Red Hat Ansible Automation Platform 2.4

基于 Operator 的安装的 Red Hat Ansible Automation Platform 性能注意事项

配置自动化控制器以提高基于 Operator 的安装的性能

Last Updated: 2024-06-17

Red Hat Ansible Automation Platform 2.4 基于 Operator 的安装的 Red Hat Ansible Automation Platform 性能注意事项

配置自动化控制器以提高基于 Operator 的安装的性能

法律通告

摘要

本指南提供有关如何配置自动化控制器和容器组资源请求和其他 kubernetes 配置选项的建议，以根据基于自动化控制器的 Operator 安装，进行扩展来更有效地运行作业。

目录

前言

从操作的角度来看，将应用程序部署到容器编配平台（如 Red Hat OpenShift Container Platform）会有很多优点。例如，可以通过简单的原位升级对应用程序基础镜像进行更新，而无需中断。对于部署到传统虚拟机的应用程序所需的操作系统进行升级可能会对系统运行带来更多的破坏性和风险。

虽然应用程序和 Operator 开发人员可以为 OpenShift Container Platform 用户提供有关应用程序部署的许多选项，但这些配置必须由最终用户提供，因为它们依赖于 OpenShift Container Platform 的配置。

例如，在 Openshift 集群中使用节点标签有助于确保不同工作负载在特定节点上运行。这种类型的配置必须由用户提供，因为 Ansible Automation Platform Operator 无法实现这一点。

对红帽文档提供反馈

如果您对本文档有任何改进建议，或发现了任何错误，请通过 https://access.redhat.com 联系技术支持，以使用 **docs-product** 组件在 Ansible Automation Platform JIRA 项目中创建一个问题。

第 1 章 POD 规格的变化

1.1. 简介

pod 的 Kubernetes 概念是共同部署在同一主机上的一个或多个容器，也是可被定义、部署和管理的最小计算单元。

Pod 等同于一个容器的虚拟机实例（物理或虚拟）。每个 pod 分配有自己的内部 IP 地址，因此拥有完整的端口空间，并且 pod 内的容器可以共享其本地存储和网络。

Pod 具有生命周期。它们被定义，然后分配到节点上运行，然后运行它们直到容器退出或它们因为其他原因被删除为止。根据策略和退出代码，Pod 可在退出后删除，或者被保留下来以启用对容器日志的访问。

Red Hat Ansible Automation Platform 提供了一个简单的默认 pod 规格，但您可以提供一个自定义 YAML 或 JSON 文档来覆盖默认 pod 规格。此自定义文档使用自定义字段，如 **ImagePullSecrets**，它们可以被序列化为有效的 Pod JSON 或 YAML。

有关选项的完整列表，请参阅 Openshift Online 文档。

提供长时间运行的服务的 pod 示例。

这个示例展示了 pod 的许多特性，其中大多数已在其他主题中阐述，因此这里仅简略提及：

```
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
```

标签	描述
annotations:	pod 可以被"标上"一个或多个标签，然后使用这些标签在一个操作中选择和管理多组 pod。标签以 key:value 格式存储在 metadata 散列中。本例中的一个标签是 **docker-registry=default**。
generateName:	Pod 在其命名空间内需要具有唯一名称。一个 pod 定义可以使用 **generateName** 属性指定名称的基础，并且会自动添加随机字符来生成唯一名称。
containers:	**containers** 指定一组容器定义。在这种情况下（在大多数情况下），仅定义一个容器。
env:	环境变量将必要的值传递给每个容器。
image:	pod 中的每个容器使用自己的 Docker 格式的容器镜像进行安装。
ports:	容器可以绑定到 pod IP 上提供的端口。
resources:	指定 pod 时，您可以选择性地描述容器需要的资源量。要指定的最常见资源是 CPU 和内存 (RAM)。其他资源可用。
securityContext ：	OpenShift Online 为容器定义了一个安全上下文，用于指定是否允许其作为特权容器运行，作为所选用户运行，等等。默认上下文的限制性比较强，但管理员可以根据需要更改此设置。
volumeMounts:	容器指定外部存储卷应当挂载到容器内的什么位置上。在本例中，一个卷用于存储 registry 的数据，另一个卷则提供凭证的访问途径，registry 需要这些凭证来向 OpenShift Online API 发出请求。

