "NoSchedule"
```

3.2. 작업 실행을 위한 노드 지정

컨테이너 그룹 Pod 사양에 노드 선택기를 추가하여 특정 노드에서만 실행되도록 할 수 있습니다. 먼저 작업을 실행할 노드에 레이블을 추가합니다.

다음 절차에서는 노드에 레이블을 추가합니다.

절차

1. *레이블과 함께 클러스터의 노드를 나열합니다.*

   ```
   kubectl get nodes --show-labels
   ```

   출력은 다음과 유사합니다(예: 테이블에 표시됨).

이름	상태	역할	나이	버전	라벨
worker0	Ready	<none>	1d	v1.13.0	**...,kubernetes.io/hostname=worker0**

이름	상태	역할	나이	버전	라벨
worker1	Ready	<none>	1d	v1.13.0	**… ,kubernetes.io/hostname=worker1**
worker2	Ready	<none>	1d	v1.13.0	**… ,kubernetes.io/hostname=worker2**

2. *노드 중 하나를 선택하고 다음 명령을 사용하여 레이블을 추가합니다.*

```
kubectl label nodes <your-node-name> <aap_node_type>=<execution>
```

예를 들면 다음과 같습니다.

```
kubectl label nodes <your-node-name> disktype=ssd
```

*여기서 **<your-node-name >**은 선택한 노드의 이름입니다.*

3. *선택한 노드에 **disktype=ssd** 라벨이 있는지 확인합니다.*

```
kubectl get nodes --show-labels
```

4. *출력은 다음과 유사합니다(예: 테이블에 표시됨).*

이름	상태	역할	나이	버전	라벨
worker0	Ready	<none>	1d	v1.13.0	**… disktype=ssd,kubernetes.io/hostname=worker0**
worker1	Ready	<none>	1d	v1.13.0	**… ,kubernetes.io/hostname=worker1**
worker2	Ready	<none>	1d	v1.13.0	**… ,kubernetes.io/hostname=worker2**

*이제 **worker0** 노드에 **disktype=ssd** 레이블이 있는지 확인할 수 있습니다.*

5. *자동화 컨트롤러 UI에서 컨테이너 그룹의 사용자 지정 Pod 사양의 metadata 섹션에 해당 라벨을 지정합니다.*

```
apiVersion: v1
kind: Pod
metadata:
  disktype: ssd
  namespace: ansible-automation-platform
```

```
spec:
  serviceAccountName: default
  automountServiceAccountToken: false
  nodeSelector:
    aap_node_type: execution
  containers:
    - image: >-
      registry.redhat.io/ansible-automation-platform-22/ee-supported-
rhel8@sha256:d134e198b179d1b21d3f067d745dd1a8e28167235c312cdc233860410ea3ec3e
      name: worker
      args:
        - ansible-runner
        - worker
        - '--private-data-dir=/runner'
      resources:
        requests:
          cpu: 250m
          memory: 100Mi
```

추가 설정

***extra_settings** 를 사용하면 awx-operator를 사용하여 많은 사용자 지정 설정을 전달할 수 있습니다. **extra_settings** 매개변수는 **/etc/tower/settings.py** 에 추가되며 **extra_volumes** 매개변수 대신 사용 할 수 있습니다.*

이름	설명	Default
extra_settings	추가 설정	''

***extra_settings** 매개변수 구성 예*

```
spec:
  extra_settings:
    - setting: MAX_PAGE_SIZE
      value: "500"

    - setting: AUTH_LDAP_BIND_DN
      value: "cn=admin,dc=example,dc=com"

