ALIBABA CLOUD

阿里云

弹性容器实例

容器配置

文档版本：20210223

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.设置容器启动命令和参数

本文介绍如何为容器设置启动时要执行的命令和参数。

背景信息

ECI实例通过容器镜像中的预设参数来启动容器。如果您想覆盖镜像的启动默认值，可以使用以下参数进行配置：

- 工作目录（WorkingDir）。

  镜像构建时，您可以使用WORKDIR来指定容器的一个目录， 容器启动时执行的命令则会在该目录下执行。更多信息，请参见WORKDIR。

  通过配置ECI实例的工作目录（ WorkingDir），可以覆盖WORKDIR。

  > 说明
  > - 如果镜像里未指定WORKDIR，且创建ECI实例也未配置工作目录，则工作目录默认为根目录。
  > - 如果指定的工作目录不存在，系统将自动创建。

- 启动命令（Command）和启动参数（Arg）

  镜像构建时，您可以使用ENTRYPOINT和CMD来指定启动容器后要执行的命令。更多信息，请参见ENTRYPOINT和CMD。

  通过配置ECI实例的启动命令（Command）和启动参数（Arg），可以覆盖ENTRYPOINT和CMD。具体生效规则如下：

镜像ENTRYPOINT	镜像CMD	容器Command	容器Arg	最终执行	说明
[mkdir]	[/data/backup]	未设置	未设置	[mkdir /data/backup]	Commands和Args均未设置，则使用镜像默认的配置。
[mkdir]	[/data/backup]	[cd]	未设置	[cd]	设置了Commands，未设置Args，则忽略镜像CMD，只执行容器Commands。
[mkdir]	[/data/backup]	未设置	[/opt/backup]	[mkdir /opt/backup]	设置了Args，未设置Commands，则执行镜像ENTRYPOINT和容器Args。
[mkdir]	[/data/backup]	[cd]	[/opt/backup]	[cd /opt/backup]	同时设置了Commands和Args，则执行容器Commands和Args。

> 注意
>
> 启动命令必须为容器镜像支持的命令，否则会导致容器启动失败。

Kubernetes方式

您可以设置comdand和arg字段来指定启动命令和启动参数。yaml示例如下：

更多信息，请参见为容器设置启动时要执行的命令和参数。

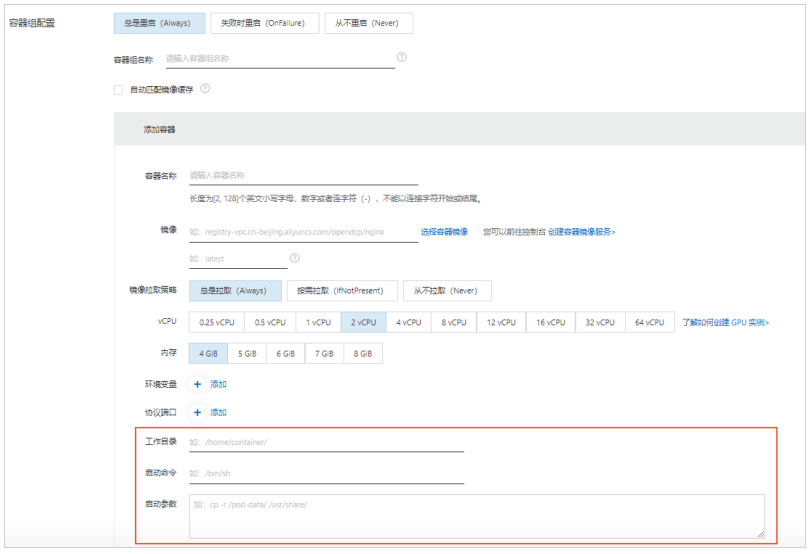
OpenAPI方式

调用CreateContainerGroup接口创建ECI实例时，您可以设置容器相关参数来指定工作目录、启动命令和启动参数。相关参数说明如下表所示。更多信息，请参见CreateContainerGroup。

