阿里云
弹性伸缩

最佳实践

文档版本：20190724

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
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	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。	`swich` *`{stand | slave}`*

目录

1 让ESS更灵活的新特性

弹性伸缩（Elastic Scaling Service, ESS）是一种根据业务需求和策略，自动调整其弹性计算资源的管理服务。在满足业务需求高峰增长时无缝地增加 ECS 实例，并在业务需求下降时自动减少ECS实例以节约成本。

为了提供更加弹性、灵活的伸缩服务，ESS弹性伸缩配置中新增了 UserData、KeyPair、RamRole、Tags 4个特性。使用 UserData，您可以快速安全的完成自动化的配置过程，在 ECS 实例数量随着业务需求弹性变化的同时，您还能够安全、快速地完成应用级别的扩容和缩容。您还可以通过配置 KeyPair、Tags 等参数，实现更加高效、智能的 ECS 实例管理服务。

本文将详细介绍ESS新增的 4 个特性，并结合具体场景，向您阐述这些特性在 ESS 中的使用方式。您可以根据自己的业务场景，灵活地使用这些特性来满足您的业务需求。

实例自定义数据

实例自定义数据（UserData），是阿里云 ECS 为您提供的一种自定义实例启动行为及传入数据的功能，该功能兼容 Windows 实例及 Linux 实例，主要有 2 种用途：

- 作为实例自定义脚本，在启动实例时执行。
- 作为普通数据，将一定的信息传入实例中，您可以在实例中引用这些数据。

使用ESS来满足您 ECS 实例数随着业务需求弹性伸缩的要求时，如果您还要自动化地实现应用级别的扩容和缩容，常用的方法可能是通过自定义镜像的方式来实现，也可能是通过使用 Terraform 等开源的IT基础架构管理工具来实现。

ESS 伸缩配置中添加了 UserData 参数以后，您只需要准备好您的 UserData 自定义脚本数据，然后以 Base64 编码的方式传入伸缩配置中即可。当ESS弹性扩容 ECS 实例数时，UserData 实例自定义脚本会在实例启动的时候自动地执行，从而帮您实现应用级别的扩容和缩容。相比借助于自定义镜像或其它开源工具来实现应用自动扩展的方法，使用 ESS 原生的 UserData 特性显得更加快捷、安全。

在创建伸缩配置，并使用了UserData参数时，需要注意以下几点：

- 专有网络（VPC）的伸缩配置才能使用UserData参数。
- UserData 要以 Base64 编码的方式传入。
- UserData 将以不加密的方式传入，所以请不要以明文方式传入机密的信息（比如密码、私钥数据等），如果必须传入，建议加密后，然后以 Base64 的方式编码后再传入，在实例内部以同样的方式反解密。

SSH 秘钥对

在使用 SSH 登录远程 Linux 服务器时，您可以选择使用密码的方式来登录，也可以选择使用 SSH 秘钥对 的方式来登录。当您要管理的服务器集群较多时，频繁地输入密码不仅浪费时间，而且容易发生密码输入错误，无法登陆服务器的情况。

此时，如果您通过SSH 秘钥对的方式来登陆服务器，您只需要配置好您的公钥和私钥，即可登录到服务器。一次配置，长期有效。阿里云创建的SSH 秘钥对只支持RSA 2048位的密钥对。在生成秘钥的时候，阿里云会保存密钥的公钥部分，并返回给您秘钥的私钥部分。

ESS 弹性伸缩配置中的 KeyPairName 参数，为您提供了 SSH 秘钥对的方式来登录服务器的能力。在创建伸缩配置时，选择您想要使用的秘钥对名称作为 KeyPairName 参数配置到伸缩配置中。当 ECS 实例被弹性伸缩服务创建出来时，实例会存储此秘钥对的公钥部分，您只需要在本机配置一下秘钥对的私钥部分，便可以使用 SSH 秘钥对的方式快速地登录到您的服务器上去。

在创建伸缩配置，并使用了 KeyPairName 参数时，需要注意以下几点：

- Windows ECS 实例，忽略该参数。即使传入了 KeyPairName，也不会生效。
- 当传入了 KeyPairName 参数后，Linux ECS 实例的密码登录方式会被初始化成禁止。
- 关于秘钥对的创建，您可以参考 #unique_6 接口 。

RAM 角色名称

RAM (Resource Access Management) 是阿里云为客户提供的用户身份管理与访问控制服务。使用 RAM，您可以创建、管理用户账号（比如员工、系统或应用程序），并可以控制这些用户账号对您名下资源具有的操作权限。当您的企业存在多用户协同操作资源时，使用 RAM 可以让您避免与其他用户共享云账号密钥，按需为用户分配最小权限，从而降低您的企业信息安全风险。

RAM 支持创建不同的角色，不同的角色具有对不同的云产品的不同的操作权限。ESS 弹性伸缩配置新增了 RamRoleName 参数，您可以通过设置该参数，让您的 ECS 实例来扮演不同的角色，这些实例便拥有了这些角色不同的云产品的操作权限。在给伸缩配置指定RamRoleName 参数时，您需要确保当前的 RamRole 策略中允许您的ECS实例来扮演该角色，否则伸缩配置无法有效地弹出 ECS 实例。

关于 RamRole 的策略信息和创建方法，您可以参考 CreateRole 接口。

标签

阿里云 ECS 提供标签（Tags）服务，您可以通过给 ECS 实例绑定不同的标签的方式，实现对 ECS 实例的分类管理。

您可以通过查询不同的标签方式，获取符合条件的 ECS 实例列表。同样，您也可以通过查询 ECS 实例的方式，查询出匹配到的标签。ESS 弹性伸缩配置新增了 Tags 参数，您可以通过设置不同

的标签对，来对您ESS伸缩服务弹出的机器进行分类管理。每个伸缩配置暂时最多只能支持5对标签，当指定的标签数超过5对，伸缩配置将创建失败。

最佳实践

为了让您能够更加清晰地理解并准确地使用 ESS 伸缩服务，本章节将结合这些新特性，向您详细地介绍伸缩组和伸缩配置的创建过程，并为这些新参数中的每个参数都设置一个简单的使用场景，来加强您的理解。

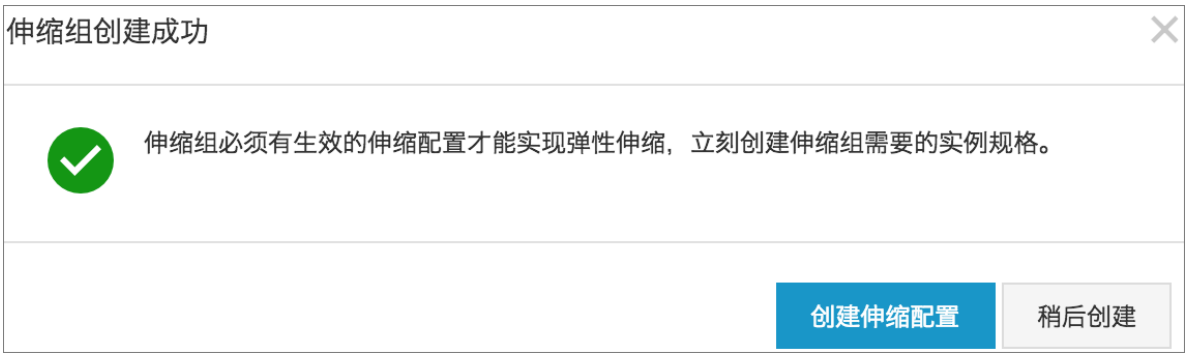
创建伸缩组

1. 创建一个伸缩组。登录ESS控制台，单击控制台右侧 创建伸缩组，弹出伸缩组创建对话框。

2. 伸缩组分为经典网络伸缩组、专有网络伸缩组。由于专有网络（VPC）环境下的伸缩配置才能使用实例自定义数据（UserData）参数，因此本文将基于专有网络伸缩组展开介绍。您需要指定伸缩组所在的虚拟交换机，然后点击确定，创建一个专有网络伸缩组。

创建伸缩配置

1. 创建完伸缩组以后，弹出 创建伸缩配置 对话框。



2. 单击 创建伸缩配置 按钮，弹出伸缩配置参数配置界面。

基本的伸缩配置参数这里不作过多介绍，您可以参考 创建伸缩配置 接口来了解每个参数的作用和设置方式。这里将重点介绍 UserData、KeyPairName、RamRoleName、Tags 4个参数的设置。

UserData 参数设置

UserData 支持 Windows 系统和 Linux 操作系统，这里将介绍 UserData 在 Linux 操作系统下，如何通过自定义shell脚本的方式，来实现在ECS实例启动时自动地运行已定义好的脚本。

1. 定义一个shell脚本，来实现在实例首次启动后向 */root/output10.txt* 文件写入字符串 `Hello World. The time is now{当前时间}`。在定义shell脚本时，需注意以下3点：

   - 格式：首行必须是 `#!`，如 `#!/bin/sh`。
   - 限制：在 Base64 编码前，脚本内容（包括首行在内）不能超过16 KB。
   - 频率：仅在首次启动实例时执行一次。

   脚本的内容如下：

   ```
   #!/bin/sh
   echo "Hello World.  The time is now $(date -R)!" | tee /root/
   output10.txt
   ```