    - setting: SYSTEM_TASK_ABS_MEM
      value: "500"
```

3.3. 사용자 정의 POD 시간 초과

*자동화 컨트롤러의 컨테이너 그룹 작업은 Kubernetes API에 Pod를 제출하기 직전에 **running** 상태로 전환됩니다. 자동화 컨트롤러에서는 **AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT** 초 전에 Pod가 **Running** 상태가 될 것으로 예상합니다. 자동화 컨트롤러에서Running 상태가 되지 않는 작업을 취소하기 전에 더 오래 대기하려면 **AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT** 을 더 높은 값으로 설정할 수 있습니다. **AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT** 은 자동화 컨트롤러에서 Pod에서 Ansible 작업이 시작될 때까지 Pod 생성에서 대기하는 시간입니다. 리소스 제약 조건으로 인해 Pod를 예약할 수 없는 경우에도 시간을 확장할 수 있습니다. 자동화 컨트롤러 사양에서 **extra_settings** 를 사용하여 이 작업을 수행할 수 있습니다. 기본값은 2시간입니다.*

*이는 Kubernetes에서 예약할 수 있는 것보다 더 많은 작업을 일관되게 시작하는 데 사용되며 작업이 보류 중인 상태에서 **AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT** 보다 긴 기간을 보냅니다.*

제어 용량을 사용할 수 있을 때까지 작업이 시작되지 않습니다. 컨테이너 그룹에 실행할 용량이 있는 것보다 더 많은 작업이 시작되면 Kubernetes 작업자 노드를 확장하는 것이 좋습니다.

3.4. 작업자 노드에 예약된 작업

자동화 컨트롤러와 Kubernetes는 모두 작업 예약에 역할을 합니다.

작업이 시작되면 해당 종속성이 충족됩니다. 즉, 작업 템플릿, 프로젝트, 인벤토리 설정에 필요에 따라 자동화 컨트롤러에서 프로젝트 업데이트 또는 인벤토리 업데이트가 시작됩니다.

자동화 컨트롤러의 다른 비즈니스 논리에 의해 작업이 차단되지 않고 작업을 시작할 컨트롤 플레인에 컨트롤 용량이 있는 경우 작업이 디스패처에 제출됩니다. 작업을 제어하는 "비용"의 기본 설정은 1 용량 입니다. 따라서 용량이 100개인 제어 Pod는 한 번에 최대 100개의 작업을 제어할 수 있습니다. 제어 용량이 지정된 경우 작업이 보류 중에서 waiting 로 전환됩니다.

제어 계획 Pod의 백그라운드 프로세스인 디스패처는 작업을 실행하기 위한 작업자 프로세스를 시작합니다. 컨테이너 그룹과 연결된 서비스 계정을 사용하여 Kubernetes API와 통신하고 자동화 컨트롤러의 컨테이너 그룹에 정의된 대로 Pod 사양을 사용하여 Pod를 프로비저닝합니다. 자동화 컨트롤러의 작업 상태는 실행 중 으로 표시됩니다.

*Kubernetes에서 Pod를 예약합니다. Pod는 **AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT** 의 보류 중 상태로 유지될 수 있습니다. **ResourceQuota** 를 통해 Pod가 거부되면 작업이보류 중인 상태에서 다시 시작됩니다. 네임스페이스의 Pod에서 사용할 수 있는 리소스 수를 제한하도록 네임스페이스에 리소스 할당량을 구성할 수 있습니다. ResourceQuotas에 대한 자세한 내용은 Resource Quotas 를 참조하십시오.*

4장. OPENSHIFT CONTAINER PLATFORM에서 ANSIBLE 자동화 컨트롤러 구성

Kubernetes 업그레이드 중에 자동화 컨트롤러가 실행 중이어야 합니다.

4.1. OPENSHIFT CONTAINER PLATFORM 업그레이드 중 다운타임 최소화

자동화 컨트롤러에서 다음과 같은 구성을 변경하여 업그레이드 중 다운타임을 최소화합니다.

사전 요구 사항

- Ansible Automation Platform 2.4
- Ansible 자동화 컨트롤러 4.4
- OpenShift Container Platform
  - > 4.10.42
  - > 4.11.16
  - > 4.12.0
- Postgres의 HA(고가용성) 배포
- 자동화 컨트롤러 Pod를 예약할 수 있는 여러 작업자 노드

절차

1. AutomationController 사양에서 **RECEPTOR_KUBE_SUPPORT_RECONNECT** 를 활성화합니다.

```
apiVersion: automationcontroller.ansible.com/v1beta1
kind: AutomationController
metadata:
  ...
spec:
  ...
  ee_extra_env: |
    - name: RECEPTOR_KUBE_SUPPORT_RECONNECT
      value: enabled
    ```
```

2. AutomationController 사양에서 정상 종료 기능을 활성화합니다.

```
termination_grace_period_seconds: <time to wait for job to finish>
```

3. AutomationController 사양의 배포를 분배하도록 Pod가 웹에 대해 **podAntiAffinity** 를 구성합니다.

```
task_affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
```
```

```
        labelSelector:
          matchExpressions:
          - key: app.kubernetes.io/name
            operator: In
            values:
            - awx-task
        topologyKey: topology.kubernetes.io/zone
      weight: 100
  web_affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: app.kubernetes.io/name
              operator: In
              values:
              - awx-web
          topologyKey: topology.kubernetes.io/zone
        weight: 100
```

4. ***OpenShift Container Platform*에서 *PodDisruptionBudget* 을 구성합니다.**

```
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: automationcontroller-job-pods
spec:
  maxUnavailable: 0
  selector:
    matchExpressions:
      - key: ansible-awx-job-id
        operator: Exists
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: automationcontroller-web-pods
spec:
  minAvailable: 1
  selector:
    matchExpressions:
      - key: app.kubernetes.io/name
        operator: In
        values:
          - <automationcontroller_instance_name>-web
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: automationcontroller-task-pods
spec:
  minAvailable: 1
  selector:
    matchExpressions:
```

```
- key: app.kubernetes.io/name
  operator: In
  values:
    - <automationcontroller_instance_name>-task
```