批量计算

用户指南

为了无法计算的价值 | 阿里云

用户指南

如何提交一个作业

A君： 我有一个python程序，我在本地可以这么运行:python test.py, 但是想要在云上运行怎么做？

test.py 内容如下：

```
print('Hello, cloud!')
```

云上运行大致过程： 您提交一个作业到BatchCompute，BatchCompute会按照你提供的配置去申请机器，启动虚拟机，在虚拟机中运行: python test.py, 得到结果后自动上传到OSS中。然后您可以去OSS中查看运行的结果。

1. 提交作业的方法有很多，下面列举4种：

(1). 使用命令行(一条命令提交作业):

```
bcs sub "python test.py" -p ./test.py
```

搞定！

这条命令会将test.py文件打包成 worker.tar.gz 上传到指定位置，然后再提交作业运行。

bcs命令需要先安装 batchcompute-cli工具才能使用， 请看这里。

- bcs sub命令：

```
bcs sub <commandLine> [job_name] [options]
```

更多参数详情可以通过”bcs sub -h”查看。

(2). 使用控制台提交作业:

下面列举详细解释步骤:

1). 将test.py打包上传到云端OSS

在test.py所在目录运行下面的命令:

```
tar -czf worker.tar.gz test.py # 将 test.py 打包到 worker.tar.gz
```

然后使用OSS控制台将 worker.tar.gz 上传到OSS。

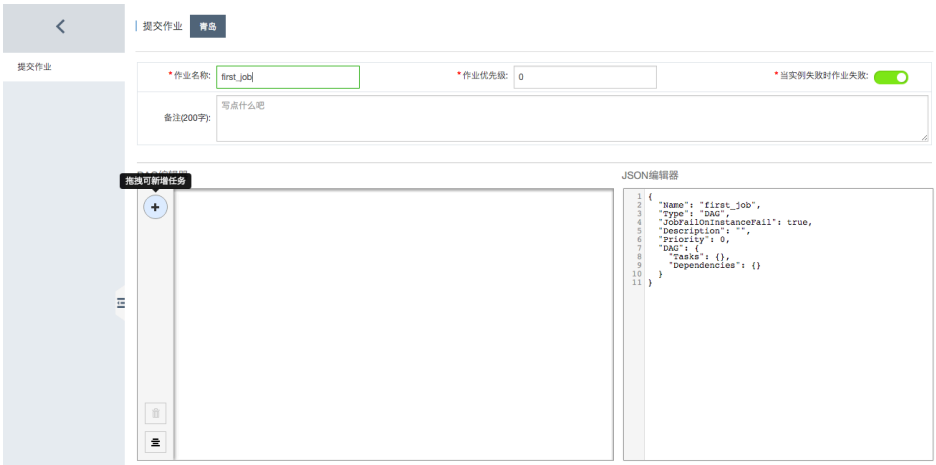
> 如果还没有开通OSS，请先开通.
>
> 还需要创建Bucket，假设创建了Bucket名称为 mybucket
>
> 然后在这个Bucket下创建一个目录: test