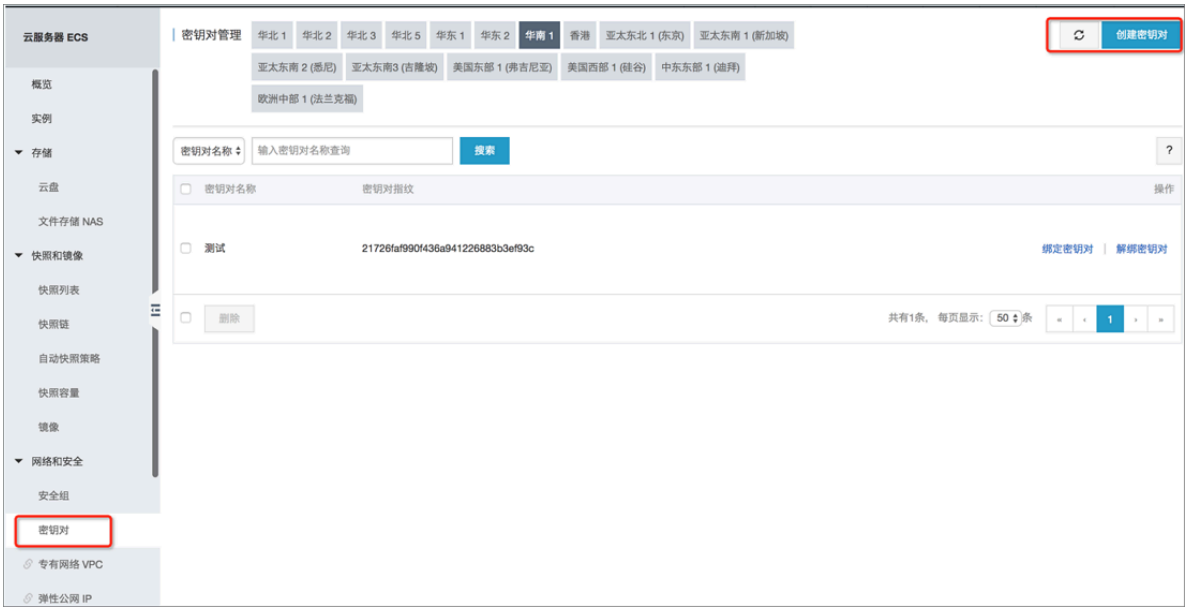
   脚本经过 Base64 编码后内容如下：

   ```
   IyEvYmluL3NoDQplY2hvICJIZWxsbyBXb3JsZC4gIFRoZSB0aW1lIGlzIG5v
   dyAkKGRhdGUgLVIpISIgfCB0ZWUgL3Jvb3Qvb3V0cHV0MTAudHh0
   ```

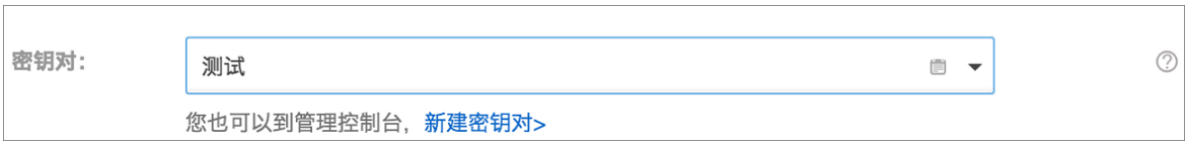
2. shell脚本准备好以后，伸缩配置中 镜像类型 选择 公共镜像，镜像 选择Linux类型的系统镜像如 Ubuntu，伸缩配置中用户自定义数据填写 Base64 编码以后的脚本数据，并勾选 输入已采用base64编码 选项，上述参数设置结果如下图所示：

KeyPairName参数设置

1. 在设置 KeyPairName 参数前，您需要确保伸缩配置所在区域已经存在创建好的秘钥对，如果没有创建好的秘钥对，需访问 秘钥对 列表页新建秘钥对。秘钥对创建界面如下图所示。



2. 秘钥对创建好以后，系统会返回秘钥的私钥部分，请妥善保管。秘钥的公钥部分由阿里云统一管理。伸缩配置参数中，请选择您想要用来登录实例的秘钥对名称，如下图所示：



Tags参数设置

在配置伸缩配置中实例标签参数时，您可以选用已有的标签，也可以新建标签。

说明：

实例标签参数目前只支持 key、value 都不为空的场景。在设置实例标签参数时，最多设置5对标签，超过5对标签伸缩配置将无法创建。伸缩配置中，配置好标签参数以后的效果，如下图所示：

RamRoleName参数设置

1. RamRoleName 参数目前在 ESS 控制台暂未开放，您可以通过调用 创建伸缩配置 API 的方式来使用此参数。因为弹性伸缩服务创建出来的 ECS 实例参数会扮演 RamRoleName 对应的角色.

   说明:

   ESS 伸缩配置中配置该参数时，您需要确保当前的 RamRole策略中允许您的 ECS 实例来扮演该角色。如 AliyunECSImageExportDefaultRole 镜像的导出角色，该角色允许当前用户的所有 ECS 实例来扮演这个角色，角色信息如下:

   ```
   {
   "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
     "Service": [
       "ecs.aliyuncs.com"
     ]
      }
    }
   ],
   "Version": "1"
   }
   ```

   其中 ecs.aliyuncs.com 表示该角色当前用户的所有 ECS 实例来扮演。

2. 配置好伸缩配置以后，单击 确定 创建伸缩配置。

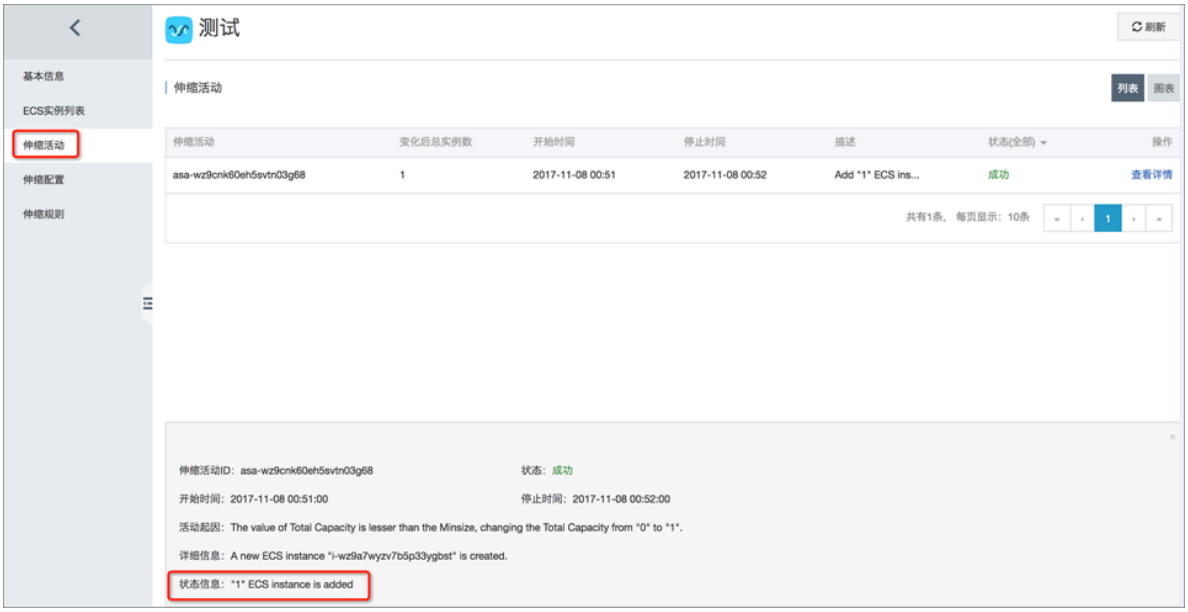
   

3. 单击 确定 按钮，启动伸缩配置。

验证伸缩活动

由于伸缩组创建的时候指定了最小实例数为1，弹性伸缩服务会启动一个伸缩任务，生产1台实例到当前伸缩组，使得当前伸缩组满足最小实例数的限制。

1. 您可以进入伸缩活动列表页，查看当前伸缩组对应的伸缩活动执行详情，如下图所示：



2. 实例通过当前伸缩配置生产出来以后，您需要在本机配置伸缩配中指定的秘钥对的私钥，然后以SSH Key 的方式来登录实例。不同的操作中，SSH 私钥的配置方式不同，这里以 Mac 系统为例，来介绍如何配置私钥。步骤如下：

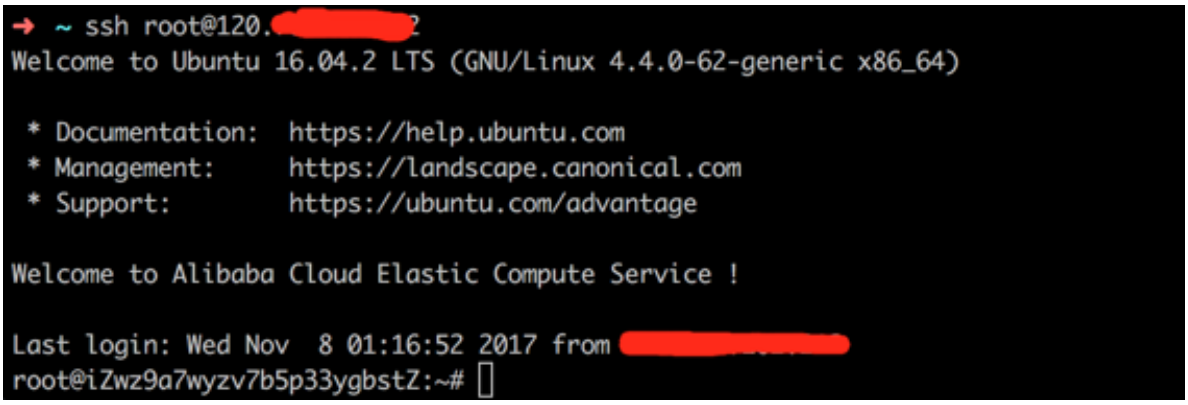
   a. 找到 */Users/{username}/.ssh/* 目录，如果找不到，需要新建。

   b. 在 */Users/{username}/.ssh/* 目录下新建 *id_rsa* 文件，将私钥信息拷贝进文件并保存。

   c. 执行 `ssh-add` 命令将私钥信息配置到系统。

   d. 使用命令 `ssh root@{publicIp}` 登录实例。其中 `publicIp` 代表实例的公网IP，可以通过查看本伸缩组的 ECS 实例列表获取。单击 实例ID 进去详情页查看，可以获取到实例的公网Ip，实例的标签等信息。

