ALIBABA CLOUD

阿里云

弹性容器实例

容器配置

文档版本：20220317

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

ECI实例通过容器镜像中的预设参数来启动容器。如果您在构建镜像时没有设置启动命令和参数，或者想要变更启动命令和参数，可以在创建ECI实例时设置。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

如果您想覆盖镜像中设置的启动默认值，包括工作目录、启动命令和参数，可以通过以下参数进行配置：

- 工作目录

  镜像构建时，通过WORKDIR可以指定容器的工作目录， 容器启动时执行的命令会在该目录下执行。更多信息，请参见WORKDIR。

  创建ECI实例时，通过配置ECI实例中容器的工作目录（ 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。

- 启动命令和参数

  镜像构建时，通过ENTRYPOINT和CMD可以指定启动容器后要执行的命令和参数。更多信息，请参见ENTRYPOINT和CMD。

  创建ECI实例时，通过配置ECI实例中容器的启动命令（Command）和参数（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]	Command和Arg均未设置，则执行镜像ENTRYPOINT和CMD。
[mkdir]	[/data/backup]	[cd]	未设置	[cd]	设置了Command，未设置Arg，则只执行Command，忽略镜像ENTRYPOINT和CMD。
[mkdir]	[/data/backup]	未设置	[/opt/backup]	[mkdir /opt/backup]	设置了Arg，未设置Command，则执行镜像ENTRYPOINT和容器Arg。
[mkdir]	[/data/backup]	[cd]	[/opt/backup]	[cd /opt/backup]	同时设置了Command和Arg，则执行容器Command和Arg。

> **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

使用Kubernetes方式创建ECI实例时，您可以通过容器的workingDir、command和args字段来设置工作目录、启动命令和参数。示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
spec:
  containers:
  - name: busybox
    image: busybox
    workingDir: /work
    command: ["printenv"]
    args: ["HOSTNAME", "KUBERNETES_PORT"]
  restartPolicy: OnFailure
```

更多信息，请参见为容器设置启动时要执行的命令和参数。

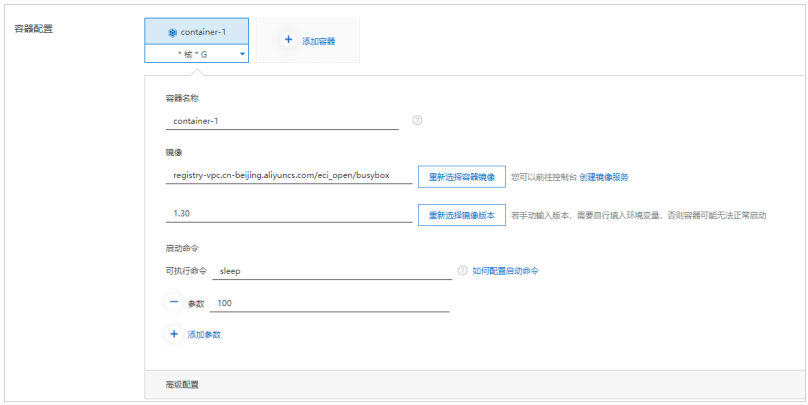
OpenAPI方式

调用CreateContainerGroup接口创建ECI实例时，您可以通过容器中的WorkingDir、Command和Arg参数来设置工作目录、启动命令和参数。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	容器启动命令对应的参数。最多10个。

控制台方式

通过弹性容器实例售卖页创建ECI实例时，您可以在**容器配置**中设置各个容器的启动命令和参数，如下图所示。


容器配置
container-1
添加容器
容器名称
container-1
镜像
registry-vpc.cn-beijing.aliyuncs.com/eci_open/busybox
重新选择容器镜像
您可以前往控制台 创建镜像服务
1.30
重新选择镜像版本
若手动输入版本，需要自行填入环境变量，否则容器可能无法正常启动
启动命令
可执行命令
sleep
如何配置启动命令
参数
100
添加参数
高级配置

2.使用探针对容器进行健康检查

本文介绍如何配置应用存活探针（ Liveness Probe）和应用业务探针（Readiness Probe），对容器进行健康检查。

背景信息

Kubernetes中，容器的健康检查由kubelet定期执行，kubelet通过存活探针和业务探针来检查容器的状态和运行情况。

- 应用存活探针（ Liveness Probe）

  用于检查容器是否正常运行。如果检查成功，则表示容器正常运行。如果检查失败，系统会根据配置的容器重启策略进行相应的处理。如果未配置该探针，则默认容器一直正常运行。

  应用存活探针可以应用于以下场景：

  - 当应用程序处于运行状态但无法进行进一步操作时，Liveness Probe将捕获到deadlock，重启对应的容器，使得应用程序在存在bug的情况下依然能够运行。
  - 长时间运行的应用程序最终可能会转换到broken状态，此时除了重新启动，无法恢复。Liveness Probe可以检测并补救这种情况。

- 应用业务探针（Readiness Probe）

  用于检查容器是否已经就绪，可以为请求提供服务。如果检查成功，则表示容器已经准备就绪，可以接收业务请求。如果检查失败，则表示容器没有准备就绪，系统将停止向该容器发送任何请求，直至重新检查成功。

  应用业务探针可以应用于以下场景：

  如果应用程序暂时无法对外部流量提供服务，例如应用程序需要在启动期间加载大量数据或配置文件，此时，如果不想终止应用程序，也不想向其发送请求，可以通过Readiness Probe来检测和缓解这种情况。

Kubernetes方式

使用Kubernetes方式创建ECI实例时，您可以通过容器的livenesProbe和readinessProbe字段来设置Liveness Probe或者Readiness Probe。示例如下：

- 设置Liveness Probe

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: busybox:latest
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    #设置Liveness Probe，通过命令行方式进行检查
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5     #容器启动5秒后开始检查
      periodSeconds: 5      #每5秒执行一次检查
```