假设您上传到了mybucket这个Bucket下的test目录下，则OSS路径表示为：
oss://mybucket/test/worker.tar.gz

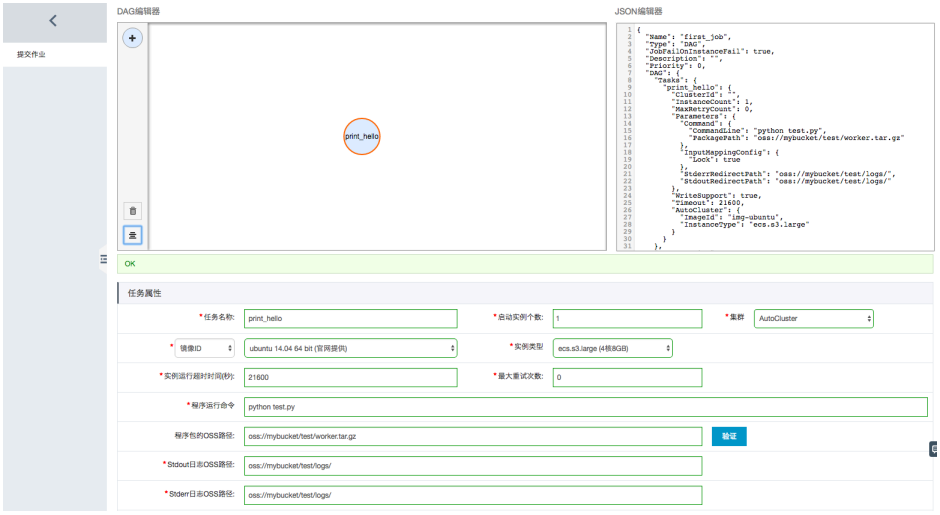
2). 使用控制台提交作业

打开 提交作业页面。

- 按照表单提示，填写作业名称： first_job



- 拖拽一个任务，按照下图填写表单, 其中ECS镜像ID可以从这里获取: 镜像



然后点击下面的”提交作业”按钮, 即可提交成功。

提交成功后，自动跳转到作业列表页面，您可以在这里看到你提交的作业状态。

等待片刻后作业运行完成，即可查看结果。

(3). 使用 Python SDK 提交作业

1) 将test.py打包上传到云端OSS

同上一节。

2) 提交作业

```
from batchcompute import Client, ClientError
from batchcompute import CN_SHENZHEN as REGION

ACCESS_KEY_ID = 'your_access_key_id' #需要配置
ACCESS_KEY_SECRET = 'your_access_key_secret' #需要配置

job_desc = {
"Name": "my_job_name",
"Description": "hello test",
"JobFailOnInstanceFail": true,
"Priority": 0,
"Type": "DAG",
"DAG": {
"Tasks": {
"test": {
"InstanceCount": 1,
"MaxRetryCount": 0,
"Parameters": {
"Command": {
```

```
"CommandLine": "python test.py",
"PackagePath": "oss://mybucket/test/worker.tar.gz"
},
"StderrRedirectPath": "oss://mybucket/test/logs/",
"StdoutRedirectPath": "oss://mybucket/test/logs/"
},
"Timeout": 21600,
"AutoCluster": {
"InstanceType": "bcs.a2.large",
"ImageId": "img-ubuntu"
}
}
},
"Dependencies": {}
}
}

client = Client(REGION, ACCESS_KEY_ID, ACCESS_KEY_SECRET)
result = client.create_job(job_desc)
job_id = result.Id

....
```

3) 更多关于Python SDK内容

请看 Python SDK

(4). 使用 Java SDK 提交作业

1) 将test.py打包上传到云端OSS

同上一节。

2) 提交作业

```
import com.aliyuncs.batchcompute.main.v20151111.*;
import com.aliyuncs.batchcompute.model.v20151111.*;
import com.aliyuncs.batchcompute.pojo.v20151111.*;
import com.aliyuncs.exceptions.ClientException;

public class SubmitJob{

String REGION = "cn-shenzhen";
String ACCESS_KEY_ID = ""; //需要配置
String ACCESS_KEY_SECRET = ""; //需要配置

public static void main(String[] args) throws ClientException{
JobDescription desc = new SubmitJob().getJobDesc();

BatchCompute client = new BatchComputeClient(REGION, ACCESS_KEY_ID, ACCESS_KEY_SECRET);
CreateJobResponse res = client.createJob(desc);
String jobId = res.getJobId();
```

```
//...
}

private JobDescription getJobDesc() {
JobDescription desc = new JobDescription();

desc.setName("testJob");
desc.setPriority(1);
desc.setDescription("JAVA SDK TEST");
desc.setType("DAG");
desc.setJobFailOnInstanceFail(true);

DAG dag = new DAG();

dag.addTask(getTaskDesc());

desc.setDag(dag);
return desc;
}

private TaskDescription getTaskDesc() {
TaskDescription task = new TaskDescription();

task.setClusterId(gClusterId);
task.setInstanceCount(1);
task.setMaxRetryCount(0);
task.setTaskName("test");
task.setTimeout(10000);

AutoCluster autoCluster = new AutoCluster();
autoCluster.setImageId("img-ubuntu");
autoCluster.setInstanceType("bcs.a2.large");
// autoCluster.setResourceType("OnDemand");

task.setAutoCluster(autoCluster);

Parameters parameters = new Parameters();
Command cmd = new Command();
cmd.setCommandLine("python test.py");
// cmd.addEnvVars("a", "b");

cmd.setPackagePath("oss://mybucket/test/worker.tar.gz");
parameters.setCommand(cmd);
parameters.setStderrRedirectPath("oss://mybucket/test/logs/");
parameters.setStdoutRedirectPath("oss://mybucket/test/logs/");
// InputMappingConfig input = new InputMappingConfig();
// input.setLocale("GBK");
// input.setLock(true);
// parameters.setInputMappingConfig(input);

task.setParameters(parameters);

// task.addInputMapping("oss://my-bucket/disk1/", "/home/admin/disk1/");
// task.addOutputtMapping("/home/admin/disk2/", "oss://my-bucket/disk2/");
// task.addLogMapping( "/home/admin/a.log","oss://my-bucket/a.log");
```

```
return task;
}
}
```