   SSH登录成功的效果，如下图所示：



3. 查看 `/root/output10.txt` 文件，校验 UserData 参数的执行结果，如下图所示：

```
root@iZwz9a7wyzv7b5p33ygbstZ:~# cat /root/output10.txt
Hello World.  The time is now Wed, 08 Nov 2017 00:52:55 +0800!
root@iZwz9a7wyzv7b5p33ygbstZ:~#
```

从上图可以看出伸缩配置中设置的 KeyPairName、UserData 等参数都已生效。

> 说明:
> 上述只展示了 UserData 的使用方式，使用的shell脚本比较简单。您可以根据自己的需求，定制您自己的脚本，然后将脚本内容经过 Base64 编码以后配置到伸缩配置中，实例在被弹出来并启动的时候，会执行您的自定义脚本。您可以通过这种方式，来实现应用的自动化扩容、缩容，应用的生命周期管理等功能。

ESS 不仅提供了在业务需求高峰或低谷时自动调节 ECS 实例数量的能力，而且提供了在 ECS 实例上快速安全地部署服务的能力。同时，ESS 还提供了一些管理 ECS 实例的新特性，帮助您更加高效灵活地管理您的 ECS 实例。

合理地使用 ESS 弹性伸缩服务，不仅能够有效地降低您的服务器成本，而且能够有效地降低您的服务管理和运维成本。您可以登录 弹性伸缩控制台 体验ESS的特性，同时也可以参考 创建伸缩配置 接口来使用这些新特性。

2 实例自定义数据

为了提供更加高效灵活的伸缩服务，弹性伸缩配置中新增了实例自定义数据（UserData）。您可以利用 UserData 自动完成 ECS 实例配置，从而安全快速地实现应用级别的扩容和缩容。

本文将详细介绍 UserData，并结合具体场景向您展示 UserData 的使用方式。您可以根据自己的业务场景，灵活地使用 UserData 来满足您的业务需求。

UserData 是阿里云 ECS 为您提供的一种自定义实例启动行为及传入数据的功能，该功能兼容 Windows 实例及 Linux 实例，主要有两种用途：

- 作为实例自定义脚本，在实例启动时自动执行。
- 作为普通数据，将一定的信息传入实例中，您可以在实例中引用这些数据。

在使用弹性伸缩调整 ECS 实例数时，如果您希望自动实现应用级别的扩容和缩容，常用方法是利用自定义镜像或者 Terraform 等开源的 IT 基础架构管理工具。

利用 UserData 参数，您只需要准备好自定义脚本数据，然后以 Base64 编码的方式传入伸缩配置中即可。当弹性扩容 ECS 实例时，实例自定义脚本会在实例启动时自动执行。相比自定义镜像和 Terraform 等开源工具，弹性伸缩原生的 UserData 特性更加快捷安全。

注意事项

如果您要在创建伸缩配置时使用 UserData 参数，需要注意：

- 只有专有网络（VPC）的伸缩配置能够使用 UserData 参数。
- 要以 Base64 编码的方式传入 UserData 参数。

另外，UserData 将以不加密的方式传入，请避免以明文方式传入机密的信息（比如密码、私钥数据等）。如果必须传入，建议先进行加密，以 Base64 的方式编码后再传入，然后在实例内部以同样的方式反解密。

> 说明：
> 更多 UserData 参数的使用方法，请参阅 实例自定义数据。

最佳实践

本章节将结合 Userdata 参数介绍伸缩组和伸缩配置的创建过程，并设置一个简单的使用场景，以便您理解弹性伸缩服务并准确地应用到自己的业务中。

创建伸缩组

首先，您需要创建一个伸缩组。由于只有专有网络（VPC）的伸缩配置能够使用 UserData 参数，因此本文将基于专有网络伸缩组展开介绍。

1. 登录 弹性伸缩控制台，单击页面右上角的 创建伸缩组。
2. 在 创建伸缩组 对话框中，填写必需参数并指定伸缩组所在的虚拟交换机。

3. 单击 提交 创建一个专有网络伸缩组。

创建伸缩配置

创建完伸缩组以后，会弹出 伸缩组创建成功 对话框，单击 创建伸缩配置 按钮，继续创建伸缩配置，如下图所示：

基本的伸缩配置参数这里不做过多介绍，您可以参考 创建伸缩配置 来了解每个参数的作用和设置方式，完成伸缩配置并启用伸缩组。下面通过 2 个场景来介绍 UserData 参数的设置。

UserData参数设置

UserData 支持 Windows 系统和 Linux 操作系统，这里介绍在 Linux 操作系统中的应用。您可以利用 UserData 实现在 ECS 实例启动时自动执行自定义 shell 脚本，在定义 shell 脚本时，需注意以下几点：

- 格式：首行必须以 `#!` 开头，例如 `#!/bin/sh`。
- 限制：在 Base64 编码前，脚本内容（包括首行在内）不能超过 16 KB。
- 频率：仅在首次启动实例时执行一次。

示例1

定义一个 shell 脚本，来实现在实例首次启动后向 `/root/output10.txt` 文件写入字符串 `Hello World. The time is now {当前时间}`。

脚本内容如下：

```
#!/bin/sh
echo "Hello World.  The time is now $(date -R)!" | tee /root/output10.
txt
```

脚本经过 Base64 编码后内容如下：

```
IyEvYmluL3NoDQplY2hvICJIZWxsbyBXb3JsZC4gIFRoZSB0aW1lIGlzIG5vdyAkKGRhdG
UgLVIpISIgfCB0ZWUgL3Jvb3Qvb3V0cHV0MTAudHh0
```

示例2

定义一个 shell 脚本，来实现在 ECS 启动时配置 DNS、yum、NTP 等。

脚本内容如下：

```
#!/bin/sh
# Modify DNS
echo "nameserver 8.8.8.8" | tee /etc/resolv.conf
# Modify yum repo and update
rm -rf /etc/yum.repos.d/*
touch myrepo.repo
echo "[base]" | tee /etc/yum.repos.d/myrepo.repo
echo "name=myrepo" | tee -a /etc/yum.repos.d/myrepo.repo
echo "baseurl=http://mirror.centos.org/centos" | tee -a /etc/yum.repos
.d/myrepo.repo
echo "gpgcheck=0" | tee -a /etc/yum.repos.d/myrepo.repo
echo "enabled=1" | tee -a /etc/yum.repos.d/myrepo.repo
yum update -y
# Modify NTP Server
echo "server ntp1.aliyun.com" | tee /etc/ntp.conf
systemctl restart ntpd.service
```

脚本经过 Base64 编码后内容如下：

```
IyEvYmluL3NoCiMgTW9kaWZ5IEROUwplY2hvICJuYW1lc2VydmVyIDguOC44LjgiIHwgdG
VlIC9ldGMvcmVzb2x2LmNvbmYKIyBNb2RpZnkgeXVtIHJlcG8gYW5kIHVwZGF0ZQpybSAt
```

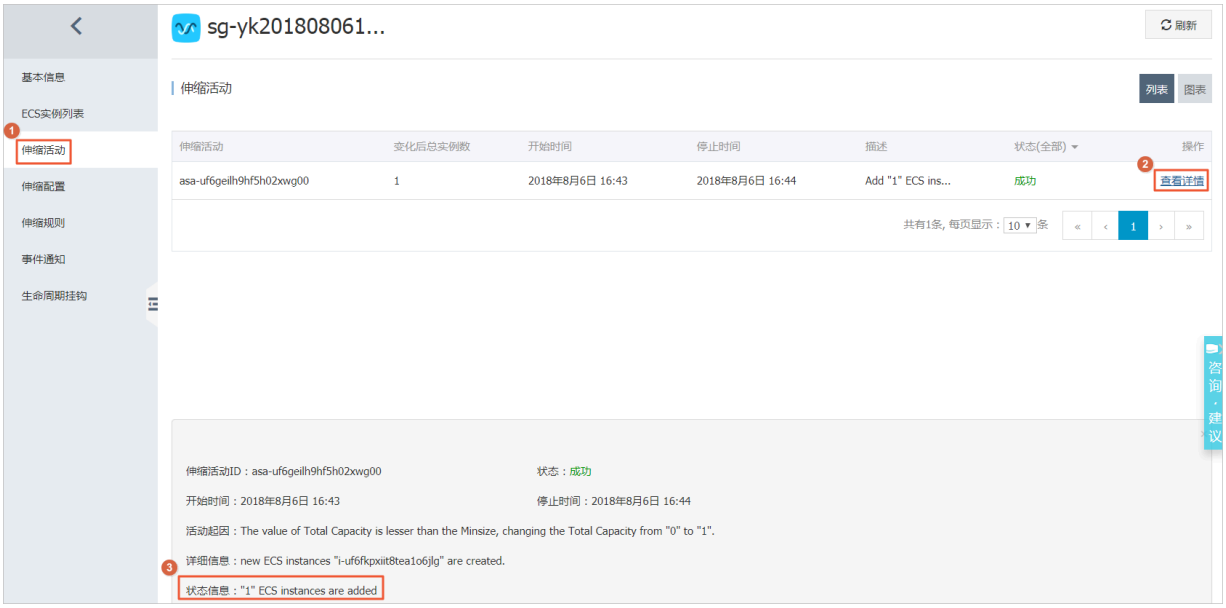
```
cmYgL2V0Yy95dW0ucmVwb3MuZC8qCnRvdWNoIG15cmVwby5yZXBvCmVjaG8gIltiYXNlXS
IgfCB0ZWUgL2V0Yy95dW0ucmVwb3MuZC9teXJlcG8ucmVwbwplY2hvICJuYW1lPW15cmVw
byIgfCB0ZWUgLWEgL2V0Yy95dW0ucmVwb3MuZC9teXJlcG8ucmVwbwplY2hvICJiYXNldX
JsPWh0dHA6Ly9taXJyb3IuY2VudG9zLm9yZy9jZW50b3MiIHwgdGVlIC1hIC9ldGMveXVt
LnJlcG9zLmQvbXlyZXBvLnJlcG8KZWNobyAiZ3BnY2hlY2s9MCIgfCB0ZWUgLWEgL2V0Yy
95dW0ucmVwb3MuZC9teXJlcG8ucmVwbwplY2hvICJlbmFibGVkPTEiIHwgdGVlIC1hIC9l
dGMveXVtLnJlcG9zLmQvbXlyZXBvLnJlcG8KeXVtIHVwZGF0ZSAteQojIE1vZGlmeSBOVF
AgU2VydmVyCmVjaG8gInNlcnZlciBudHAxLmFsaXl1bi5jb20iIHwgdGVlIC9ldGMvbnRw
LmNvbmYKc3lzdGVtY3RsIHJlc3RhcnQgbnRwZC5zZXJ2aWNl
```