标签	描述
ImagePullSecrets	一个 pod 可以包含一个或多个容器，这些容器必须从某些 registry 中拉取。如果容器来自需要身份验证的 registry，您可以给出一个 **ImagePullSecrets** 列表，该列表引用命名空间中存在的 **ImagePullSecrets**。指定这些可让 Red Hat OpenShift Container Platform 在拉取镜像时与容器 registry 进行身份验证。如需更多信息，请参阅 Kubernetes 文档中的 Pod 和容器的资源管理。
restartPolicy:	pod 重启策略，可能的值有 **Always**、**OnFailure** 和 **Never**。默认值为 **Always**。
serviceAccount:	Pod 对 OpenShift Online API 发出请求是一种比较常见的模式，它有一个 **serviceAccount** 字段，用于指定 pod 在发出请求时使用哪个服务帐户用户进行身份验证。这可以为自定义基础架构组件提供精细的访问控制。
volumes:	pod 定义了可供其容器使用的存储卷。在本例中，它提供了一个用于存储 registry 的临时卷，以及一个包含服务帐户凭证的 secret 卷。

您可以使用自动化控制器并在自动化控制器 UI 中编辑 pod 规格来更改用于运行 Kubernetes 集群中作业的 pod。用于创建运行作业的 pod 的 pod 规格，采用 YAML 格式。有关编辑 pod 规格的更多信息，请参阅自定义 pod 规格。

1.1.1. 自定义 pod 规格

您可以使用以下步骤自定义 pod。

流程

1. 在自动化控制器 UI 中，前往 **Administration → Instance Groups**。
2. 检查**自定义 pod 规格**。
3. 在 **Pod Spec Override** 字段中，使用切换来指定命名空间来启用和扩展 **Pod Spec Override** 字段。
4. 点击 **Save**。
5. 可选：如果要提供额外的自定义，点 **Expand** 查看整个自定义窗口。

作业启动时使用的镜像由与作业关联的执行环境决定。如果 Container Registry 凭证与执行环境关联，则自动化控制器将使用 **ImagePullSecret** 来拉取镜像。如果您不希望授予服务帐户管理 secret 的权限，您必须预先创建 **ImagePullSecret**，在 pod 规格中指定它，并从使用的执行环境省略任何凭证。

1.1.2. 启用 Pod 引用其他安全 registry 中的镜像

如果容器组使用需要凭证的安全 registry 中的容器，您可以将 Container Registry 凭证与分配给作业模板的 Execution Environment 关联。自动化控制器使用它来在容器组作业运行的 OpenShift Container Platform 命名空间中创建 **ImagePullSecret**，并在作业完成后进行清理。

另外，如果容器组命名空间中已存在 **ImagePullSecret**，您可以在 **ContainerGroup** 的自定义 pod 规格中指定 ImagePullSecret。

请注意，容器组中运行的作业所使用的镜像始终被与作业关联的执行环境覆盖。

使用预先创建的 ImagePullSecrets （高级）

如果要使用此工作流并预先创建 **ImagePullSecret**，您可以提供必要的信息，以便从之前访问安全容器 registry 的系统上的本地 **.dockercfg** 文件创建它。

步骤

用于较新的 Docker 客户端的 **.dockercfg** 文件或 **$HOME/.docker/config.json** 是一个 Docker 凭证文件，如果您之前已登录到安全或不安全的 registry，则该文件会保存您的信息。

1. 对于安全的 registry 已有一个 **.dockercfg** 文件，您可以运行以下命令来从该文件中创建 secret：

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

2. 或者，如果您有一个 **$HOME/.docker/config.json** 文件：

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

3. 如果您还没有安全 registry 的 Docker 凭证文件，您可以通过运行以下命令来创建 secret：

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

4. 要使用 secret 为 Pod 拉取镜像，您必须将 secret 添加到您的服务帐户中。本例中服务帐户的名称应与 Pod 使用的服务帐户的名称匹配。默认为 default 服务帐户。