3) 更多关于Java SDK内容

请看 Java SDK

2: 关于批量计算中的CommandLine:

- CommandLine不等同于SHELL，仅支持”程序+参数”方式，比如”python test.py”或”sh test.sh”。
- 如果你想要执行SHELL，可以使用”/bin/bash -c ‘cd /home/xx/ && python a.py’ ”
- 或者将SHELL写到一个sh脚本中如：test.sh, 然后用”sh test.sh”执行。

CommandLine具体位置:

- 命令行工具中 bcs sub <cmd> [job_name] [options] 的cmd.
- 使用java sdk时 cmd.setCommandLine(cmd)中的cmd.
- python sdk中的 taskName.Parameters.Command.CommandLine.

A君： 我在OSS上有10G的数据, 想要在BatchCompute中使用，有没有方法挂载为虚拟机上的一个目录，我可以像访问本地文件一样使用数据？

假设 OSS目录为: oss://mybucket/mydir/， 这个目录下有10G的数据。

你可以将这个OSS目录挂载到 /home/admin/mydir/ 目录下，任务程序可以把它当做本地目录来处理。

- 注意：如果使用windows镜像，只能挂载到盘符， 如: “E:”，而不能挂载到文件夹.

1. 挂载数据目录

提交作业的时候，可以配置挂载，请看下面的例子。

(1). 使用 Java SDK

```
TaskDescription taskDesc = new TaskDescription();
taskDesc.addInputMapping("oss://mybucket/mydir/", "/home/admin/mydir/"); //只读挂载
taskDesc.addOutputMapping("/home/admin/mydir/", "oss://mybucket/mydir/"); //可写挂载
```

(2). 使用 Python SDK

```
# 只读挂载
job_desc['DAG']['Tasks']['my-task']['InputMapping'] = {
"oss://mybucket/mydir/": "/home/admin/mydir/"
}

# 可写挂载
job_desc['DAG']['Tasks']['my-task']['OutputMapping'] = {
"/home/admin/mydir/": "oss://mybucket/mydir/"
}
```

(3). 使用命令行

```
bcs sub "python main.py" -r oss://mybucket/mydir/:/home/admin/mydir/   # 如果有多个映射，请用逗号隔开
```

注意:

配置InputMapping，是只读挂载，意思是只能读，不能写，不能删除。

配置OutputMapping，凡是写到 /home/admin/mydir/目录下的文件或目录，在任务运行完成后，会被自动上传到 oss://mybucket/mydir/ 下面。

2. 挂载程序目录

A君： 我在OSS上有个程序 main.py, 想要在BatchCompute中使用, 如何使用？

假如 main.py 在OSS上的路径为: oss://mybucket/myprograms/main.py。

可以挂载 oss://mybucket/myprograms/ 为 /home/admin/myprograms/，

然后指定运行命令行：python /home/admin/myprograms/main.py 即可。

(1). 使用 Java SDK

```
TaskDescription taskDesc = new TaskDescription();
taskDesc.addInputMapping("oss://mybucket/myprograms/", "/home/admin/myprograms/"); //只读挂载

Command cmd = new Command()
cmd.setCommandLine("python /home/admin/myprograms/main.py")

params.setCommand(cmd);
taskDesc.setParameters(params);
```

(2). 使用 Python SDK

```
# 只读挂载
job_desc['DAG']['Tasks']['my-task']['InputMapping'] = {
"oss://mybucket/myprograms/": "/home/admin/myprograms/"
}
job_desc['DAG']['Tasks']['my-task']['Parameters']['Command']['CommandLine']='python
/home/admin/myprograms/main.py'
```

(3). 使用命令行

```
bcs sub "python /home/admin/myprograms/main.py" -r oss://mybucket/myprograms/:/home/admin/myprograms/
```