shell 脚本准备完毕后，伸缩配置中镜像类型选择 系统镜像，然后选择 Linux 类型的镜像（例如 Ubuntu）：

在 高级选项 下的 实例自定义数据 中填写 Base64 编码以后的脚本数据，并勾选 输入已采用base64编码选项，下图为 示例1 的效果：

3. 查看伸缩活动

由于创建伸缩组时指定 伸缩最小实例数 为 1，启动伸缩组后，弹性伸缩服务会立即启动一个伸缩任务，向当前伸缩组添加 1 台实例，以满足最小实例数的限制。您可以进入伸缩活动列表页，查看当前伸缩组对应的伸缩活动执行详情，如下图所示：



4. 验证 UserData 参数效果

登录伸缩活动中创建的实例，查看 `/root/output10.txt` 文件，验证 示例1 的 UserData 参数的执行结果，如下图所示：

```
root@iZwz9a7wyzv7b5p33ygbstZ:~# cat /root/output10.txt
Hello World.  The time is now Wed, 08 Nov 2017 00:52:55 +0800!
root@iZwz9a7wyzv7b5p33ygbstZ:~#
```

查看服务状态，验证 示例2 的 UserData 参数的执行结果，如下图所示：

```
1. root@iZb[redacted]b1bz7jq6zzZ:~ (ssh)
[root@iZ[redacted]z7jq6zzZ ~]# cat /etc/resolv.conf
nameserver 8.8.8.8
[root@iZ[redacted]z7jq6zzZ ~]# cat /etc/yum.repos.d/myrepo.repo
[base]
name=myrepo
baseurl=http://mirror.centos.org/centos
gpgcheck=0
enabled=1
[root@iZb[redacted]z7jq6zzZ ~]# cat /etc/ntp.conf
server ntp1.aliyun.com
[root@iZb[redacted]7jq6zzZ ~]#
```

可以看出伸缩配置中设置的 UserData 参数已经生效。

本最佳实践向您展示了 UserData 的使用方式，但使用的 shell 脚本比较简单。您可以根据自己的需求定制脚本，更加高效灵活地管理您的 ECS 实例。

更多特性

面对不断变化的业务需求，弹性伸缩不仅为您提供了在业务需求高峰或低谷时自动调节 ECS 实例数量的能力，而且为您提供了在 ECS 实例上快速安全地部署服务的能力。除实例自定义数据（UserData）外，SSH 密钥对（KeyPairName）、RAM 角色名称（RamRoleName）和标签（Tags）也可以帮助您更加高效灵活地管理 ECS 实例。合理地使用弹性伸缩服务，不仅能够有效地降低您的服务器成本，而且能够有效地降低您的服务管理和运维成本。

您可以登录 弹性伸缩控制台 体验弹性伸缩的新特性，同时也可以参考 创建伸缩配置 来使用这些新特性。

3 通过扩缩容策略降低成本

本文介绍如何基于多实例规格和多可用区应用成本优化策略，提高弹性伸缩的成功率，同时降低成本。

背景信息

弹性伸缩支持多实例规格，您可以在伸缩配置中指定备选的实例规格。在无法弹出高优先级规格的实例时，弹性伸缩会自动尝试下一优先级规格的实例，直至成功创建实例。多实例规格可以有效应对单个实例规格库存不足的情况，保证伸缩活动可以顺利执行。在业务高峰时，您可能需要争分夺秒地弹出高配实例规格承载业务流量，更关注性能，而不限定于一种特定规格，此时多实例规格尤为实用。

弹性伸缩支持多可用区，您可以在创建伸缩组时指定多台虚拟交换机，在一台虚拟交换机所在的可用区库存不足时，弹性伸缩会自动尝试在其它可用区创建实例，保证伸缩活动可以顺利执行。配置多可用区后，您还可以根据业务部署情况配置对应的扩缩容策略，灵活满足业务需要。多可用区扩缩容策略包括优先级策略、均衡分布策略和成本优化策略，详细信息，请参阅多可用区扩缩容策略说明。

由于抢占式实例受到市场价格限制，竞价失败可能会导致扩容不及时，影响业务运行。此时您可以选择应用成本优化策略，在抢占式实例创建失败时，伸缩组会自动尝试创建相同规格的按量实例，兼顾了成功率和成本，配合多实例规格更会大大提高伸缩活动成功率。应用成本优化策略的伸缩组会按vCPU单价从低到高尝试创建ECS实例，即使您未选用抢占式实例，也能够以最低的价格使用同等规模的ECS实例资源。

注意事项

- 多可用区扩缩容策略仅适用于网络类型为专有网络的伸缩组。
- 不支持修改伸缩组的多可用区扩缩容策略。

准备工作

在创建应用成本优化策略的伸缩组前，请确保：

- 您已经创建了专有网络。
- 您在专有网络下创建了多个虚拟交换机，且虚拟交换机分布在多个可用区内。

操作步骤

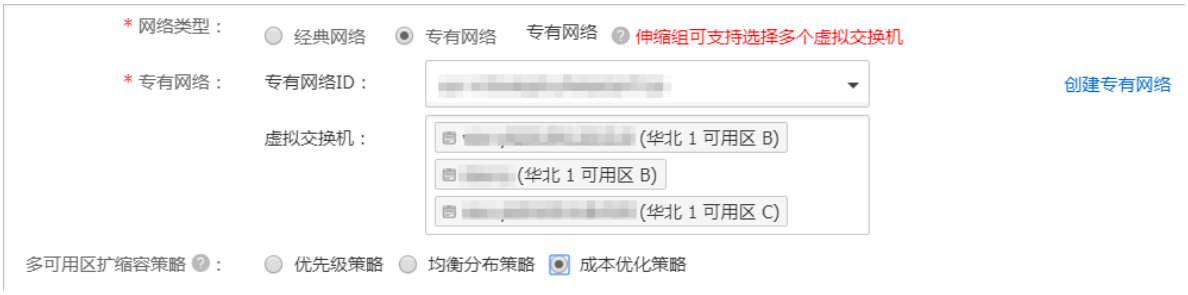
说明：

本步骤重点介绍多可用区扩缩容策略相关的选项，如需了解其它伸缩组选项，请参阅使用自定义伸缩配置创建伸缩组或者使用实例启动模板创建伸缩组。

1. 登录弹性伸缩控制台。
2. 在创建伸缩组页面中，将网络类型配置为专有网络，然后选择专有网络下的多个虚拟交换机。

   说明:

   由于一个虚拟交换机只归属于一个可用区，选择多个虚拟交换机即可以在多个可用区创建ECS实例，合理利用不同可用区的库存。
3. 在创建伸缩组页面中，将多可用区扩缩容策略配置为成本优化策略。



4. 根据需要配置其余伸缩组选项。
5. 为伸缩组创建伸缩配置，将计费方式配置为抢占式实例，然后选择多个实例规格（不超过10个）。

   说明:

   - 建议您按照vCPU、内存、处理器主频、内网带宽或者内网收发包等维度选择多个相近的实例规格。
   - 建议您根据预算设定最高价，如果您使用自动出价，伸缩组会按照抢占式实例的市场价格出价并创建抢占式实例。
   - I/O优化实例和非I/O优化实例的配置相差较大，即使同时选择也很难增加伸缩成功率。
6. 根据需要配置其余伸缩配置选项。
7. 启用伸缩组。
8. 创建一条伸缩规则add1，效果为向伸缩组添加一台ECS实例。

成本控制效果

假设在操作步骤中，为伸缩组指定了两个可用区下的虚拟交换机：华北1 可用区 B、华北 1 可用区 C，伸缩配置中指定了两种实例规格：ecs.sn1.large、ecs.sn1.xlarge。由于计费方式为抢占式实例，每种实例规格会对应两种单价：抢占式实例vCPU单价、按量付费实例vCPU单价。

注意:

本文列出的价格仅用作示例，购买时请以售卖页的实时价格为准。


弹性伸缩 ESS
伸缩组名称:
规格族
实例规格
计算型(原独享) sn1
计算型(原独享) sn1
计算型(原独享) sn1
计算型(原独享) sn1
计算型(原独享) sn1
当前选择实例:
设置单台实例规格上限价格:
使用
0.697
多选的实例规格:
规格：
单台实
已选择 2
可在已选
镜像 *
系统镜像
自定义镜像
抢占式实例市场价格

组合实例规格和计费方式后，可以得出四种创建实例的方案（vCPU单价从低到高排序）：

方案编号	实例规格	计费方式	vCPU	市场价格	vCPU单价
Solution1	ecs.sn1.xlarge	抢占式实例	8	0.158/时	0.01975/时
Solution2	ecs.sn1.large	抢占式实例	4	0.088/时	0.022/时
Solution3	ecs.sn1.xlarge	按量付费	8	1.393/时	0.174125/时
Solution4	ecs.sn1.large	按量付费	4	0.697/时	0.17425/时