名称	类型	是否必选	示例值	描述
Container.N.WorkingDir	String	否	/usr/local/	容器工作目录。
Container.N.Command.N	RepeatList	否	sleep	容器启动命令。最多20个。
Container.N.Arg.N	RepeatList	否	100	容器启动参数。最多10个。

控制台方式

通过弹性容器实例控制台创建ECI实例时，您可以在**容器组配置**中添加容器，同时设置工作目录、启动命令和启动参数，如下图所示。



2.使用Downward API

本文介绍如何使用Downward API，将Pod信息呈现给运行中的容器。

背景信息

Kubernetes Downward API提供了以下两种方式，可以将Pod信息（包括Pod和容器字段）呈现给运行中的容器。

- 环境变量（Environment variables）

  用于单个变量，可以将Pod信息直接注入容器内部。

- Volume挂载（Volume Files）

  可以将Pod信息生成为文件，直接挂载到容器内部。

目前阿里云容器服务Kubernetes（ACK和ASK）和弹性容器实例（ECI），已经支持了Downward API的大部分常用字段，下文将为您介绍使用方式。

环境变量方式

您可以通过Downward API将Pod的名称、命名空间、IP等信息注入到容器的环境变量中。通过环境变量可以获得的值如下表所示。

参数	描述
metadata.name	Pod名称。
metadata.namespace	Pod命名空间。
metadata.uid	Pod的UID。
metadata.labels['<KEY>']	Pod的标签值。
metadata.annotations['<KEY>']	Pod的注解值。
spec.serviceAccountName	Pod服务账号名称。
spec.nodeName	节点名称。
status.podIP	节点IP。

Deployment示例如下：

```
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
```

```
metadata:
 name: vk-downward-env
 labels:
  app: nginx
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
    annotations:
      regionId: cn-beijing
      platform: Aliyun ECI
    labels:
      app: nginx
      env: test
  spec:
   containers:
   - name: nginx
    image: nginx
    env:
    - name: MY_metadata.name
     valueFrom:
      fieldRef:
       fieldPath: metadata.name
    - name: MY_metadata.namespace
     valueFrom:
      fieldRef:
       fieldPath: metadata.namespace
    - name: MY_metadata.uid
     valueFrom:
      fieldRef:
       fieldPath: metadata.uid
    - name: MY_metadata.labels
     valueFrom:
      fieldRef:
       fieldPath: metadata.labels['env']
    - name: MY_metadata.annotations
     valueFrom:
      fieldRef:
```

```
      fieldPath: metadata.annotations['regionId']
  - name: MY_status.podIP
    valueFrom:
      fieldRef:
        fieldPath: status.podIP
  - name: MY_spec.serviceAccountName
    valueFrom:
      fieldRef:
        fieldPath: spec.serviceAccountName
  - name: MY_spec.nodeName
    valueFrom:
      fieldRef:
        fieldPath: spec.nodeName
nodeName: virtual-kubelet  #ACK场景下指定nodeName调度Pod到ECI
```

登入容器查看环境变量，可以看到fieldRef已经生效。示例如下：

```
root@default-vk-downward-env:/# env
MY_spec.nodeName=virtual-kubelet
MY_spec.serviceAccountName=default
MY_metadata.annotations=cn-beijing
MY_metadata.namespace=default
MY_metadata.uid=f4881309-f3dd-11e9-bcf9-9efaf54dcfa7
MY_metadata.name=vk-downward-env
MY_metadata.labels=test
MY_status.podIP=192.168.6.245
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_SERVICE_PORT=6443
PWD=/
PKG_RELEASE=1~buster
HOME=/root
KUBERNETES_PORT_443_TCP=tcp://172.22.*.*:443
NJS_VERSION=0.3.5
TERM=xterm
SHLVL=1
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=172.22.*.*
KUBERNETES_SERVICE_HOST=192.168.*.*
KUBERNETES_PORT=tcp://172.22.*.*:443
KUBERNETES_PORT_443_TCP_PORT=443
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
NGINX_VERSION=1.17.4
_=/usr/bin/env
```