3. InputMapping挂载限制说明

(1) OSS上存储对象的命名

- 单个文件名最长为255个字节，其他字符集编码按照折算成UTF-8编码后所实际占用的字节数量计算，通常一个汉字占用3个字节。
- 从根目录开始计算的组合路径+文件名长度最大为1023个字节。
- 合法的文件名仅按照UTF-8字符集进行定义，其他字符集需转换为UTF-8之后进行合法性检查。
- 支持UTF-8 0x80以上的所有字符。
- 不支持0x00-0x1F和0x7F，以及\ / : * ？ “ < > |字符。

(2) InputMapping挂载文件的访问权限

考虑到多个节点向OSS写入文件的一致性问题，InputMapping挂载服务本身针对OSS是只读行为，应用程序通过文件系统接口的操作，在*任何情况*下都不会修改OSS上对应文件的内容，也不会删除OSS上的对应文件。

所有对挂载目录的修改操作，都会缓存在本地，应用程序可以通过文件系统接口读取到修改后的内容，但是不会自动同步到OSS。在应用程序运行结束，unmount文件系统并停止挂载服务后，所有本地缓存的改动都会被放弃，接下来如果重新启动挂载服务并mount，那么本地看到文件同OSS上对应Object的内容一致，上次的改动不再保留。

在挂载目录中，OSS上已经存在的文件都会赋予读取、执行的权限，没有写入权限。应用程序可以执行读操作，但是修改、截断、重命名和删除等操作都会被拒绝。

在挂载目录中，应用程序可以自由地创建文件和文件夹，这些新创建出来的内容都会赋予读取、写入、和执行权限，应用程序可以修改、截断、重命名和删除这些文件。

对于Windows操作系统来说，系统认为在只读文件夹下的内容也是无法删除的。因此对于一个OSS上

已经存在的文件夹，应用程序可以在里面创建文件，也可以进行修改和截断操作，但是删除和重命名操作会被Windows系统禁止。比如\127.0.0.1\ossdata\bucket\dir 是OSS上已经存在的一个文件夹，应用程序挂载后又在dir文件夹里创建了一个文件file,那么这个\127.0.0.1\ossdata\bucket\dir\file文件是没办法删除和重命名的。但是如果在\127.0.0.1\ossdata\bucket\dir下创建文件夹local， 此时在\127.0.0.1\ossdata\bucket\dir\local中再创建其他的文件就都是可以删除和重命名的了，如\127.0.0.1\ossdata\bucket\dir\local\file。Linux操作系统中不存在这个问题。

(3) 挂载语言问题

OSS上所有的Object的名称都是用UTF-8编码来保存的，挂载服务本身可以对字符集进行转换，您需要在集群/作业的描述中指定应用程序使用的字符集，这样经过挂载服务的转换，应用程序才可以访问到正确的文件。

注意，应用程序使用的字符集和操作系统默认的字符集可能是不同的。例如工作在一个简体中文操作系统中的繁体程序，需要配置字符集为BIG5，这样挂载服务会自动将OSS上UTF-8的路径和文件名转换为BIG5编码，虽然从操作系统上观察到挂载盘中的文件名都是乱码，但是应用程序的访问是正确的。

这个字符集转换功能仅仅影响路径的文件名，对于文件的内容并没有作用。

(4) 文件锁

基于NFS的DOS Share和文件锁会影响性能，所以如果有频繁的IO操作，建议关闭文件锁(在通过mount挂载的时候增加nolock选项)。但是有些特定的应用程序如3DSMAX必须用的文件锁特性，如果缺失这个特性会导致应用程序执行不正确，在这个时候就需要打开文件锁选项。

(5) 其他约定

在应用程序运行期间，不应该修改OSS应用程序通过挂载正在访问的文件夹，否则可能引起冲突。任何对应用程序正在访问的数据做出的修改(包括但不限于删除、修改文件内容、截断文件、Append文件内容)都可能引发应用程序运行结果错误。

OSS上的容量近似无限，所以操作系统统计的当前磁盘利用率并没有意义，并不代表当前OSS上存储空间的使用状况。

可以同时分别挂载同一bucket中的多个目录到不同的位置。

可以同时分别挂载同一帐号下多个bucket中的内容到不同的位置。

同一时刻只支持挂载同一个帐号下的文件，不能同时挂载多个帐号下的内容。

阿里云文件存储（Network Attached Storage，后面简称 NAS）是面向阿里云 ECS 实例、HPC 和 Docker 等计算节点的文件存储服务，提供标准的文件访问协议，用户无需对现有应用做任何修改，即可使用具备无限容量及性能扩展、单一命名空间、多共享、高可靠和高可用等特性的分布式文件系统。