期望动作：当发生扩容伸缩活动时，伸缩组优先按方案 Solution1 创建实例，如果在可用区 B 和可用区 C 下均无法创建出实例，则依次尝试方案 Solution2、Solution3 和 Solution4。

接下来，执行伸缩规则add1触发伸缩活动，向伸缩组添加1台ECS实例。然后，前往弹性伸缩控制台的ECS实例列表页面，单击刚创建的ECS实例，查看计费方式和实例规格，分别为ecs.sn1.xlarge和按量-抢占式实例，成功降低了成本。


实例详情
本实例磁盘
本实例快照
本实例安全组
本实例安全防护
实例规格：ecs.sn1.xlarge
配置信息
更换系统盘
更多
CPU：8核
内存：16 GB
实例类型：I/O优化
操作系统：CentOS 7.6 64位
付费信息
购买相同配置
更多
付费方式：按量-抢占式实例

4 使用ESS SDK快速创建多可用区伸缩组

ESS 伸缩组分为经典网络伸缩组和专有网络伸缩组。当您在创建专有网络伸缩组的时候，需要配置伸缩组对应的交换机。伸缩组创建出来以后，通过当前的伸缩组弹性创建出来的 ECS 实例都属于该交换机。

原 ESS 弹性伸缩服务限定，一个专有网络伸缩组只能配置一个交换机。由于一个交换机只归属于一个可用区，这样存在的问题就是，当您配置好 ESS 伸缩组的交换机以后，如果交换机所在的可用区由于库存不足等原因无法创建出 ECS 实例，那么您伸缩组中的伸缩配置、伸缩规则以及伸缩组对应的报警任务等都将生效。

为了优化上述问题，提高伸缩组的可用性，ESS伸缩组新增多可用区参数（VSwitchIds.N），您在创建伸缩组的时候可以使用该参数为您的伸缩组配置多个交换机，当一个交换机所在可用区无法创建实例的时候，ESS弹性伸缩服务会为您自动切换到其它可用区。在使用该参数的时候，您需要注意以下几点：

- 如果使用了 VSwitchIds.N 多可用区参数，VSwitchId 参数将被忽略。
- VSwitchIds.N 参数中，N的取值范围为[1, 5]，即一个伸缩最多可以配置5个交换机。
- VSwitchIds.N 参数中指定的交换机必须在同一个 VPC 下。
- VSwitchIds.N 参数中N代表交换机的优先级，编号为1的交换机为创建实例的第一选择，交换机优先级随编号的增大依次降低。
- 当优先级较高的交换机所在可用区无法创建实例时，会自动选择下一优先级的交换机来创建实例。当您在使用多可用区参数来创建伸缩组时，尽可能地设置同一地域下不同可用区的交换机来创建您的伸缩组，这样可以有效地减少单可用区无法创建出实例问题的发生，提高了伸缩组的可用性。

使用SDK创建多可用区伸缩组

本章将介绍如何使用 SDK 创建多可用区的伸缩组，这里以 Java 语言和 Python 语言为例进行介绍。

Java

1. 导入 ESS Java SDK

   首先，您需要下载依赖库 `aliyun-java-sdk-core`、`aliyun-java-sdk-ess`，您可以查看 maven-central界面，搜索并下载相应的jar包，`aliyun-java-sdk-ess` 对应的 jar 包的版

本号需要是2.1.3及以上版本才能使用多可用区参数，aliyun-java-sdk-core 对应的 jar 包的版本号推荐使用最新版本。

您也可以使用 maven 来管理您 Java 项目的依赖库，在您的项目对应的 pom.xml 文件中加入下面的依赖项：

```
<dependency>
<groupId>com.aliyun</groupId>
<artifactId>aliyun-java-sdk-ess</artifactId>
<version>2.1.3</version>
</dependency>
<dependency>
<groupId>com.aliyun</groupId>
<artifactId>aliyun-java-sdk-core</artifactId>
<version>3.5.0</version>
</dependency>
```

2. 使用 Java SDK 创建多可用区伸缩组

导入 ESS Java SDK 到您的 Java 工程后，您就可以使用 SDK 来创建多可用区伸缩组了。使用 Java SDK 创建多可用区伸缩组的代码如下：

```
public class EssSdkDemo {
public static final String       REGION_ID          = "cn-hangzhou";
public static final String       AK                 = "ak";
public static final String       AKS                = "aks";
public static final Integer      MAX_SIZE           = 10;
public static final Integer      MIN_SIZE           = 1;
public static final String       SCALING_GROUP_NAME = "TestScalin
gGroup";

//交换机列表，交换机优先级从前往后依次降低，第一位的交换机优先级最高。
public static final String[]     vswitchIdArray     = { "vsw-id1", "
vsw-id2", "vsw-id3", "vsw-id4", "vsw-id5" };
public static final List<String> vswitchIds         = Arrays.asList(
vswitchIdArray);
public static void main(String[] args) throws Exception {
    IClientProfile clientProfile = DefaultProfile.getProfile(
REGION_ID, AK, AKS);
    IAcsClient client = new DefaultAcsClient(clientProfile);
    createScalingGroup(client);
}

/**
 * 创建多可用区伸缩组。
 * @param client
 * @return
 * @throws Exception
*/
public static String createScalingGroup(IAcsClient client) throws
Exception {
    CreateScalingGroupRequest request = new CreateScalingGroupRe
quest();
    request.setRegionId("cn-beijing");
    request.setMaxSize(MAX_SIZE);
    request.setMinSize(MIN_SIZE);
    request.setScalingGroupName(SCALING_GROUP_NAME);
    request.setVSwitchIds(vswitchIds);
```

```
    CreateScalingGroupResponse response = client.getAcsResponse(
request);
    return response.getScalingGroupId();
}
}
```

上述代码中，VSwitch 的优先级随着它在列表中出现的先后顺序依次降低，排在列表最前面的 VSwitch 优先级最高。

Python

1. 安装 ESS Python SDK

   同 Java 语言一样，在使用 ESS Python SDK 前，您需要先下载依赖库 *aliyun-python-sdk-ess*、*aliyun-python-sdk-core*。本文推荐使用 pip 的方式来安装 Python 依赖包，关于 pip 的安装您可以参考 Installation-Pip。安装好 pip 以后，您可以使用命令 `pip install aliyun-python-sdk-ess==2.1.3 pip install aliyun-python-sdk-core==3.5.0` 安装两个依赖库。

2. 使用 Python SDK 创建多可用区伸缩组

   导入 ESS Python SDK 依赖库以后，您就可以使用 SDK 来创建多可用区伸缩组了。使用 Python SDK 创建多可用区伸缩组的代码如下：

```
#  coding=utf-8
import json
import logging
from aliyunsdkcore import client
from aliyunsdkess.request.v20140828.CreateScalingGroupRequest import
 CreateScalingGroupRequest
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s %(filename)s[line:%(lineno)d] %(
levelname)s %(message)s',
                    datefmt='%a, %d %b %Y %H:%M:%S')
# 请替换自己的ak信息。
ak = 'ak'
aks = 'aks'
scaling_group_name = 'ScalingGroupTest'
max_size = 10
min_size = 1
vswitch_ids = ["vsw-id1", "vsw-id2", "vsw-id3", "vsw-id4", "vsw-id5
"]
region_id = 'cn-beijing'
clt = client.AcsClient(ak, aks, region_id)
```

```
def _create_scaling_group():
request = CreateScalingGroupRequest()
request.set_ScalingGroupName(scaling_group_name)
request.set_MaxSize(max_size)
request.set_MinSize(min_size)
request.set_VSwitchIds(vswitch_ids)
response = _send_request(request)
return response.get('ScalingGroupId')
```

```
def _send_request(request):
```

```
request.set_accept_format('json')
try:
    response_str = clt.do_action(request)
    logging.info(response_str)
    response_detail = json.loads(response_str)
    return response_detail
except Exception as e:
    logging.error(e)
if __name__ == '__main__':
scaling_group_id = _create_scaling_group()
print '创建伸缩组成功，伸缩组ID:' + str(scaling_group_id)
```

上述代码中，VSwitch 的优先级随着它在列表中的出现的顺序依次降低，排在列表最前面的 VSwitch 优先级最高。

您可以参考 #unique_24 获取更多关于伸缩组创建的相关信息。伸缩组创建完以后，您可以参考 创建伸缩配置 来创建伸缩组下对应的伸缩配置。关于伸缩组和伸缩配置的创建，您还可以参考 让ESS更灵活的新特性 来了解完整的伸缩组和伸缩配置创建过程。

5 自动将伸缩组ECS实例添加到Redis实例白名单

本文介绍如何使用弹性伸缩、消息服务和函数计算，将弹性扩张时创建的ECS实例自动添加到Redis实例的白名单。您可以作为参考，实现更多自动化操作。

背景信息

目前，弹性伸缩已经接入负载均衡、云数据库RDS等产品，但是暂未接入云数据库Redis。如果您将业务数据存储在Redis实例上，可能需要配置伸缩组内的ECS实例访问Redis实例。等待ECS实例创建完成后再逐个手动添加不仅费时费力，也容易出现失误，在维护大量实例时成本较高。

针对这种情况，您可以在伸缩组中创建生命周期挂钩，生命周期挂钩在弹性扩张时会自动向指定的MNS主题发送消息，然后通过函数计算中的MNS主题触发器，触发执行上传的代码，自动将ECS实例添加到Redis实例的白名单。

准备工作

在操作前，请确保您已经开通云数据库Redis、消息服务、弹性伸缩和函数计算。

注意事项

- 建议在相同地域创建消息服务主题、消息服务队列、函数计算服务、函数计算函数和Redis实例。
- 本文以Java语言的形式给出示例代码，您可以根据业务需求，将此类最佳实践扩展到其它语言。