Volume挂载方式

您可以通过Downward API将Pod的Label、Annotation等信息通过Volume挂载到容器的某个文件中。通过Volume挂载可以获得的值如下表所示。

参数	描述
metadata.name	Pod名称。
metadata.namespace	Pod命名空间。
metadata.uid	Pod的UID。

参数	描述
metadata.labels['<KEY>']	Pod的标签值。
metadata.annotations['<KEY>']	Pod的注解值。
metadata.labels	Podd的所有标签。
metadata.annotations	Pod的所有注解。

Deployment示例如下：

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
 name: vk-downward-down-volume
 labels:
  app: nginx
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
    annotations:
      regionId: cn-beijing
      platform: Aliyun ECI
    labels:
      app: nginx
      env: test
  spec:
   containers:
   - name: nginx
    image: nginx
    volumeMounts:
    - name: podinfo
     mountPath: /etc/podinfo
     readOnly: false
   volumes:
```

```
  volumes:
  - name: podinfo
    downwardAPI:
      items:
        - path: "metadata.name"
          fieldRef:
            fieldPath: metadata.name
        - path: "metadata.namespace"
          fieldRef:
            fieldPath: metadata.namespace
        - path: "metadata.uid"
          fieldRef:
            fieldPath: metadata.uid
        - path: "metadata.labels"
          fieldRef:
            fieldPath: metadata.labels
        - path: "metadata.annotations"
          fieldRef:
            fieldPath: metadata.annotations
  nodeName: virtual-node-eci-0
```

登入容器查看volume的挂载目录，可以看到volume的fieldRef已经生效，并存储在容器指定的目录下。示例如下：

```
Welcome to Alibaba Cloud Elastic Container Instance!
root@default-vk-downward-down-volume:/# cd /etc/podinfo/
root@default-vk-downward-down-volume:/etc/podinfo# ls
metadata.annotations  metadata.labels  metadata.name  metadata.namespace  metadata.uid
root@default-vk-downward-down-volume:/etc/podinfo# cat metadata.namespace
default
root@default-vk-downward-down-volume:/etc/podinfo# cat metadata.name
vk-downward-down-volume
root@default-vk-downward-down-volume:/etc/podinfo# cat metadata.uid
fa50b2b2-f3e3-11e9-bcf9-9efaf54dcfa7
root@default-vk-downward-down-volume:/etc/podinfo# cat metadata.annotations
platform="Aliyun ECI"
regionId="cn-beijing"
root@default-vk-downward-down-volume:/etc/podinfo# cat metadata.labels
app="nginx"
env="test"
root@default-vk-downward-down-volume:/etc/podinfo#
```

目前仅支持Pod字段，还不支持容器字段，例如：

```
limits.cpu
requests.cpu
limits.memory
requests.memory
limits.ephemeral-storage
requests.ephemeral-storage
```

3.Security Context

Security Context，即安全上下文，可以用于定义Pod或Container的权限和访问控制。

背景信息

Security Context，即安全上下文，可以用于定义Pod或Container的权限和访问控制。

ECI默认内核参数

在 Linux 中，管理员可以通过 sysctl 接口修改内核运行时的参数。ECI中的内核参数通过以下方式查看：

```
sysctl -a
```

也可以通过该链接查看ECI的默认内核参数：Virtual-Kubelet-Example

安全策略

阿里云ACK和ASK默认启用了准入控制器插件PodSecurityPolicy，即如果Pod中设置了SecurityContext相关内容，将接受准入控制器对Pod的权限检查，如果SecurityContext中设置了超出指定安全策略所允许的范围，K8s集群将拒绝创建该Pod。更多信息，请参见PodSecurityPolicy。

阿里云ACK和ASK使用名为ack.privileged的PodSecurityPolicy对象作为默认的安全策略，您可以通过以下命令查看该安全策略允许的权限范围。