- 设置Readiness Probe

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: readiness
  name: readiness-exec
spec:
  containers:
  - name: readiness
    image: busybox:latest
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    #设置Readiness Probe，通过命令行方式进行检查
    readinessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5     #容器启动5秒后开始检查
      periodSeconds: 5      #每5秒执行一次检查
```

更多信息，请参见配置存活、就绪和启动探测器。

OpenAPI方式

调用CreateContainerGroup接口创建ECI实例时，您可以通过容器中的LivenessProbe和ReadinessProbe参数来设置应用存活探针和应用业务探针。相关参数说明如下表所示。更多信息，请参见CreateContainerGroup。

- LivenessProbe相关参数

名称	类型	是否必选	示例值	描述
Container.N.LivenessProbe.HttpGet.Path	String	否	/healthyz	使用HTTP请求方式进行健康检查时，HTTP Get请求检测的路径。
Container.N.LivenessProbe.HttpGet.Port	Integer	否	8888	使用HTTP请求方式进行健康检查时，HTTP Get请求检测的端口号。
Container.N.LivenessProbe.HttpGet.Scheme	String	否	HTTP	使用HTTP请求方式进行健康检查时，HTTP Get请求对应的协议类型，取值范围： • HTTP • HTTPS
Container.N.LivenessProbe.InitialDelaySeconds	Integer	否	5	检查开始执行的时间，以容器启动完成为起点计算。
Container.N.LivenessProbe.PeriodSeconds	Integer	否	1	检查执行的周期，默认为10秒，最小为1秒。
Container.N.LivenessProbe.SuccessThreshold	Integer	否	1	从上次检查失败后重新认定检查成功的检查次数阈值（必须是连续成功），默认为1。当前必须为1。
Container.N.LivenessProbe.FailureThreshold	Integer	否	3	从上次检查成功后认定检查失败的检查次数阈值（必须是连续失败），默认为3。
Container.N.LivenessProbe.TimeoutSeconds	Integer	否	1	检查超时的时间，默认为1秒，最小为1秒。
Container.N.LivenessProbe.Exec.Command.N	RepeatList	否	cat /tmp/healthy	使用命令行方式进行健康检查时，在容器内执行的命令。
Container.N.LivenessProbe.TcpSocket.Port	Integer	否	8000	使用TCP Socket方式进行健康检查时，TCP Socket检测的端口。

- ReadinessProbe相关参数

名称	类型	是否必选	示例值	描述
Container.N.ReadinessProbe.HttpGet.Path	String	否	/healthyz	使用HTTP请求方式进行健康检查时，HTTP Get请求检测的路径。
Container.N.ReadinessProbe.HttpGet.Port	Integer	否	8888	使用HTTP请求方式进行健康检查时，HTTP Get请求检测的端口号。
Container.N.ReadinessProbe.HttpGet.Scheme	String	否	HTTP	使用HTTP请求方式进行健康检查时，HTTP Get请求对应的协议类型，取值范围： • HTTP • HTTPS
Container.N.ReadinessProbe.InitialDelaySeconds	Integer	否	5	检查开始执行的时间，以容器启动完成为起点计算。
Container.N.ReadinessProbe.PeriodSeconds	Integer	否	1	检查执行的周期，默认为10秒，最小为1秒。
Container.N.ReadinessProbe.SuccessThreshold	Integer	否	1	从上次检查失败后重新认定检查成功的检查次数阈值（必须是连续成功），默认为1。当前必须为1。
Container.N.ReadinessProbe.FailureThreshold	Integer	否	3	从上次检查成功后认定检查失败的检查次数阈值（必须是连续失败），默认为3。
Container.N.ReadinessProbe.TimeoutSeconds	Integer	否	1	检查超时的时间，默认为1秒，最小为1秒。
Container.N.ReadinessProbe.Exec.Command.N	RepeatList	否	cat /tmp/healthy	使用命令行方式进行健康检查时，在容器内执行的命令。
Container.N.ReadinessProbe.TcpSocket.Port	Integer	否	8000	使用TCP Socket方式进行健康检查时，TCP Socket检测的端口。

控制台方式

通过弹性容器实例售卖页创建ECI实例时，您可以在**容器配置**的**高级配置**中开启健康检查功能。配置操作如下图所示。

> **说明**
>
> 使用控制台配置时，仅支持命令行和HTTP请求两种检查方式，不支持TCP Socket方式。

相关参数说明如下表所示。

参数	描述
时间设置	包括延迟时间和超时时间。 • 延迟时间：容器启动多久后开始检查。 • 超时时间：检查的超时等待时间，如果超时，则视为检查失败。
检查方式	包括以下两种方式： • 命令行脚本：探针在容器内执行命令，并检查命令退出的状态码，如果状态码为0，则表示检查成功。 • HTTP请求方式：探针向容器发送HTTP请求，如果返回状态码符合以下要求：200≤状态码<400，则表示检查成功。
命令行脚本	当检查方式配置为命令行脚本时，需要配置在容器内执行的命令行脚本。
HTTP请求方式	当检查方式配置为HTTP请求方式时，需要配置HTTP Get请求包含的路径、端口和协议。

配置示例

以Nginx容器为例，创建一个设置了Liveness Probe和Readiness Probe的ECI实例，然后模拟服务异常，以查看探针的配置效果。

1. 使用Java SDK创建一个ECI实例。

   创建ECI实例时，容器镜像使用Nginx，设置Liveness Probe和Readiness Probe的代码示例如下：