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

5. 可选：要使用 secret 来推送和拉取 (pull) 构建镜像，该 secret 必须可在 pod 内挂载。您可通过运行以下命令实现这一目的：

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

6. 可选： 对于构建，还必须将 secret 引用为构建配置中的 pull secret。

成功创建容器组后，新创建的容器组的 **Details** 选项卡将保留。这可让您查看和编辑容器组信息。这与您点 **Instance Group** 链接中的 **Edit** 图标 ✎ 时打开的菜单相同。您还可以编辑实例，并查看与此实例组关联的作业。

1.2. POD 和容器的资源管理

指定 pod 时，您可以指定容器需要的每个资源量。要指定的最常见资源是 CPU 和内存 (RAM)。

当您为 Pod 中容器指定资源请求时，kubernetes-scheduler 会使用此信息分配节点来放置 Pod。

当您为容器、*kubelet* 或节点代理指定资源限值时，会强制实施这些限制，以便允许正在运行的容器使用该资源比您设置的限制更多。kubelet 还至少保留针对该容器使用的系统资源的请求数量。

1.2.1. 请求和限值

如果运行 pod 的节点有足够的可用资源，容器可以使用超过其对该资源的请求的资源。但是，容器不允许使用超过其资源的限值。

例如，如果您为容器设置了 256 MiB 的内存请求，并且该容器位于调度到具有 8GiB 内存且没有其他 pod 的节点的 pod 中，则容器可以尝试使用更多 RAM。

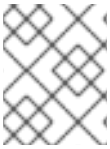
如果为该容器设置了 4GiB 的内存限值，kubelet 和容器运行时会强制实施限制。运行时可防止容器使用超过配置的资源限值。

如果容器中的进程尝试消耗超过允许的内存量，系统内核会终止尝试分配的进程，并显示 *Out Of Memory* (OOM) 错误。

您可以通过两种方式实施限制：

- 通过重新主动：系统在看到违反情况后进行干预。
- 通过强制：系统可防止容器超过限制。

不同的运行时可以有不同的方法来实施相同的限制。



注意

如果您为资源指定限制，但没有指定任何请求，且没有为该资源应用一个默认请求，则 Kubernetes 会复制您指定的限制，并将它用作资源的请求值。

1.2.2. 资源类型

CPU 和内存都是资源类型。资源类型具有一个基本单元。CPU 代表计算处理，并以 Kubernetes CPU 单位指定。内存以字节为单位指定。

CPU 和内存统称为计算资源或资源。计算资源是可请求、分配和消耗的可测量数量。它们与 API 资源不同。API 资源（如 pod 和服务）是可通过 Kubernetes API 服务器读取和修改的对象。

1.2.3. 为 pod 和容器指定资源请求和限值

对于每个容器，您可以指定资源限值和请求，包括：

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

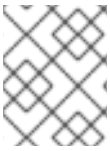
虽然您只能为单个容器指定请求和限值，但考虑 pod 的整体资源请求和限值也很有用。对于特定资源，Pod 资源请求和限值是 Pod 中每个容器这个类型的资源请求/限制总和。

1.2.4. Kubernetes 中的资源单元

CPU 资源单元

CPU 资源的限制和请求以 CPU 单位计算。在 Kubernetes 中，一个 CPU 单元等于一个物理处理器内核，或一个虚拟内核，具体取决于节点是物理主机还是在物理机中运行的虚拟机。

允许部分请求。当您定义将 **spec.containers[].resources.requests.cpu** 设置为 **0.5** 的容器时，如果请求 1.0 CPU，则请求一半时间作为 CPU 时间。对于 CPU 资源单元，数量表达式 0.1 等同于表达式 100m，它可以显示为一百个 millicpu 或一百个 millicores。millicpu 和 millicores 实际是相同的。CPU 资源始终指定为绝对数量的资源，而不是作为相对数量指定。例如，500m CPU 代表容器在单核、双核或 48 核机器上运行的计算能力。



注意

要指定小于 1.0 或 1000m 的 CPU 单元，您必须使用 milliCPU 格式。例如，使用 5m，而不是 0.005 CPU。

内存资源单元

内存的限制和请求以字节为单位。您可以使用这些数量后缀之一将内存表示为普通整数或固定点号： E、P、T、G、M、k。您还可以使用两的指数：Ei、Pi、Ti、Gi、Mi Ki。例如，以下代表相同的值：