目前，批量计算的用户也可以在API和SDK中通过配置用户NAS相关信息使用NAS服务。

1. NAS相关

在批量计算中使用NAS，需要用户先了解NAS产品相关的一些概念，目前批量计算服务仅支持经典网络，创建NAS权限组和挂载点时建议选择经典网络。

- 文件系统（FileSystem）：文件系统是用户购买NAS的基本单元，存储容量，价格，Quota限制，挂载点均与文件系统绑定，了解如何创建文件系统；
- 权限组（AccessGroup）：权限组是 NAS 提供的白名单机制，通过向权限组内添加规则来允许 IP 地址或网段以不同的权限访问文件系统。每个挂载点都必须与一个权限组绑定，了解如何使用权限组；
- 挂载点（MountEntry）：挂载点是文件系统实例在专有网络或经典网络内的一个访问目标地址，每个挂载点都对应一个域名，用户 mount 时通过指定挂载点的域名来挂载对应的 NAS 文件系统到本地。了解如何创建挂载点

以上是用户在批量计算中使用NAS过程中最重要的三个概念，如果需要了解购买和使用NAS其他相关内容，请参考：NAS官方文档

2. 使用NAS

在批量计算中使用NAS，需要用户在NAS产品里创建文件系统，权限组，挂载点，并通过SDK在创建集群或提交作业时将这些信息提供给批量计算。

用户数据

以下文件系统名，权限组名，挂载点名只起示例作用，具体名称根据用户设定可能有所不同。

- 用户在NAS中已经创建名为nas_file_system（NAS文件系统名由NAS后端生成，通常为一组无意义的字符，这里为示例方便）的文件系统；
- 用户在NAS中已经创建名为nas_access_group的权限组(注意创建时需要选择网络类型为经典网络);
- 用户在nas_file_system文件系统中创建经典网络的挂载点，并绑定nas_access_group权限组，挂载点名由NAS系统生成，假设为0266e49fea，在挂载点管理界面查看挂载地址，一般类似0266e49fea-yio75.cn-beijing.nas.aliyuncs.com；

指定文件系统和权限组

用户在批量计算中使用NAS之前，需要批量计算创建集群时将VM添加到用户的权限组里，用户可以通过集群描述的Mounts.NAS.FileSystem和Mounts.NAS.AccessGroup字段将文件系统和权限组的信息注册到批量计算服务。

(1). 使用Python SDK

```
# nas_file_system和nas_access_group根据用户情况有所不同
# 以下是创建集群时指定NAS文件系统和权限组信息

cluster_desc['Configs']['Mounts']['NAS']['FileSystem'] = ['nas_file_system' , ]
cluster_desc['Configs']['Mounts']['NAS']['AccessGroup'] = ['nas_access_group' , ]

# 用户也可以在提交AutoJob时指定NAS文件系统和权限组信息
job_desc['DAG']['Tasks']['my-task']['AutoCluster']['Configs']['Mounts']['NAS']['FileSystem'] = ['nas_file_system' , ]
job_desc['DAG']['Tasks']['my-task']['AutoCluster']['Configs']['Mounts']['NAS']['AccessGroup'] = ['nas_access_group' ,
]
```

(2). 使用JAVA SDK

指定挂载点

批量计算在运行作业之前，根据用户作业描述提供的挂载信息自动将NAS挂载为本地目录，需要用户在Mounts.Entries中指定NAS挂载点到本地目录的映射。

(1). 使用Python SDK

```
# For Linux
job_desc['DAG']['Tasks']['my-task']['Mounts']['Entries'] = {
'Source': 'nas://0266e49fea-yio75.cn-beijing.nas.aliyuncs.com:/',
'Destination': '/home/admin/mydir/',
'WriteSupport': true,
}


# For Windows
job_desc['DAG']['Tasks']['my-task']['Mounts']['Entries'] = {
'Source': 'nas://0266e49fea-yio75.cn-beijing.nas.aliyuncs.com:/!',
'Destination': '/home/admin/mydir/',
'WriteSupport': true,
}
```

(2). 使用JAVA SDK

注意:

- 指定挂载信息的时候，批量计算为区分挂载类型，MountEntry中的Source需要以nas://作为前缀

，后面接上挂载地址和NAS文件系统目录；
- 指定Windows系统挂载信息时，MountEntry中的Source还需要在目录后面加上!，Windows挂载还有其他注意的细节，具体参看：Windows挂载注意事项；
- 用户也可以在用户程序中自行挂载（上节中指定文件系统和权限组的步骤必不可少），具体文档请参考：手动挂载；