```
//配置应用存活探针，在容器运行5秒后，每3秒检测一次80端口，超时时间为10秒，成功和失败的阈值均为3次
CreateContainerGroupRequest.Container.ContainerProbe livenessProbe = new CreateContainerGroupRequest.Container.ContainerProbe();
livenessProbe.setTcpSocketPort(80);
livenessProbe.setInitialDelaySeconds(5);
livenessProbe.setPeriodSeconds(3);
livenessProbe.setFailureThreshold(3);
livenessProbe.setSuccessThreshold(1);
livenessProbe.setTimeoutSeconds(10);
//配置应用业务探针，在容器运行5秒后，每3秒检测一次80端口，超时时间为10秒，成功阈值为1次，失败阈值为3次
CreateContainerGroupRequest.Container.ContainerProbe readinessProbe = new CreateContainerGroupRequest.Container.ContainerProbe();
readinessProbe.setTcpSocketPort(80);
readinessProbe.setInitialDelaySeconds(5);
readinessProbe.setPeriodSeconds(3);
readinessProbe.setFailureThreshold(3);
readinessProbe.setSuccessThreshold(3);
readinessProbe.setTimeoutSeconds(10);
```

2. 查看ECI实例创建成功的相关事件。

   ECI实例创建成功后，查看实例事件，可以看到实例正常启动。

事件名称	类型	描述	开始时间	结束时间
nigixtest.167323cd9a6b391f	Normal	Created container nginx	2021年4月6日10:27:37	2021年4月6日10:27:37
nigixtest.167323cd9f10000d	Normal	Started container nginx	2021年4月6日10:27:37	2021年4月6日10:27:37
nigixtest.167323cd8d711955	Normal	Successfully pulled image "nginx" in 4.365524944s	2021年4月6日10:27:36	2021年4月6日10:27:36
nigixtest.167323cc893c273c	Normal	Pulling image "nginx"	2021年4月6日10:27:32	2021年4月6日10:27:32

3. 修改Nginx监听端口，模拟服务异常。

   i. 修改Nginx监听端口。