一、云数据库Redis操作步骤

1. 登录云数据库Redis控制台。
2. 创建一台Redis实例，用于为自动创建的ECS实例提供数据库服务。
3. 查看Redis实例的白名单，确定执行代码前的白名单状态。

二、消息服务操作步骤

1. 登录消息服务控制台。

2. 创建一个MNS主题，用作执行函数的触发器。示例主题的名称为`fc-trigger`。

3. 创建一个MNS队列，用作函数执行结果的接收器。示例队列的名称为`fc-callback`，示例代码中通过`QUEUE_NAME`指定该队列，发送包含函数执行结果的消息。

三、弹性伸缩操作步骤

1. 登录弹性伸缩控制台。

2. 创建一个伸缩组，详细步骤请参见使用自定义伸缩配置创建伸缩组或者使用实例启动模板创建伸缩组。
3. 创建一个生命周期挂钩。

a. 适用的伸缩活动类型配置为弹性扩张活动，用于通知弹性扩张事件。
b. 通知方式配置为MNS主题，与MNS队列相比，主题可以通知多个订阅者，执行多种操作。
c. MNS主题配置为`fc-trigger`，用于在自动创建的ECS实例进入加入挂起中状态时执行代码，将ECS实例添加到云数据库Redis的白名单。
d. 根据需要配置其它选项。

四、函数计算操作步骤

1. 登录函数计算控制台。

2. 新建一个服务，用于承载需要执行的函数。示例服务的名称为as-hook-mns-fc-redis。

3. 在服务下新建函数，订阅MNS主题并上传代码。

   a. 在函数模板页面中，选择空白函数。

   b. 在触发器配置页面中，选择MNS 主题触发器，然后根据需要配置其它选项。

   c. 在基础管理配置页面中，所在服务配置为as-hook-mns-fc-redis，函数入口配置为fc.Example::handleRequest，然后根据需要配置其它选项。

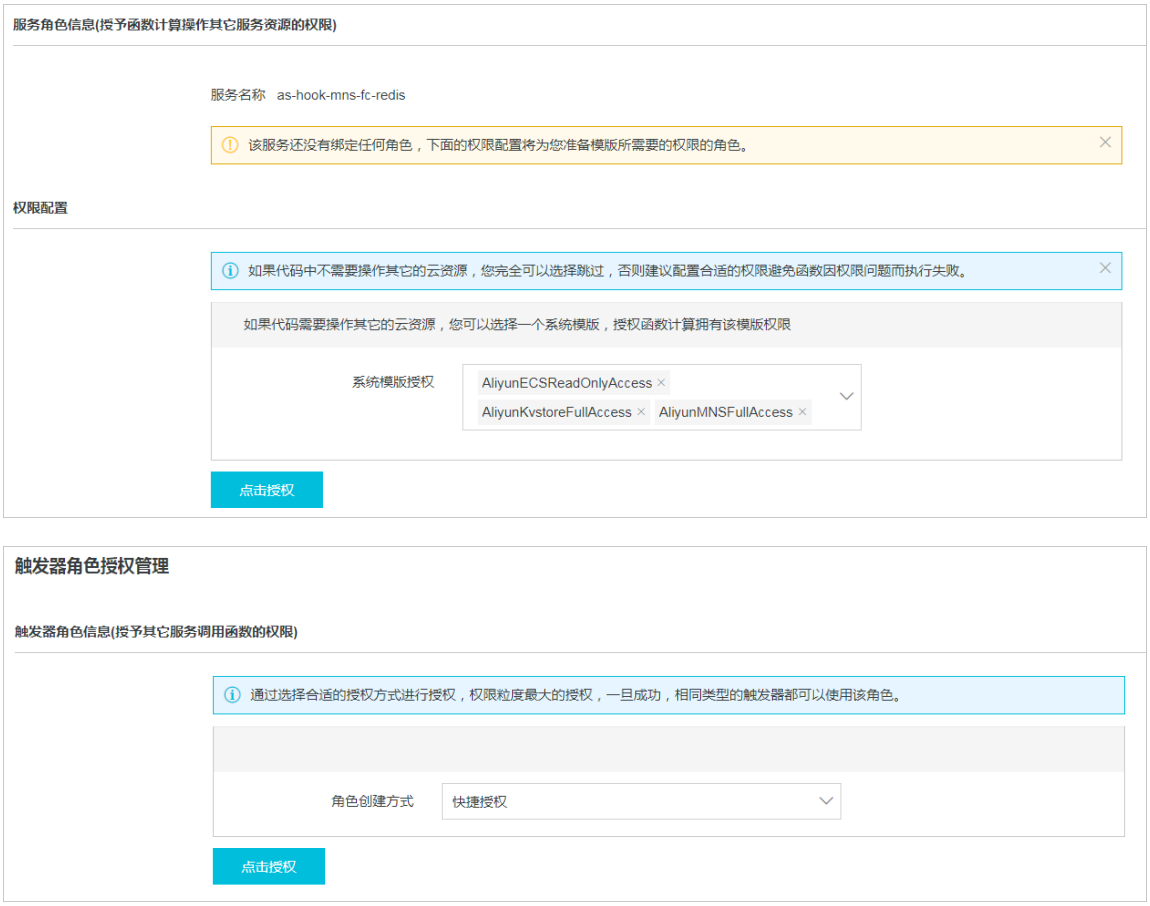
> **说明:**
>
> · 函数入口由代码决定，请根据实际情况配置。

- 本文示例采用上传jar包，实现将自动创建的ECS实例添加到云数据库Redis的白名单。有关编程语言说明，请参见函数计算Java编程说明。

d. 在权限配置页面中，根据需要授予函数访问其它资源的权限，并授予消息服务调用函数的权限。

说明:

建议遵循权限最小化原则，仅授予必需的权限，防范潜在风险。



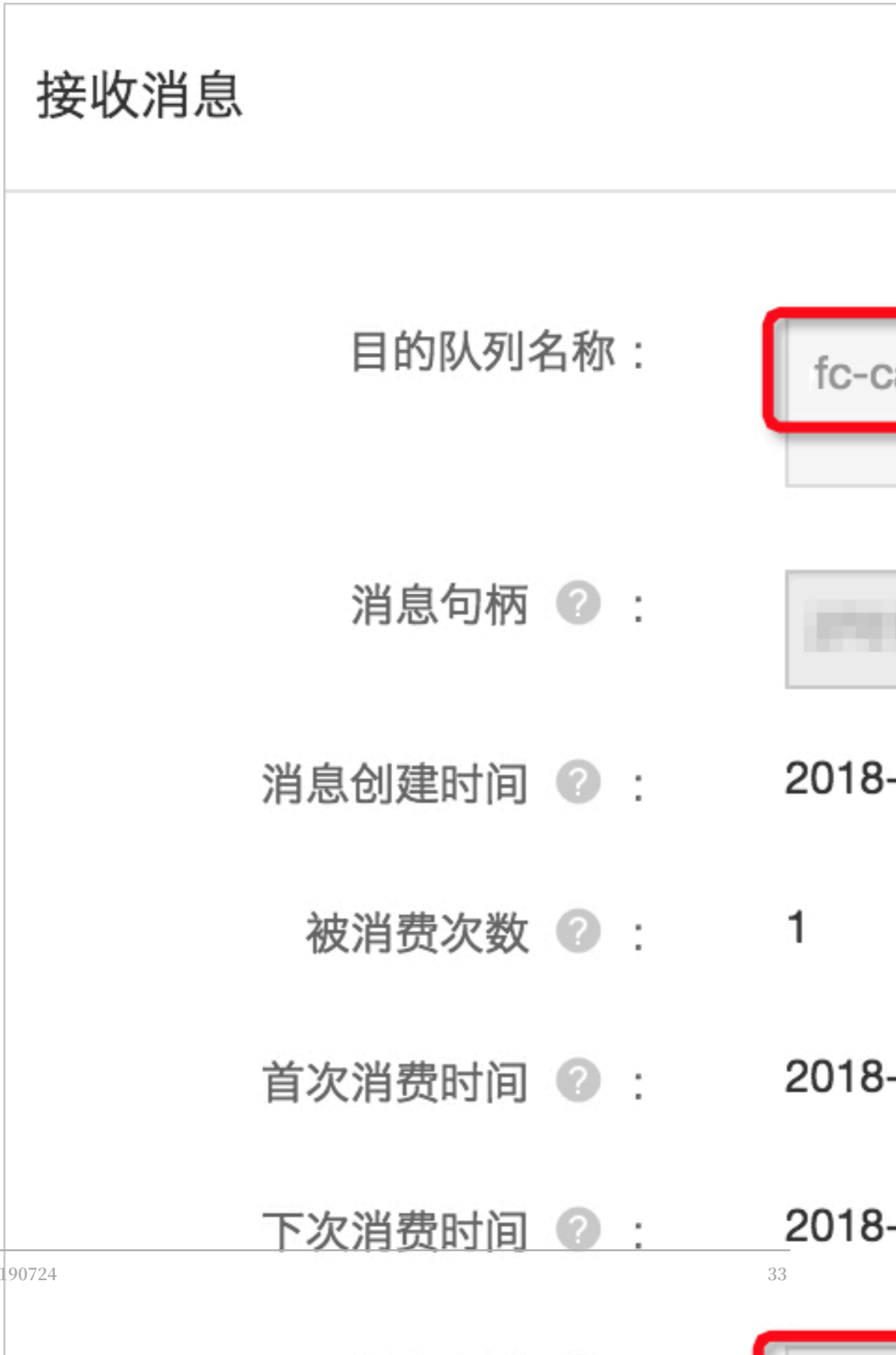
e. 在信息核对页面中，核对函数信息和触发器信息，然后单击创建。

执行效果

配置完成后，执行效果如下：

1. 在满足弹性扩张的条件时，伸缩组触发伸缩活动，自动创建ECS实例。
2. 生命周期挂钩挂起伸缩活动，同时发送消息到MNS主题。
3. 函数计算中，MNS主题触发器触发函数执行过程，并将消息内容作为输入信息（包括ECS实例的ID等信息），执行Java代码。
4. 代码执行时，会通过接口获取ECS实例的私网IP，然后将私网IP添加到Redis实例的白名单（default 分组）。