```
kubectl get psp ack.privileged -o yaml
```

创建安全策略

下面创建一个包含sysctl权限的PodSecurityPolicy对象，并通过RBAC绑定到服务账户sa-sysctl：

```
# 创建一个的服务账号
apiVersion: v1
kind: ServiceAccount
metadata:
 name: sa-sysctl
---
# 创建一个psp对象，该psp允许创建含有spec.securityContext.sysctls选项的Pod
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
 name: sysctl
spec:
 allowedUnsafeSysctls: # 填入允许Pod使用的sysctl参数
 - 'kernel.msgmax'
 - 'net.*'
 seLinux:
  rule: RunAsAny
```

```
 supplementalGroups:
  rule: RunAsAny
 runAsUser:
  rule: RunAsAny
 fsGroup:
  rule: RunAsAny
 volumes:
 - '*'
---
# 创建一个角色并引用上面的psp
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: psp:sysctl
rules:
- apiGroups: ['policy']
 resources: ['podsecuritypolicies']
 verbs:    ['use']
 resourceNames: ['sysctl']
---
# 将角色绑定到服务账号，此时该服务账号创建的所有Pod都使用上面的psp作为安全策略
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: sa-sysctl:sysctl
roleRef:
 kind: ClusterRole
 name: psp:sysctl
 apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
 name: sa-sysctl
```

Pod sysctl

接下来使用上面的服务账号来创建一个含有spec.securityContext.sysctls选项的Pod：

```
apiVersion: v1
kind: Pod
metadata:
 name: sysctl-example
spec:
 serviceAccountName: sa-sysctl
 securityContext:
  sysctls:
  - name: net.core.somaxconn
    value: "1024"
  - name: kernel.msgmax
    value: "65536"
 containers:
 - name: busybox
  image: busybox
  command: [ "sh", "-c", "sleep 12000" ]
```

但是，并不是所有的 sysctl 值都可以支持通过pod来修改，目前ECI支持Pod修改的 sysctl 以及限制如下：

支持大类	排除项
kernel.shm.*全部支持	排除kernel.shm_rmid_forced 。
kernel.msg.*全部支持	
kernel.sem 支持	
fs.mqueue.*全部支持	
net.*全部支持	排除net.ipv4.ip_local_port_range和net.ipv4.tcp_syncookies 。

Container SecurityContext

除了可以设置Pod维度的 securityContext，还可以设置容器维度的 securityContext，即只对容器组内的单个容器生效。

目前容器纬度的 `securityContext` 支持的字段如下：

支持的参数	说明
runAsUser	会覆盖 Dockerfile 中的 `USER` 指令。
capabilities	只能 ADD ["NET_ADMIN"]，支持["NET_RAW"]需要白名单。

不支持修改的参数的默认值如下：

不支持的参数	默认值
privileged	false
procMount	DefaultProcMount
readOnlyRootFilesystem	true

实际演示

在默认情况下，如果在容器内进行网络相关的操作，会得到如下的报错。

因为在Docker的容器中对权限的限制是非常严格的，包括NET_ADMIN等在内很多权限默认都没有。

```
/ $ ip route list
default via 192.168.127.253 dev eth0  src 192.168.85.229  metric 1024
192.168.64.0/18 dev eth0 scope link  src 192.168.85.229
192.168.127.253 dev eth0 scope link  src 192.168.85.229  metric 1024
/ $ ip route delete 192.168.64.0/18
ip: RTNETLINK answers: Operation not permitted
```

给容器增加NET_ADMIN，支持网络管理：

```
apiVersion: v1
kind: Pod
metadata:
 name: sysctl-example
spec:
 containers:
 - name: busybox
  image: busybox
  command: [ "sh", "-c", "sleep 12000" ]
  securityContext:
   runAsUser: 2000
   capabilities:
    add: ["NET_ADMIN"]
```

尝试删除并增加一条路由：