```
vi /etc/nginx/conf.d/default.conf
```

   修改示例如下：

```
server {
    listen       8080;
    listen  [::]:8080;
    server_name  localhost;

    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.
    #
    #location ~ \.php$ {
-- INSERT --
```

ii. 重启Nginx。

```
nginx -s reload
```

4. 查看探针的生效情况。

   重启Nginx几秒后，容器会自动进行重启。此时查看实例事件，可以看到在Liveness Probe和Readiness Probe各有三次失败之后，容器进行了重启。

事件名称	类型	描述	开始时间	结束时间
nigixtest.167325b909548ca9	Normal	Container nginx failed liveness probe, will be restarted	2021年4月6日11:02:47	2021年4月6日11:02:47
nigixtest.167325b90de4acb1	Normal	Container image "nginx" already present on machine	2021年4月6日11:02:47	2021年4月6日11:02:47
nigixtest.167325b90f476404	Normal	Created container nginx	2021年4月6日11:02:47	2021年4月6日11:02:47
nigixtest.167325b9138d313e	Normal	Started container nginx	2021年4月6日11:02:47	2021年4月6日11:02:47
nigixtest.167325b8f5c3a91a	Warning	Liveness probe failed: dial tcp 192.168.11.23:80: connect: connection refused	2021年4月6日11:02:47	2021年4月6日11:02:47
nigixtest.167325b8fb78342c	Warning	Readiness probe failed: dial tcp 192.168.11.23:80: connect: connection refused	2021年4月6日11:02:47	2021年4月6日11:02:47
nigixtest.167325b842f30264	Warning	Liveness probe failed: dial tcp 192.168.11.23:80: connect: connection refused	2021年4月6日11:02:44	2021年4月6日11:02:44
nigixtest.167325b848a667fb	Warning	Readiness probe failed: dial tcp 192.168.11.23:80: connect: connection refused	2021年4月6日11:02:44	2021年4月6日11:02:44
nigixtest.167325b790231587	Warning	Liveness probe failed: dial tcp 192.168.11.23:80: connect: connection refused	2021年4月6日11:02:41	2021年4月6日11:02:41
nigixtest.167325b795cfc219	Warning	Readiness probe failed: dial tcp 192.168.11.23:80: connect: connection refused	2021年4月6日11:02:41	2021年4月6日11:02:41

3.在容器内获取元数据

本文介绍如何在容器内获取元数据。

目前ECI提供三种方式可以将Pod（ECI实例）信息和容器Meta数据呈现给运行中的容器。

- 方式一：通过MetaServer访问元数据
- 方式二：配置容器环境变量
- 方式三：Downward API

方式一：通过MetaServer访问元数据

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/meta-data/instance-id
```

   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。

实例元数据项	说明
/network/interfaces/macs/[mac]/vswitch-cidr-block	网卡所属的虚拟交换机IPv4 CIDR段。
/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。

实例元数据项	说明
/owner-account-id	实例拥有者的阿里云账号ID。
/region-id	实例所属地域。
/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。

方式二：配置容器环境变量

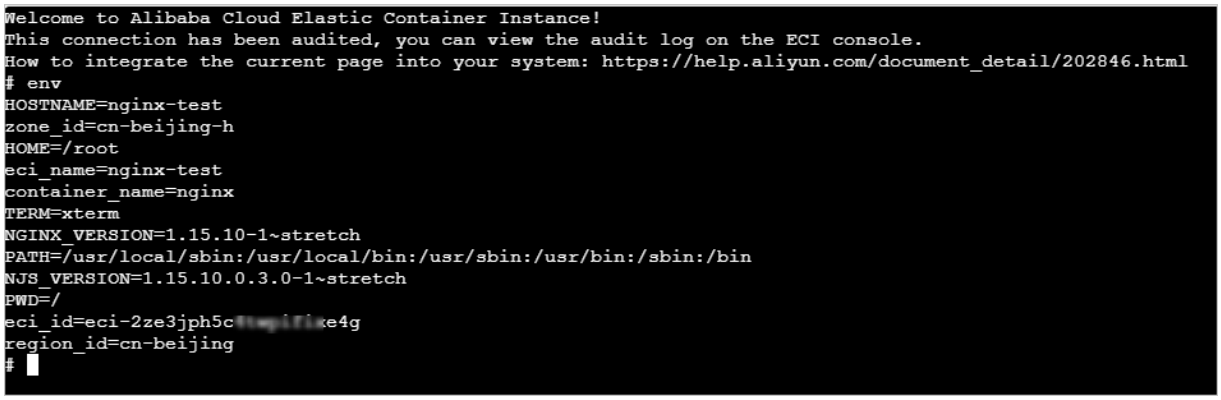
通过配置ECI实例容器环境变量的Value便可获取实例相关信息，ECI实例目前可以获取到实例ID、实例名称、实例所属地域、实例所属可用区、实例容器名称等元数据项。