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

请注意它们的后缀。如果您请求 400m 内存，则请求为 0.4 字节，而不是 400 兆字节 (400Mi) 或 400 MB (400M)。

CPU 和内存规格示例

以下集群有足够的可用资源来调度带有专用 100m CPU 和 250Mi 的任务 pod。集群也可以超过该专用用量，最多 2000m CPU 和 2Gi 内存。

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

自动化控制器不会调度使用超过限制集的资源的作业。如果任务 pod 使用的资源超过限制集，则容器会受 Kubernetes 和重启的 OOMKilled。

1.2.5. 资源请求的大小建议

所有使用容器组的作业都使用相同的 pod 规格。pod 规格包括运行作业的 pod 的资源请求。

所有作业都使用相同的资源请求。pod 规格中特定作业的指定资源请求会影响 Kubernetes 根据 worker 节点上可用的资源调度作业 pod。这些是默认值。

- 一个分叉通常需要 100Mb 内存。这通过使用 **system_task_forks_mem** 设置。如果您的作业有五个分叉，则作业 pod 规格必须请求 500Mb 内存。

- 对于具有特别高 fork 值或需要更大的资源请求的作业模板，您应该使用不同的 pod 规格创建单独的容器组，以指示更大的资源请求。然后，您可以将它分配给作业模板。例如，一个 fork 值为 50 的作业模板必须与请求 5GB 内存的容器组配对。

- 如果某个作业的 fork 值足够大，没有单个 pod 可以包含该作业，请使用作业分片功能。这会分割清单，使单个作业"slices"适合由容器组调配的自动化 pod。

第 2 章 控制平面（CONTROL PLANE）调整

control plane 指的是包含 web 和任务容器（除其他方面）的自动化控制器 pod，它提供了用户界面并处理作业的调度和启动。在自动化控制器自定义资源上，*副本*数量决定了自动化控制器部署中自动化控制器 pod 的数量。

2.1. 任务容器的请求和限值

您必须为任务容器的 CPU 和内存限值设置一个值。对于在执行节点上运行的每个作业，必须在 control plane 上处理来调度、打开和接收该作业的回调事件。

对于自动化控制器的 Operator 部署，此 control plane 容量使用量在 **controlplane** 实例组上进行跟踪。可用的容量取决于任务容器上的用户设置的限制、使用自动化控制器规格中的 **task_resource_requirements** 字段，或者在创建自动化控制器时在 OpenShift UI 中决定。

您还可以设置对集群有意义的内存和 CPU 资源限值。

2.2. 容器资源要求

您可以在下限 (requests) 和上限 (limits) 和 Web 容器配置资源要求。执行环境 control plane 用于项目更新，但通常与作业的默认执行环境相同。

设置资源请求和限值是最佳实践，因为已定义这两个容器具有较 *高质量的服务类*。这意味着，如果底层节点受资源约束，且集群必须获得 pod 以防止运行内存或其他故障，则 control plane pod 不太可能被获取。

这些请求和限值适用于自动化控制器的控制 pod，以及是否设置了限制，确定实例的 *容量*。默认情况下，控制作业需要一个容量单位。任务容器的内存和 CPU 限制用于决定控制节点的容量。有关如何计算它的更多信息，请参阅 Resouce determination for capacity algorithm。

另请参阅 worker 节点上调度的作业

名称	描述	default
web_resource_requirements	Web 容器资源要求	请求：{CPU: 100m, memory: 128Mi}
task_resource_requirements	任务容器资源要求	请求：{CPU: 100m, memory: 128Mi}
ee_resource_requirements	EE control plane 容器资源要求	请求：{CPU: 100m, memory: 128Mi}
redis_resource_requirements	Redis control plane 容器资源要求	requests: {CPU:100m, memory: 128Mi}