```
/ $ ip route list
default via 192.168.127.253 dev eth0  src 192.168.85.230  metric 1024
192.168.64.0/18 dev eth0 scope link  src 192.168.85.230
192.168.127.253 dev eth0 scope link  src 192.168.85.230  metric 1024
/ $ ip route delete 192.168.64.0/18
/ $ ip route list
default via 192.168.127.253 dev eth0  src 192.168.85.230  metric 1024
192.168.127.253 dev eth0 scope link  src 192.168.85.230  metric 1024
/ $ ip route add 192.168.64.0/18 dev eth0 scope link  src 192.168.85.230
/ $ ip route list
default via 192.168.127.253 dev eth0  src 192.168.85.230  metric 1024
192.168.64.0/18 dev eth0 scope link  src 192.168.85.230
192.168.127.253 dev eth0 scope link  src 192.168.85.230  metric 1024
```

查看容器内进程的user都为2000，也生效了。

```
/ $ ps
PID   USER     TIME COMMAND
    1 2000     0:00 sleep 12000
   13 2000     0:00 /bin/sh
   30 2000     0:00 /bin/sh
   37 2000     0:00 ps
```

样例地址请参见：eci-securitycontext。

4.获取ECI实例元数据

本文介绍ECI实例元数据，以及如何在实例内部获取元数据。

获取实例元数据

实例元数据包含ECI实例在阿里云系统中的基本信息，例如实例ID、IP地址、所属地域、交换机ID等。您可以使用以下方式获取ECI实例元数据。

1. 连接容器。具体操作，请参见调试ECI实例。
2. 执行以下命令访问元数据的根目录。

```
curl http://100.100.100.200/latest/meta-data/
```

3. 在URL中添加具体的元数据名称即可获取具体的元数据。

   例如：执行以下命令获取实例ID。