Key	Value	说明
eci_id	__ECI_ID__	实例ID。
eci_name	__ECI_NAME__	实例名称。
region_id	__REGION_ID__	实例所属地域。
zone_id	__ZONE_ID__	实例所属可用区。

Key	Value	说明
container_name	__CONTAINER_NAME__	实例容器名称。

```
params = {
        'Container.1.Image': 'registry-vpc.cn-shanghai.aliyuncs.com/eci_open/nginx:alpine',
        'Container.1.Name': 'nginx',
        'SecurityGroupId': 'sg-uf6biempwqvodk7a****',
        'VSwitchId': 'vsw-uf6mhqg2wiq9iifhn****',
        'ContainerGroupName': 'test-env',
        # 自定义环境变量
        'Container.1.EnvironmentVar.1.Key': 'eci_id',
        'Container.1.EnvironmentVar.2.Key': 'eci_name',
        'Container.1.EnvironmentVar.3.Key': 'region_id',
        'Container.1.EnvironmentVar.4.Key': 'zone_id',
        'Container.1.EnvironmentVar.5.Key': 'container_name',
        'Container.1.EnvironmentVar.1.Value': '__ECI_ID__',
        'Container.1.EnvironmentVar.2.Value': '__ECI_NAME__',
        'Container.1.EnvironmentVar.3.Value': '__REGION_ID__',
        'Container.1.EnvironmentVar.4.Value': '__ZONE_ID__',
        'Container.1.EnvironmentVar.5.Value': '__CONTAINER_NAME__',
    }
```

您可以通过控制台连接容器查看是否生效。具体操作，请参见调试ECI实例。



方式三：Downward API

Kubernetes Downward API提供了以下两种方式：

- 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
```

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

参数	描述
metadata.annotations	Pod的所有注解。

> **说明**
>
> 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。

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:
      - 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-kubelet
```

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#
```

4.配置Security Context

本文介绍如何为Pod或者Container配置Security Context，定义Pod或者Container的权限和访问控制。

背景信息

Security Context，即安全上下文，用于定义Pod或者Container的权限和访问控制，包括Discretionary Access Control、SELinux、Linux Capabilities等。更多信息，请参见Security Context。

Kubernetes提供了两种配置安全上下文的方法：

- Pod Security Context

  Pod级别，应用于Pod内所有容器和Volume。

  ECI支持通过配置Pod Security Context来修改sysctl参数和runAsUse。

- Container Security Context

  容器级别，应用于指定的容器。

  ECI支持通过配置Container Security Context来修改runAsUse和capabilities。

> **说明**
>
> Kubernetes使用Pod安全策略（Pod Security Policy）来验证和限制Pod的安全上下文。如果Pod的安全上下文配置不满足安全策略的约束，则无法创建Pod。更多信息，请参见Pod安全策略。
>
> 如果您使用阿里云ACK或者ASK，则默认使用名为ack.privileged的Pod Security Policy。更多信息，请参见使用Pod安全策略。

配置Pod Security Context

在Linux中，通常可以通过sysctl接口修改内核运行时的参数。对于ECI实例的内核参数，您可以通过以下命令进行查看。更多信息，请参见sysctl.sh。