建议使用 **topology_spread_constraints** 将控制节点分散到单独的底层 Kubernetes worker 节点上。合理的请求和限值集合是限制其总和等于节点上实际资源的限值。如果只设定了 **limits**，则请求会自动设置为等于限制。但是，由于控制 pod 中容器之间的资源使用量不同，所以您可以 **将请求** 设置为较低数量，例如，节点上可用的资源的 25%。对于 worker 节点有 4 个 CPU 和 16 GB RAM 的集群，容器自定义示例如下：

```
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. 使用自动化控制器设置的替代容量限制

OpenShift 中控制节点的容量由内存和 CPU 限值决定。但是，如果没有设置它们，则容量由文件系统上的 pod 检测到的内存和 CPU 决定，而这是底层 Kubernetes 节点的 CPU 和内存。

如果自动化控制器 pod 不是该节点上的唯一 pod，这可能会导致意外出现底层 Kubernetes pod 的问题。如果您不想直接在任务容器上设置限制，您可以使用 **extra_settings**，请参阅 Custom pod timeout 部分中的 *Extra Settings* 以了解如何配置以下内容

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

这充当应用程序中的软限制，使自动化控制器能够控制它试图运行的工作量，同时不会面临 Kubernetes 本身的任何 CPU 节流，或者因为内存用量超过所需限制而获得。这些设置接受 Kubernetes 资源定义中资源请求和限值接受的相同格式。

第 3 章 指定专用节点

Kubernetes 集群在多个虚拟机或节点之上运行（通常位于 2 到 20 个节点之间的任何位置）。pod 可以调度到其中任何节点上。当您创建或调度新 pod 时，请使用 **topology_spread_constraints** 设置来配置在调度或创建时如何在底层节点上分布新 pod。

不要将 pod 调度到单个节点上，因为如果该节点失败，则这些 pod 提供的服务也会失败。

将 control plane 节点调度到不同节点上运行到自动化作业 pod。如果 control plane pod 与作业 pod 共享节点，control plane 可能会成为资源不足并降低整个应用程序的性能。

3.1. 将 POD 分配给特定的节点

您可以将 operator 创建的控制器 pod 限制为在特定的节点子集中运行。

- **node_selector** 和 **postgres_selector** 将自动化控制器 pod 限制为仅在匹配所有指定键或值对的节点上运行。
- **tolerations** 和 **postgres_tolerations** 允许将自动化控制器 pod 调度到具有匹配污点的节点。如需了解更多详细信息，请参阅 Kubernetes 文档中的 污点和容限。

下表显示了可以在 YAML 的自动化控制器规格部分（或使用 OpenShift UI 表单）上设置的设置和字段。

Name	描述	default
postgres_image	要拉取镜像的路径	postgres
postgres_image_version	要拉取的镜像版本	13
node_selector	Automationcontroller pod 的 nodeSelector	""
topology_spread_constraints	Automationcontroller pod 的 topologySpreadConstraints	""
容限（tolerations）	Automationcontroller pod 的容限	""
annotations	Automationcontroller pod 的注解	""
postgres_selector	Postgres pod 的 nodeSelector	""
postgres_tolerations	Postgres pod 的容限	""

topology_spread_constraints 有助于优化在与节点选择器匹配的计算节点上分散 control plane pod。例如，如果此选项的 **maxSkew** 参数设置为 **100**，这意味着最大地分散到可用节点上。因此，如果存在三个匹配的计算节点和三个 pod，则会为每个计算节点分配一个 pod。此参数有助于防止 control plane pod 相互竞争资源。

将控制器 pod 限制到特定节点的自定义配置示例

```
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. 指定用于作业执行的节点

您可以将节点选择器添加到容器组 pod 规格中，以确保它们仅针对某些节点运行。首先为您要对其运行作业的节点添加标签。

以下流程为节点添加标签。

步骤

1. 列出集群中的节点，及其标签：

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

输出结果与此相似（在此处显示在表中）：

名称	Status	角色	年龄	Version	标签
worker0	Ready	<none>	1d	v1.13.0	**…,kubernetes.io/hostname=worker0**
worker1	Ready	<none>	1d	v1.13.0	**…,kubernetes.io/hostname=worker1**

名称	Status	角色	年龄	Version	标签
worker2	Ready	<none>	1d	v1.13.0	**… ,kubernetes.io/hostname=worker2**