5. 代码执行结果会发送到MNS队列`fc-callback`，您可以在消息服务控制台查看结果详情。下图消息中`success`为`true`，表明ECS实例成功添加到了Redis实例的白名单。

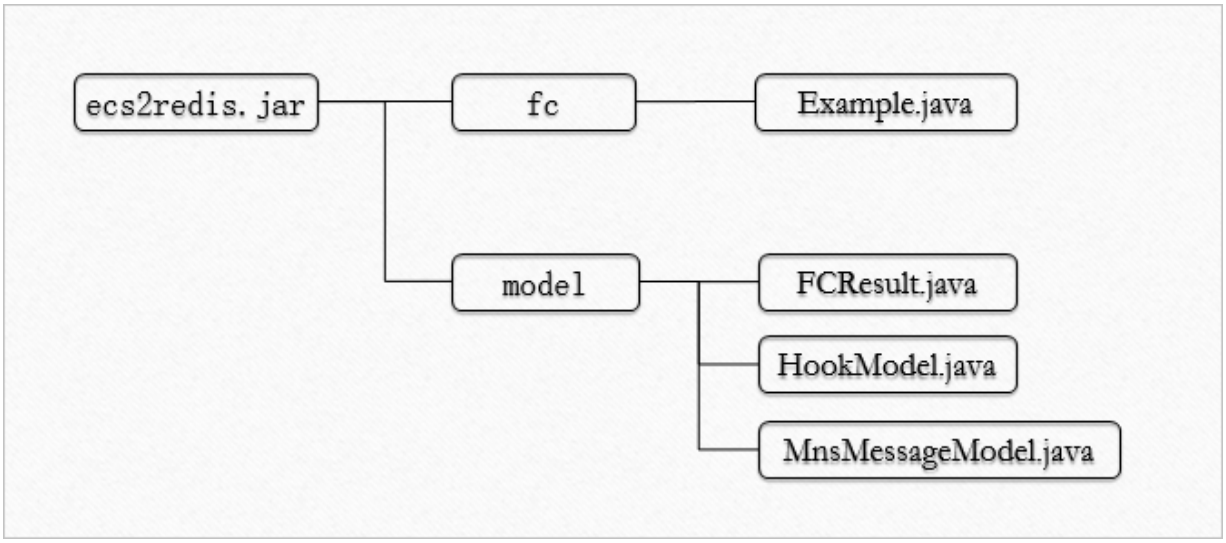


您还可以继续消费MNS队列中的消息，比如获取success、LifecycleHookId和LifecycleActionToken，编程提前结束生命周期挂钩。

上述最佳实践供您参考，您也在其它场景下通过多款产品实现自动化管理，从而更加灵活地管理伸缩组内的资源。

示例代码

示例代码仅供参考，请结合具体业务进行测试改造。主要功能涉及四个java文件，通过Maven管理，目录结构如下：



Maven依赖如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.aliyun.fc.wujin</groupId>
    <artifactId>demo</artifactId>
```

```
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>aliyun-java-sdk-ecs</artifactId>
            <version>4.10.1</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun.fc.runtime</groupId>
            <artifactId>fc-java-core</artifactId>
            <version>1.0.0</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>aliyun-java-sdk-core</artifactId>
            <version>3.2.6</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>aliyun-java-sdk-r-kvstore</artifactId>
            <version>2.0.3</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.25</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>4.2.5.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.apache.httpcomponents</groupId>
            <artifactId>httpclient</artifactId>
            <version>4.5.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>com.springsource.org.apache.commons.lang</
artifactId>
            <version>2.6.0</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun.mns</groupId>
            <artifactId>aliyun-sdk-mns</artifactId>
            <version>1.1.8.4</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.1.0</version>
                <configuration>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</
descriptorRef>
                    </descriptorRefs>
                    <appendAssemblyId>false</appendAssemblyId> <!-- 设
置jar文件的文件名中不包含Assembly Id -->
                </configuration>
```

```
                <executions>
                    <execution>
                        <id>make-assembly</id> <!-- 用于匹配需要合并的执行
目标 -->
                        <phase>package</phase> <!-- 设置在打包阶段执行jar
文件合并操作 -->
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