```
sysctl -a
```

在Pod级别，您可以通过配置Security Context来修改sysctl参数和runAsUse。

目前ECI支持修改的sysctl参数如下：

- kernel.shm*（kernel.shm_rmid_forced除外）
- kernel.msg*
- kernel.sem
- fs.mqueue.*
- net.*（net.ipv4.ip_local_port_range和net.ipv4.tcp_syncookies除外）

> **警告**
>
> 为了避免破坏操作系统的稳定性，请您在充分了解sysctl参数变更影响后，再进行配置。

配置示例如下：

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

配置Container Security Context

在Container级别，您可以为指定的容器设置Security Context。

> **说明**
>
> 对于Pod Security Context和Container Security Context中均设置的参数（如runAsUse），容器的设置将覆盖Pod的设置。

目前ECI支持配置的参数如下：

支持配置的参数	说明
runAsUse	设置运行容器的用户ID。该参数配置会覆盖Dockerfile中的USER指令。

支持配置的参数	说明
capabilities	为容器内的进程授予某些特定的权限。更多信息，请参见Linux Capabilities。 目前支持配置以下权限： • AUDIT_WRITE • CHOWN • DAC_OVERRIDE • FSETID • FOWNER • KILL • MKNOD • NET_ADMIN • NET_BIND_SERVICE • NET_RAW • SETGID • SETUID • SETFCAP • SETPCAP • SYS_CHROOT • SYS_PTRACE • SYS_RAWIO 说明 SYS_RAWIO默认不支持，需提交工单申请。

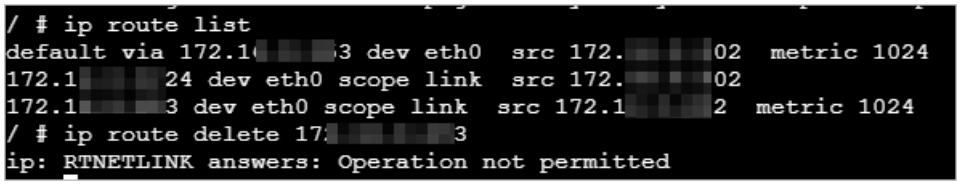
部分不支持配置的参数及其默认值如下：

不支持配置的参数	说明
privileged	容器是否以特权模式运行，默认为false。
AllowedProcMountTypes	指定容器可以挂载的proc类型，默认为DefaultProcMount。

不支持配置的参数	说明
readOnlyRootFilesystem	容器运行的根文件系统是否为只读，默认为true。

配置示例如下：

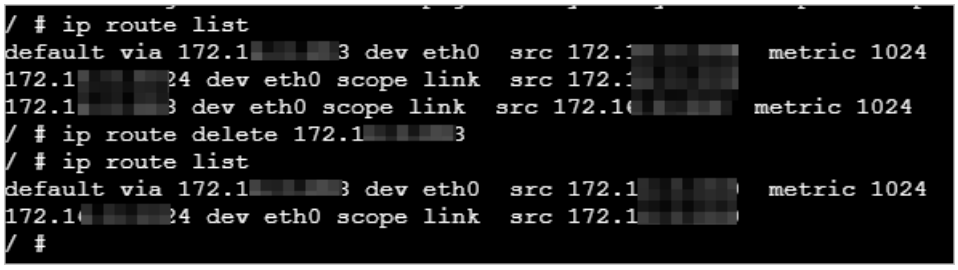
默认情况下，容器并不具备NET_ADMIN权限。如果在容器内进行网络相关操作，会返回报错提示。



您可以为容器配置Security Context，修改capabilities参数来增加NET_ADMIN权限。以下为示例：

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

配置后，在容器内可以进行网络相关操作。



5.开启Core dump

本文介绍如何开启Core dump，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

ECI默认关闭Core dump，避免磁盘占用过多而导致业务不可用。您可以根据需要选择以下一种方式开启Core dump：

- 方式一：开启Core dump运维任务

  手动开启Core dump后，将生成一个运维任务。在容器运行异常终止或者退出时，触发Core dump生成的core文件将自动保存到OSS中。

- 方式二：自定义设置core文件保存路径

  支持自定义设置core文件保存到外挂存储中，设置保存路径后，将自动开启Core dump。在容器运行异常终止或者退出时，触发Core dump生成的core文件，将保存到指定的外挂存储的路径下。

> **说明**
>
> - 方式一便于操作，但有时效和地域限制，可用于临时调试和诊断程序。
>   - 方式一生成的运维任务为一次性任务，执行成功获取到一次Core文件后，将会关闭Core dump，并且运维任务有一定的期限（12小时），超出时间后任务将会失效。
>   - 方式一基于阿里云对象存储OSS和消息服务MNS，因此ECI支持的地域中，以下不支持OSS和MMS的地域暂不支持使用该方式：华北2（北京）、华北6（乌兰察布）、华南2（河源）、华南3（广州）、华东5（南京）。
> - 方式二需要额外配置外挂存储，在程序运行状态不稳定的情况下，可以采用该方式确保能够获取到core文件，但如果程序有问题，反复重启可能会产生大量core文件。

方式一：开启Core dump运维任务

控制台

1. 登录弹性容器实例控制台，创建一个ECI实例。
2. 开启Core dump。
   i. 单击实例ID打开实例详情页面。
   ii. 单击**运维**页签，然后单击**开启**。

   单击**开启**后，将生成一个运维任务，未触发Core dump时，任务状态为**等待中**。

3. 触发Core dump。

   在容器内执行 `sleep 100` 命令后同时按 `Ctrl` 键和 `\` 键，触发Core dump，生成的core文件将自动保存到OSS中。