```
curl http://100.100.100.200/latest/mate-data/instance-id
```

实例元数据列表

ECI实例目前能获取的基本实例元数据项如下表所示。

实例元数据项	说明
/dns-conf/nameservers	实例的DNS配置。
/eipv4	实例的弹性公网IP（IPv4类型）。
/hostname	实例的主机名，对应ContainerGroupName。
/instance-id	实例ID。
/mac	实例的MAC地址。
/network/interfaces/	网卡的MAC地址列表。
/network/interfaces/macs/[mac]/network-interface-id	网卡的标识ID，其中[mac]参数需要替换为实例MAC地址。
/network/interfaces/macs/[mac]/netmask	网卡对应的子网掩码。
/network/interfaces/macs/[mac]/vswitch-cidr-block	网卡所属的虚拟交换机IPv4 CIDR段。

实例元数据项	说明
/network/interfaces/macs/[mac]/vpc-cidr-block	网卡所属的VPC IPv4 CIDR段。
/network/interfaces/macs/[mac]/private-ipv4s	网卡分配的私网IPv4地址列表。
/network/interfaces/macs/[mac]/vpc-ipv6-cidr-blocks	网卡所属的VPC IPv6 CIDR段，仅支持已配置了IPv6的VPC类型实例。
/network/interfaces/macs/[mac]/vswitch-id	网卡所属安全组的虚拟交换机ID。
/network/interfaces/macs/[mac]/vpc-id	网卡所属安全组的VPC ID。
/network/interfaces/macs/[mac]/primary-ip-address	网卡主私有IP地址。
/network/interfaces/macs/[mac]/gateway	网卡对应的IPv4网关地址。
/instance/max-netbw-egress	实例规格的出方向内网最大带宽。单位：Kbit/s。
/instance/max-netbw-ingerss	实例规格的入方向内网最大带宽。单位：Kbit/s。
/network/interfaces/macs/[mac]/ipv6s	网卡分配的IPv6地址列表，仅支持已配置了IPv6的VPC类型实例。
/network/interfaces/macs/[mac]/ipv6-gateway	网卡所属的VPC的IPv6网关地址。
/network/interfaces/macs/[mac]/vswitch-ipv6-cidr-block	网卡所属的虚拟交换机IPv6 CIDR段，仅支持已配置了IPv6的VPC类型实例。
/private-ipv4	实例的私网IPv4地址。
/ntp-conf/ntp-servers	NTP服务器地址。
/owner-account-id	实例拥有者的阿里云账号ID。
/region-id	实例所属地域。

实例元数据项	说明
/serial-number	实例所对应的序列号。
/vpc-id	实例所属VPC ID。
/vpc-cidr-block	实例所属VPC的CIDR网段。
/vswitch-cidr-block	实例所属虚拟交换机的CIDR网段。
/vswitch-id	实例所属虚拟交换机ID。
/zone-id	实例所属可用区。
/ram/security-credentials/[role-name]	实例RAM角色策略所生成的STS临时凭证。只有在实例指定了RAM角色后，您才能获取STS临时凭证。其中[role-name]参数需要替换为实例RAM角色的名称。如果未指定[role-name]，将返回实例RAM角色名称。

5.查看Core dump文件

本文介绍如何设置Core dump文件的保存目录，以便在容器异常终止时查看分析Core dump文件，找出问题原因。

背景信息

在Linux中，如果程序突然异常终止或者崩溃时，操作系统会将程序当时的内存状态记录下来，保存在一个文件中，这种行为就叫做Core dump。此时，您可以查看分析Core dump文件，找出问题原因。

Linux中支持Core dump（Action为Core）的Signal如下图所示。

Signal	Standard	Action	Comment
SIGABRT	P1990	Core	Abort signal from abort(3)
SIGALRM	P1990	Term	Timer signal from alarm(2)
SIGBUS	P2001	Core	Bus error (bad memory access)
SIGCHLD	P1990	Ign	Child stopped or terminated
SIGCLD	-	Ign	A synonym for SIGCHLD
SIGCONT	P1990	Cont	Continue if stopped
SIGEMT	-	Term	Emulator trap
SIGFPE	P1990	Core	Floating-point exception
SIGHUP	P1990	Term	Hangup detected on controlling terminal or death of controlling process
SIGILL	P1990	Core	Illegal Instruction
SIGINFO	-		A synonym for SIGPWR
SIGINT	P1990	Term	Interrupt from keyboard
SIGIO	-	Term	I/O now possible (4.2BSD)
SIGIOT	-	Core	IOT trap. A synonym for SIGABRT
SIGKILL	P1990	Term	Kill signal
SIGLOST	-	Term	File lock lost (unused)
SIGPIPE	P1990	Term	Broken pipe: write to pipe with no readers; see pipe(7)
SIGPOLL	P2001	Term	Pollable event (Sys V). Synonym for SIGIO
SIGPROF	P2001	Term	Profiling timer expired
SIGPWR	-	Term	Power failure (System V)
SIGQUIT	P1990	Core	Quit from keyboard
SIGSEGV	P1990	Core	Invalid memory reference
SIGSTKFLT	-	Term	Stack fault on coprocessor (unused)
SIGSTOP	P1990	Stop	Stop process
SIGTSTP	P1990	Stop	Stop typed at terminal
SIGSYS	P2001	Core	Bad system call (SVr4); see also seccomp(2)
SIGTERM	P1990	Term	Termination signal
SIGTRAP	P2001	Core	Trace/breakpoint trap
SIGTTIN	P1990	Stop	Terminal input for background process
SIGTTOU	P1990	Stop	Terminal output for background process
SIGUNUSED	-	Core	Synonymous with SIGSYS
SIGURG	P2001	Ign	Urgent condition on socket (4.2BSD)
SIGUSR1	P1990	Term	User-defined signal 1
SIGUSR2	P1990	Term	User-defined signal 2
SIGVTALRM	P2001	Term	Virtual alarm clock (4.2BSD)
SIGXCPU	P2001	Core	CPU time limit exceeded (4.2BSD); see setrlimit(2)
SIGXFSZ	P2001	Core	File size limit exceeded (4.2BSD); see setrlimit(2)

更多信息，请参见Core dump file。

功能概述

对于ECI来说，因为VM是托管的，所以无法指定Core dump的文件名和文件目录。即使容器进程异常退出产生了Core dump文件，也无法离线查看，进行后续的分析（容器退出后，Core dump文件将随之丢失）。

默认情况下，Core dump文件的名称为core.pid，保存在当前目录下。