*Example.java*代码如下：

```
package fc;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.TypeReference;
import com.aliyun.fc.runtime.Context;
import com.aliyun.fc.runtime.StreamRequestHandler;
import com.aliyun.mns.client.CloudAccount;
import com.aliyun.mns.client.CloudQueue;
import com.aliyun.mns.client.MNSClient;
import com.aliyun.mns.model.Message;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.ecs.model.v20140526.DescribeInstancesRequest;
import com.aliyuncs.ecs.model.v20140526.DescribeInstancesResponse;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;
import com.aliyuncs.r_kvstore.model.v20150101.DescribeSecurityIpsR
equest;
import com.aliyuncs.r_kvstore.model.v20150101.DescribeSecurityIpsR
esponse;
import com.aliyuncs.r_kvstore.model.v20150101.ModifySecurityIpsRequest
;
import model.FCResult;
import model.HookModel;
import model.MnsMessageModel;
import org.apache.commons.codec.binary.Base64;
import org.apache.commons.lang.StringUtils;
import org.springframework.util.CollectionUtils;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
```

```
public class Example implements StreamRequestHandler {

    /**
     * 专有网络类型，此参数不用变
     */
    private static final String  VPC_NETWORK                  = "vpc";

    private static final String  CHAR_SET                     = "UTF-8
";

    /**
     * 接收input数组大小，4096通常够用
     */
    private static final Integer MAX_BYTE_LENGTH              = 4096;

    /**
     * REDIS 白名单默认分组
     */
    private static final String  DEFAULT_SECURITY_GROUP_NAME = "
default";

    /**
     * REDIS 修改白名单的模式
     */
    private static final String  MODIFY_MODE_APPEND           = "Append
";

    /**
     * MNS 客户端发送消息地址
     */
    private static final String  MNS_END_POINT                = "http
://%s.mns.%s.aliyuncs.com/";

    /**
     * 待添加的REDIS实例ID，根据个人情况替换
     */
    private static final String  REDIS_ID                     = "";

    /**
     * 接收本次函数计算执行结果的队列名称，根据个人情况替换
     */
    private static final String  QUEUE_NAME                   = "wujin-
fc-callback";

    /**
     * 阿里云账号UID，根据跟人情况替换
     */
    private static final Long    USER_ID                      =
1111111111111111111L;

    /**
     * 伸缩组 MNS FC 所属的region，根据个人情况替换
     */
    private static final String  REGION_ID                    = "cn-
hangzhou";

    @Override
    public void handleRequest(InputStream inputStream, OutputStream
outputStream, Context context) {
        FCResult result = new FCResult();
        String akId = context.getExecutionCredentials().getAccessKeyId
();
        String akSecret = context.getExecutionCredentials().getAccessK
eySecret();
```

```
        String securityToken = context.getExecutionCredentials().
getSecurityToken();
        try {
            //获取MNS触发函数计算时输入的内容
            String input = readInput(inputStream);
            MnsMessageModel mnsMessageModel = JSON.parseObject(input,
                    new TypeReference<MnsMessageModel>() {
                    });
            if (mnsMessageModel == null) {
                result.setSuccess(false);
                result.setMessage("mnsMessageModel is null");
                sendMns(akId, akSecret, securityToken, result.toString
());
                return;
            }
            HookModel contentModel = mnsMessageModel.getContent();
            if (contentModel == null) {
                result.setSuccess(false);
                result.setMessage("contentModel is null");
                sendMns(akId, akSecret, securityToken, result.toString
());
                return;
            }
            IAcsClient client = buildClient(akId, akSecret, securityTo
ken);
            //获取本次伸缩活动对应实例的私网IP
            List<String> privateIps = getInstancesPrivateIps(
contentModel.getInstanceIds(), client);
            if (CollectionUtils.isEmpty(privateIps)) {
                result.setSuccess(false);
                result.setMessage("privateIps is empty");
                sendMns(akId, akSecret, securityToken, result.toString
());
                return;
            }
            List<String> needAppendIps = filterPrivateIpsForAppend(
privateIps, client);
            if (!CollectionUtils.isEmpty(needAppendIps)) {
                modifySecurityIps(client, needAppendIps);
                result.setLifecycleHookId(contentModel.getLifecyc
leHookId());
                result.setLifecycleActionToken(contentModel.getLifecyc
leActionToken());
                sendMns(akId, akSecret, securityToken, result.toString
());
            }
        } catch (Exception ex) {
            result.setSuccess(false);
            result.setMessage(ex.getMessage());
            sendMns(akId, akSecret, securityToken, result.toString());
        }
    }

    /**
     * 构建请求 ECS Redis 接口客户端
     *
     * @param akId
     * @param akSecret
     * @param securityToken
     * @return
     */
    private IAcsClient buildClient(String akId, String akSecret,
String securityToken) {
```

```
        IClientProfile clientProfile = DefaultProfile.getProfile(
REGION_ID, akId, akSecret,
                securityToken);
        return new DefaultAcsClient(clientProfile);
    }

    /**
     * 将执行结果发送消息到MNS
     *
     * @param ak
     * @param aks
     * @param securityToken
     * @param msg
     */
    private void sendMns(String ak, String aks, String securityToken,
String msg) {
        MNSClient client = null;
        try {
            CloudAccount account = new CloudAccount(ak, aks,
                    String.format(MNS_END_POINT, USER_ID, REGION_ID),
securityToken);
            client = account.getMNSClient();
            CloudQueue queue = client.getQueueRef(QUEUE_NAME);
            Message message = new Message();
            message.setMessageBody(msg);
            queue.putMessage(message);
        } finally {
            if (client != null) {
                client.close();
            }
        }
    }

    /**
     * 过滤出需要添加到redis的私网IP
     *
     * @param privateIps 过滤以前的私网IP
     * @param client
     * @return
     * @throws ClientException
     */
    private List<String> filterPrivateIpsForAppend(List<String>
privateIps, IAcsClient client)
            throws ClientException {
        List<String> needAppendIps = new ArrayList<>();
        if (CollectionUtils.isEmpty(privateIps)) {
            return needAppendIps;
        }
        DescribeSecurityIpsRequest request = new DescribeSecurityIpsR
equest();
        request.setInstanceId(REDIS_ID);
        DescribeSecurityIpsResponse response = client.getAcsResponse(
request);
        List<DescribeSecurityIpsResponse.SecurityIpGroup> securityIp
Groups = response
                .getSecurityIpGroups();
        if (CollectionUtils.isEmpty(securityIpGroups)) {
            return privateIps;
        }
        for (DescribeSecurityIpsResponse.SecurityIpGroup securityIp
Group : securityIpGroups) {
            if (!securityIpGroup.getSecurityIpGroupName().equals(
DEFAULT_SECURITY_GROUP_NAME)) {
                continue;
```

```
            }
            String securityIps = securityIpGroup.getSecurityIpList();
            if (securityIps == null) {
                continue;
            }
            String[] securityIpList = securityIps.split(",");
            List<String> existIps = Arrays.asList(securityIpList);
            if (CollectionUtils.isEmpty(existIps)) {
                continue;
            }
            for (String ip : privateIps) {
                if (!existIps.contains(ip)) {
                    needAppendIps.add(ip);
                }
            }
        }
        return privateIps;
    }

    /**
     * 修改REDIS实例DEFAULT分组私网IP白名单
     *
     * @param client
     * @param needAppendIps
     * @throws ClientException
     */
    private void modifySecurityIps(IAcsClient client, List<String>
needAppendIps)
            throws ClientException {
        if (CollectionUtils.isEmpty(needAppendIps)) {
            return;
        }
        ModifySecurityIpsRequest request = new ModifySecurityIpsReq
uest();
        request.setInstanceId(REDIS_ID);
        String ip = StringUtils.join(needAppendIps.toArray(), ",");
        request.setSecurityIps(ip);
        request.setSecurityIpGroupName(DEFAULT_SECURITY_GROUP_NAME);
        request.setModifyMode(MODIFY_MODE_APPEND);
        client.getAcsResponse(request);
    }

    /**
     * 获取输入，并base64解码
     *
     * @param inputStream
     * @return
     * @throws IOException
     */
    private String readInput(InputStream inputStream) throws
IOException {
        try {
            byte[] bytes = new byte[MAX_BYTE_LENGTH];
            int tmp;
            int len = 0;
            //循环读取所有内容
            while ((tmp = inputStream.read()) != -1 && len <
MAX_BYTE_LENGTH) {
                bytes[len] = (byte) tmp;
                len++;
            }
            inputStream.close();
            byte[] act = new byte[len];
            System.arraycopy(bytes, 0, act, 0, len);
```

```
            return new String(Base64.decodeBase64(act), CHAR_SET);
        } finally {
            inputStream.close();
        }
    }

    /**
     * 获取实例列表对应的私网IP，并限制每次请求实例数量不超过100
     *
     * @param instanceIds 实例列表
     * @param client 请求客户端
     * @return
     * @throws Exception
     */
    public List<String> getInstancesPrivateIps(List<String> instanceId
s, IAcsClient client)
            throws Exception {
        List<String> privateIps = new ArrayList<>();
        if (CollectionUtils.isEmpty(instanceIds)) {
            return privateIps;
        }
        int size = instanceIds.size();
        int queryNumberPerTime = 100;
        int batchCount = (int) Math.ceil((float) size / (float)
queryNumberPerTime);
        //support 100 instance
        for (int i = 1; i <= batchCount; i++) {
            int fromIndex = queryNumberPerTime * (i - 1);
            int toIndex = Math.min(queryNumberPerTime * i, size);
            List<String> subList = instanceIds.subList(fromIndex,
toIndex);
            DescribeInstancesRequest request = new DescribeIn
stancesRequest();
            request.setInstanceIds(JSON.toJSONString(subList));
            DescribeInstancesResponse response = client.getAcsResponse
(request);
            List<DescribeInstancesResponse.Instance> instances =
response.getInstances();
            if (CollectionUtils.isEmpty(instances)) {
                continue;
            }
            for (DescribeInstancesResponse.Instance instance :
instances) {
                String privateIp = getPrivateIp(instance);
                if (privateIp != null) {
                    privateIps.add(privateIp);
                }
            }
        }
        return privateIps;
    }

    /**
     * 从 DescribeInstancesResponse.Instance 中解析出私网 IP
     *
     * @param instance DescribeInstancesResponse.Instance
     */
    private String getPrivateIp(DescribeInstancesResponse.Instance
instance) {
        String privateIp = null;
        if (VPC_NETWORK.equalsIgnoreCase(instance.getInstanceNetworkTy
pe())) {
            DescribeInstancesResponse.Instance.VpcAttributes
vpcAttributes = instance
```

```
                    .getVpcAttributes();
            if (vpcAttributes != null) {
                List<String> privateIpAddress = vpcAttributes.
getPrivateIpAddress();
                if (!CollectionUtils.isEmpty(privateIpAddress)) {
                    privateIp = privateIpAddress.get(0);
                }
            }
        } else {
            List<String> innerIpAddress = instance.getInnerIpAddress
();
            if (!CollectionUtils.isEmpty(innerIpAddress)) {
                privateIp = innerIpAddress.get(0);
            }
        }
        return privateIp;
    }
}
```

*FCResult.java*代码如下：

```
package model;

import com.alibaba.fastjson.JSON;

public class FCResult {

  private boolean success = true;

  private String  lifecycleHookId;

  private String  lifecycleActionToken;

  private String  message;

  public boolean isSuccess() {
      return success;
  }

  public void setSuccess(boolean success) {
      this.success = success;
  }

  public String getLifecycleHookId() {
      return lifecycleHookId;
  }

  public void setLifecycleHookId(String lifecycleHookId) {
      this.lifecycleHookId = lifecycleHookId;
  }

  public String getLifecycleActionToken() {
      return lifecycleActionToken;
  }

  public void setLifecycleActionToken(String lifecycleActionToken) {
      this.lifecycleActionToken = lifecycleActionToken;
  }

  public String getMessage() {
      return message;
  }
```

```
  public void setMessage(String message) {
      this.message = message;
  }

  @Override
  public String toString() {
      return JSON.toJSONString(this);
  }
}
```

*HookModel.java*代码如下：

```
package model;

import java.util.List;

public class HookModel {

    private String            lifecycleHookId;

    private String            lifecycleActionToken;

    private String            lifecycleHookName;

    private String            scalingGroupId;

    private String            scalingGroupName;

    private String            lifecycleTransition;

    private String            defaultResult;

    private String            requestId;

    private String            scalingActivityId;

    private List<String>      instanceIds;

    public String getLifecycleHookId() {
        return lifecycleHookId;
    }

    public void setLifecycleHookId(String lifecycleHookId) {
        this.lifecycleHookId = lifecycleHookId;
    }

    public String getLifecycleActionToken() {
        return lifecycleActionToken;
    }

    public void setLifecycleActionToken(String lifecycleActionToken) {
        this.lifecycleActionToken = lifecycleActionToken;
    }

    public String getLifecycleHookName() {
        return lifecycleHookName;
    }

    public void setLifecycleHookName(String lifecycleHookName) {
        this.lifecycleHookName = lifecycleHookName;
    }

    public String getScalingGroupId() {
```

```
        return scalingGroupId;
    }

    public void setScalingGroupId(String scalingGroupId) {
        this.scalingGroupId = scalingGroupId;
    }

    public String getScalingGroupName() {
        return scalingGroupName;
    }

    public void setScalingGroupName(String scalingGroupName) {
        this.scalingGroupName = scalingGroupName;
    }

    public String getLifecycleTransition() {
        return lifecycleTransition;
    }

    public void setLifecycleTransition(String lifecycleTransition) {
        this.lifecycleTransition = lifecycleTransition;
    }

    public String getDefaultResult() {
        return defaultResult;
    }

    public void setDefaultResult(String defaultResult) {
        this.defaultResult = defaultResult;
    }

    public String getRequestId() {
        return requestId;
    }

    public void setRequestId(String requestId) {
        this.requestId = requestId;
    }

    public String getScalingActivityId() {
        return scalingActivityId;
    }

    public void setScalingActivityId(String scalingActivityId) {
        this.scalingActivityId = scalingActivityId;
    }

    public List<String> getInstanceIds() {
        return instanceIds;
    }

    public void setInstanceIds(List<String> instanceIds) {
        this.instanceIds = instanceIds;
    }
}
```

*MnsMessageModel.java*代码如下：

```
package model;

public class MnsMessageModel {

    private String    userId;
```

```
    private String    regionId;

    private String    resourceArn;

    private HookModel content;

    public String getUserId() {
        return userId;
    }

    public void setUserId(String userId) {
        this.userId = userId;
    }

    public String getRegionId() {
        return regionId;
    }

    public void setRegionId(String regionId) {
        this.regionId = regionId;
    }

    public String getResourceArn() {
        return resourceArn;
    }

    public void setResourceArn(String resourceArn) {
        this.resourceArn = resourceArn;
    }

    public HookModel getContent() {
        return content;
    }

    public void setContent(HookModel content) {
        this.content = content;
    }
}
```