```
# sleep 100
^\Quit (core dumped)
```

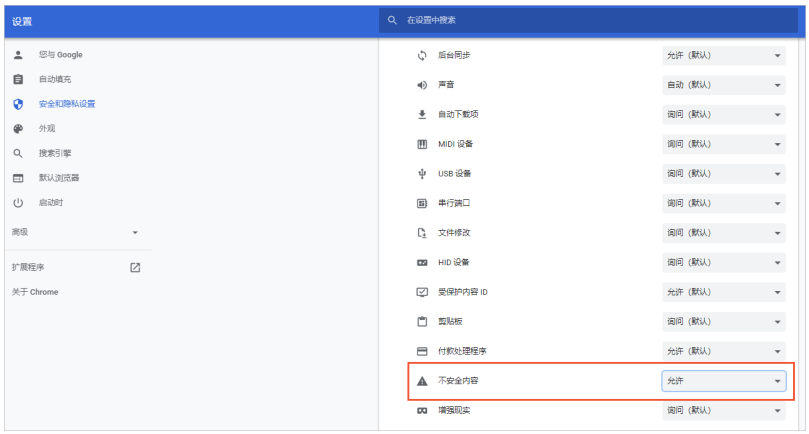
4. 下载core文件。

   在实例详情页的**运维**页签下，可以查看Core dump对应的运维任务。触发Core dump生成core文件后，运维任务的状态将变为**成功**，此时单击对应结果列中的**下载**即可下载core文件到本地。

如果下载没有反应，请检查浏览器的网站权限设置。例如Chrome浏览器可以参考以下方式开启权限：

i. 打开弹性容器实例控制台，单击浏览器地址栏前面的图标，选择**网站设置**。

ii. 将配置项**不安全内容**改为**允许**。



OpenAPI

1. 创建一个ECI实例。

   调用CreateContainerGroup接口创建ECI实例时，请勿设置CorePattern。

2. 开启Core dump。

   调用CreateInstanceOpsTask接口创建运维任务，将OpsType设为 `coredump` ，OpsValue设为 `enable` ，即可开启Core dump。更多信息，请参见CreateInstanceOpsTask。

3. 触发Core dump。

   在容器内执行 `sleep 100` 命令后同时按 `Ctrl` 键和 `\` 键，触发Core dump，生成的core文件将自动保存到OSS中。

   ```
   # sleep 100
   ^\Quit (core dumped)
   ```

4. 下载Core文件。

   调用DescribeInstanceOpsRecords接口查看运维任务的结果，从返回信息的ResultContent中，可以获取core文件保存在OSS中的地址，访问该地址即可下载core文件。

方式二：自定义设置core文件保存路径

配置说明

ECI支持自定义设置core文件保存路径，设置后将自动开启Core dump。core文件一般用于离线分析问题，因此设置core文件的保存路径时，一般采用外挂存储，而不是保存在容器本地路径，避免容器退出而丢失core文件。配置方式如下：

> **注意**
>
> 配置的路径不能以 `|` 开头，即不能通过Core dump来配置可执行程序。

- OpenAPI

  调用CreateContainerGroup接口创建ECI实例时，可传入CorePattern参数来设置core文件保存路径：

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

- Kubernetes

  创建Pod时，可添加Annotation来设置core文件保存路径：