```
/ # cd /pod/
/pod # ls
data
/pod # sleep 10
^\Quit (Core dumped)（按Ctrl+\触发）
/pod # ls
core.45  data
```

针对上述情况，ECI支持自定义设置Core dump文件的保存目录。您可以将Core dump文件保存到外挂存储中，即时容器退出了，也可以进行离线查看和分析。

配置方式如下：

- Kubernetes

```
annotations:
  k8s.aliyun.com/eci-core-pattern: "/xx/xx/core"
```

- OpenAPI

```
CorePattern = "/xx/xx/core"
```

配置示例

对于Core dump文件，一般是为了离线分析，因此在设置Core dump文件的保存目录时，一般会选择外挂存储，而不是保存在容器本地目录。目前支持的外挂存储包括云盘，NFS等。下文以NFS为例进行介绍。

1. 挂载NFS，并设置Core dump文件的保存目录。

   如下示例，将NFS挂载到容器的/pod/data/dump/目录，然后设置Core dump文件的保存目录为：/pod/data/dump/core。此时，容器的所有Core dump文件将自动保存到该目录下，并同步到NFS。即时该容器释放了，您仍可以从NFS中获取Core dump文件进行分析。

   ```
   'Volume.1.Name': 'default-volume1',
   'Volume.1.Type': 'NFSVolume',
   'Volume.1.NFSVolume.Path': '/dump/',
   'Volume.1.NFSVolume.Server': '143b24****-gfn3.cn-beijing.nas.aliyuncs.com',

   'Container.1.VolumeMount.1.Name': 'default-volume1',
   'Container.1.VolumeMount.1.MountPath': '/pod/data/dump/',

   'CorePattern':'/pod/data/dump/core-eci',
   ```

2. 在容器任意的目录下触发Core dump。

   如下示例，可以看到设置的Core dump文件的名称和保存目录已经生效。

   ```
   / # ls
   bin  dev  etc  home pod  proc root sys  tmp  usr  var
   / # sleep 10
   ^\Quit (core dumped)（按Ctrl+\触发）
   / # ls
   bin  dev  etc  home pod  proc root sys  tmp  usr  var
   / # cd /pod/data/dump/
   /pod/data/dump # ls
   core-eci.12
   ```

3. 释放该台ECI实例。
4. 将NFS挂载到新的一台ECI实例上，然后登录实例查看Core dump文件。

   如下示例，可以看到Core dump文件并没有丢失，您可以进行查看分析。

```
/ # cd /pod/data/dump/
/pod/data/dump # ls
core-eci.12
```

6.为Pod配置NTP服务

本文主要介绍如何为运行在virtual kubelet上的Pod配置NTP服务。当您在部署应用时，如果需要Pod内的容器能与NTP服务进行时间同步，您可以参考本文进行配置。

前提条件

已将virtual-kubelet升级到最新版本。

背景信息

不同类型的kubernetes集群升级virtual kubelet到最新版本的方式如下：

- Serverless kubernetes：由管理员统一负责升级。
- 托管版kubernetes：您需要自行升级。
- 专有版kubernetes：您需要自行升级。
- 自建kubernetes：您需要自行升级。

操作步骤

您需要在Pod的annotations中增加 `k8s.aliyun.com/eci-ntp-server` 注解，设置需要配置的NTP服务的IP地址。

1. 创建配置NTP服务的yaml文件。

   ```
   vim set-ntp-pod.yaml
   ```

   以下为yaml文件的内容示例：

   ```
   apiVersion: v1
   kind: Pod
   metadata:
    annotations:
     k8s.aliyun.com/eci-ntp-server: 10.10.5.1 # 设置您的NTP服务的IP地址
    name: set-custom-ntp
   spec:
    nodeName: virtual-kubelet
    containers:
     - image: centos:latest
       command:
        - sleep
        - "3600"
       imagePullPolicy: IfNotPresent
       name: centos
   ```

2. 将yaml文件中的配置应用到Pod。