2. 选择其中一个节点，并使用以下命令向其添加标签：

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

例如：

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

其中 **<your-node-name>** 是您选择的节点的名称。

3. 验证您选择的节点是否具有 **disktype=ssd** 标签：

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

4. 输出结果与此相似（在此处显示在表中）：

名称	Status	角色	年龄	Version	标签
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**

您可以看到 **worker0** 节点现在有一个 **disktype=ssd** 标签。

5. 在自动化控制器 UI 中，在容器组中自定义 pod 规格的 metadata 部分中指定该标签。

```
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
```

额外设置

使用 **extra_settings** 时，您可以使用 awx-operator 传递许多自定义设置。参数 **extra_settings** 被附加到 **/etc/tower/settings.py**，并可替代 **extra_volumes** 参数。

名称	描述	default
extra_settings	额外设置	''

extra_settings 参数配置示例

```
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. 自定义 POD 超时

在将 pod 提交到 Kubernetes API 之前，自动化控制器中的容器组作业会过渡到 **running** 状态。然后，自动化控制器需要 pod 在 **AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT** 秒前进入 **Running** 状态。如果您希望自动化控制器在取消无法进入 **Running** 状态的作业前，您可以将 **AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT** 设置为更高的值。**AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT** 是自动化控制器在 pod 中创建到 Ansible 工作之前等待的时间。如果因为资源限制而无法调度 pod，您也可以延长时间。您可以在自动化控制器规格中使用 **extra_settings** 完成此操作。默认值为 2 小时。

如果您始终启动更多作业，超过 Kubernetes 可调度的作业，并且作业花费时间超过 *pending* 中的 **AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT**。

在控制容量可用前，才会启动作业。如果启动多个作业，超过容器组有能力运行，请考虑扩展 Kubernetes worker 节点。

3.4. 在 WORKER 节点上调度的作业

自动化控制器和 Kubernetes 在调度作业时扮演一个角色。

启动作业时，其依赖项就会实现，这意味着作业模板、项目和清单设置的要求由自动化控制器启动任何项目更新或清单更新。

如果自动化控制器中的其他业务逻辑没有阻断作业，则 control plane 中有控制容量来启动作业，则会将作业提交到分配程序。控制作业的"成本"的默认设置是 1 个 *容量*。因此，具有 100 个容量的控制 pod 一次只能控制 100 个作业。给定控制容量，作业将从 *pending* 过渡到 *waiting*。

分配程序（即控制计划 pod 中的后台进程）启动一个 worker 进程来运行作业。这通过与容器组关联的服务帐户与 Kubernetes API 通信，并使用自动化控制器中的 Container Group 中定义的 pod 规格来置备 pod。自动化控制器中的作业状态显示为 *running*。

Kubernetes 现在调度 pod。pod 可以保持 **AWX_CONTAINER_GROUP_POD_PENDING_TIMEOUT** 的 *pending* 状态。如果容器集因为 **ResourceQuota** 被拒绝，作业将从 *pending* 开始。您可以在命名空间中配置资源配额，以限制命名空间中的 pod 可消耗的资源数量。有关 ResourceQuota 的更多信息，请参阅资源配额。

第 4 章 在 OPENSHIFT CONTAINER PLATFORM 中配置 ANSIBLE 自动化控制器

在 Kubernetes 升级过程中，自动化控制器必须正在运行。

4.1. 最小化 OPENSHIFT CONTAINER PLATFORM 升级过程中的停机时间

在自动化控制器中进行以下更改，以便尽可能减少升级过程中的停机时间。

先决条件

- Ansible Automation Platform 2.4
- Ansible 自动化控制器 4.4
- OpenShift Container Platform
  - > 4.10.42
  - > 4.11.16
  - > 4.12.0
- Postgres 的高可用性(HA)部署
- 可以调度自动化控制器 pod 的多个 worker 节点

步骤

1. 在 AutomationController 规格中启用 **RECEPTOR_KUBE_SUPPORT_RECONNECT** ：

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

2. 在 AutomationController 规格中启用安全终止功能：

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

3. 为 Web 和任务配置 **podAntiAffinity**，将部署分散到 AutomationController 规格中：

```
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. 在 OpenShift Container Platform 中配置 **PodDisruptionBudget** ：

```
---
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
```