  ```
  apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
  kind: Deployment
  metadata:
    name: deployment-test
    labels:
      app: test
  spec:
    replicas: 2
    selector:
      matchLabels:
        alibabacloud.com/eci: "true"
    template:
      metadata:
        labels:
          alibabacloud.com/eci: "true"
        annotations:
          k8s.aliyun.com/eci-core-pattern: "/xx/xx/core" #设置core文件保存路径
      spec:
        containers:
        - name: nginx
          image: nginx:1.7.9 # replace it with your exactly <image_name:tags>
          ports:
          - containerPort: 80
  ```

配置示例

以OpenAPI场景，挂载NAS作为外挂存储为例：

1. 创建一台ECI实例A，挂载NAS并设置core文件保存路径。

   调用CreateContainerGroup接口创建ECI实例A时传入以下参数，将NAS的 `/dump/` 目录挂载到容器的 `/data/dump-a/` 目录，将core文件保存路径设置为 `/data/dump/core` 。

```
'Volume.1.Name': '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': 'volume1',
'Container.1.VolumeMount.1.MountPath': '/data/dump-a/',

'CorePattern':'/data/dump-a/core',
```

2. 在实例A的容器任意目录下触发Core dump。

   如下示例，在容器内执行 `sleep 100` 命令后同时按 `Ctrl` 键和 `\` 键，触发Core dump后，core文件已保存到容器的 `/data/dump-a/` 路径下。

```
# ls
bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
# cd /data/dump-a/
# ls
# sleep 100
^\Quit (core dumped)
# ls
core.17
```

3. 释放ECI实例A。
4. 将同一NAS挂载到另一台ECI实例B。

   调用CreateContainerGroup接口创建ECI实例B时传入以下参数，将同一NAS的 `/dump/` 目录挂载到容器的 `/data/dump-b/` 目录。

```
'Volume.1.Name': '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': 'dvolume1',
'Container.1.VolumeMount.1.MountPath': '/data/dump-b/',
```

5. 在实例B的容器中查看core文件。

   如下示例，在容器的 `/data/dump-b/` 路径下可以看到core文件，core文件保存到外挂存储后，并没有随着实例A的释放而丢失，您仍可以查看分析core文件。

```
# ls
bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
# cd /data/dump-b/
# ls
core.17
```

6.设置容器终止消息

本文介绍如何设置容器的terminationMessagePath和terminationMessagePolicy字段，实现自定义设置容器终止消息。

Kubernetes可以通过terminationMessagePath来设置容器退出的消息来源，即当容器退出时，Kubernetes从容器的terminationMessagePath字段中指定的终止消息文件中检索终止消息，默认值为：/dev/termination-log。

通过定制terminationMessagePath字段，可以使得Kubernetes在容器运行成功或失败时，使用指定的自定义文件中的内容来填充容器的终止消息。终止消息内容最大为4KB。

在以下示例中，配置了terminationMessagePath字段为：/tmp/termination-log，则容器将把终止消息写入/tmp/termination-log给Kubernetes接收。

```
apiVersion: v1
kind: Pod
metadata:
  name: msg-path-demo
spec:
  containers:
  - name: msg-path-demo-container
    image: debian
    terminationMessagePath: "/tmp/termination-log"
```

此外，您还可以设置容器的terminationMessagePolicy字段，进一步自定义容器终止消息。该字段默认值为：File，即仅从终止消息文件中检索终止消息。您可以根据需要设置为：FallbackToLogsOnError，即在容器因错误退出时，如果终止消息文件为空，则使用容器日志输出的最后一部分内容来作为终止消息。

```
apiVersion: v1
kind: Pod
metadata:
  name: msg-path-demo
spec:
  containers:
  - name: msg-path-demo-container
    image: debian
    terminationMessagePath: "/tmp/termination-log"
    terminationMessagePolicy: "FallbackToLogsOnError"
```

Pod内所有容器的终止信息大小之和最大为12KB。当总和超过12KB时，Kubernetes的状态管理器会对其加以限制，例如：Pod内有4个InitContainer和8个应用Container，则状态管理器会限制每个容器的终止信息最大为1KB，既截取每个Container终止信息的前1KB。

7.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。具体操作，请参见升级Virtual Kubelet。

背景信息

对于不同的Kubernetes集群，Virtual Kubelet（VK）的升级支持情况如下：

- 阿里云ASK集群：自动升级VK。
- 阿里云ACK集群：分为托管版和专有版。托管版自动升级VK，专有版需要您手动升级VK。
- 自建集群：在ECS上或者线下自建的集群，需要您手动升级VK。

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                         sa
mple
===============================================================================
^* 10.10.5.1                    2           6     377     35       +40us[ +135us] +/-
14ms
```

8.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
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
```