```
kubectl apply -f set-ntp-pod.yaml
```

验证结果

登录到容器，验证NTP服务是否设置成功。

1. 获取Pod信息。

   ```
   kubectl get pod/set-custom-ntp
   ```

   返回示例如下：

   ```
   NAME          READY  STATUS   RESTARTS  AGE
   set-custom-ntp  1/1     Running  0         7m20s
   ```

2. 进入容器。

   ```
   kubectl exec set-custom-ntp -it -- bash
   ```

3. 查询容器的时间来源。

   ```
   chronyc sources
   ```

   如果返回了NTP服务的IP地址，则表示设置成功。返回示例如下：

   ```
   210 Number of sources = 1
   MS Name/IP address      Stratum Poll Reach LastRx Last                sample
   ===============================================================================
   ^* 10.10.5.1              2      6   377    35    +40us[ +135us] +/-  14ms
   ```

7.为Pod配置时区

本文主要介绍如何为运行在virtual kubelet上的Pod配置不同的时区，当您使用Pod部署应用时，如果需要Pod能指定不同地点的时区，您可以参考此文档进行配置。

前提条件

已将virtual-kubelet升级到最新版本。

背景信息

不同类型的kubernetes集群升级virtual kubelet到最新版本的方式如下：

- Serverless kubernetes：由管理员统一负责升级。
- 托管版kubernetes：您需要自行升级。
- 专有版kubernetes：您需要自行升级。
- 自建kubernetes：您需要自行升级。

操作步骤

1. 创建一个configmap，导入您需要制定的时区。

   其他时区请使用/usr/share/zoneinfo/Asia/目录下对应的文件，以下为示例：

   ```
   kubectl create configmap tz --from-file=/usr/share/zoneinfo/Asia/Shanghai
   ```

2. 创建配置时区的yaml文件。

   ```
   vim set-timezone.yaml
   ```

   将configmap挂载到/etc/localtime/Shanghai目录下，以下为yaml文件示例：

```
apiVersion: v1
kind: Pod
metadata:
 name: timezone
spec:
 containers:
 - name: timezone
  image: registry-vpc.cn-beijing.aliyuncs.com/eci_open/busybox:1.30
  command: [ "sleep", "10000" ]
  volumeMounts:
   - name: tz
    mountPath: /etc/localtime
    subPath: Shanghai
 volumes:
  - name: tz
   configMap:
    name: tz
 nodeSelector:
  type: virtual-kubelet
 tolerations:
 - key: virtual-kubelet.io/provider
  operator: Exists
```

3. 将yaml文件中的配置应用到Pod。

```
kubectl apply -f set-timezone.yaml
```

验证结果

登录到容器，验证时区是否设置成功。

1. 获取Pod信息。

```
kubectl get pod/timezone
```

返回示例如下：

```
NAME          READY  STATUS   RESTARTS  AGE
set-timezone  1/1    Running  0         7m20s
```

2. 进入容器。

```
kubectl exec timezone -it -- sh
```

3. 查询容器的时区。

```
date -R
```

如果返回的时间与设置的时区信息对应，则表示设置成功。返回示例如下：

```
Fri, 01 May 2020 10:00:11 +0800
```