3. NAS挂载注意事项

1. 权限组容量

目前NAS单个权限组最多只支持300条规则，目前批量计算服务只支持经典网络，一台VM对应一台权限规则，建议用户同时使用NAS的集群实例总和不要超过额度，否则行为未定义。

2. NAS服务收费

在批量计算中使用NAS服务不收取额外费用，由NAS服务进行计价，收费及计价标准请参考：NAS收费及价格。

3. Cluster Mounts和Job Mounts优先级

细心的用户会发现创建集群和提交作业时均支持在Mounts字段中指定挂载NAS文件系统到本地目录，批量计算目前优先级如下：

- 作业中的Mounts会覆盖掉集群Mounts中的MountEntry（包括AutoCluster），但是对Cluster Mounts中的文件系统和权限组信息不会有影响；
- 作业结束后，集群级别的MountEntry不会恢复；

1. 指定磁盘

提交作业或者创建集群时，可以指定VM系统盘的大小和类型，另外还可以指定挂载一块数据盘（可选）。

(1) 提交作业JSON时指定磁盘

指定方法，以提交作业JSON为例, 在每个task中的AutoCluster字段中定义:

```
{
...
"DAG": {
"Tasks": {
"taskName": {
"AutoCluster": {
"InstanceType": "",
"ImageId": "",
```

```
"ECSImageId": "",
"Configs": {
"Disks": {
"SystemDisk": {
"Type": "ephemeral",
"Size": 500
},
"DataDisk": {
"Type": "ephemeral",
"Size": 500,
"MountPoint": "/path/to/mount"
}
}
}
}
...
}
}
}
...
}
```

- 目前SystemDisk和DataDisk类型需要配置为一样的，比如：SystemDisk 的Type是cloud，DataDisk的Type也必须是cloud。
- 数据盘必须指定MountPoint，linux下可以挂载到目录， window下只能挂载到驱动,如E盘：” E:”

(2) 命令行提交作业时指定磁盘

```
bcs sub "echo 123" --disk system:ephemeral:40,data:ephemeral:50:/home/disk1
```

系统盘配置格式: system:[cloud|ephemeral]:[40-500], 举例: system:cloud:40, 表示系统盘挂载40GB的云盘.

数据盘配置格式: data:[cloud|ephemeral]:[5-2000]:[mount-point], 举例: data:cloud:5:/home/disk1, 表示挂载一个5GB的云盘作为数据盘, window下只能挂载到驱动,如E盘:” data:cloud:5:E” .

注意: 数据盘使用ephemeral的时候,size取值范围限制为:[5-1024]GB.

另外，可以只指定系统盘：

```
bcs sub "echo 123" --disk system:ephemeral:40
```

当然，也可以只指定数据盘：

```
bcs sub "echo 123" --disk data:cloud:50:/home/disk1
```

2. 可用磁盘类型

BatchCompute服务每个region支持的磁盘类型不尽相同，如下表所示。

Region	ECS实例类型可用磁盘类型	BCS实例类型可用磁盘类型
青岛(华北1)	cloud, ephemeral	-
杭州(华东1)	cloud	ephemeral
深圳(华南1)	-	ephemeral
北京(华北2)	-	ephemeral

注意：以bcs为前缀的实例类型，目前只支持载指定大小的数据盘，如下表所示。

实例规格	支持数据盘大小
bcs.a2.large	400GB
bcs.a2.xlarge	400GB
bcs.a2.3xlarge	400GB

提交作业或者创建集群时，需要指定BatchCompute镜像ID，BatchCompute系统将使用您指定的镜像来启动VM。

BatchCompute镜像，有别于ECS公共镜像，来源有2种：

- 直接使用批量计算服务官方提供的镜像，详见下文。
- 在云市场购买批量计算服务指定的镜像，制作用户自定义镜像。

1. 官方提供的镜像

BatchCompute 提供以下基础镜像供大家直接使用：

青岛(华北1)：

操作系统	镜像ID	ECS镜像ID	系统详情
Windows	img-windows	m-281zatqcs	Windows Server 2008 R2 企业版 64位 中文
Ubuntu	img-ubuntu	m-28lyfn0tr	Ubuntu 14.04 64位, python 2.7, jdk 1.7, Docker 1.10.1
Centos	img-centos	m-28rvocix1	Centos 6.5 64位 ，python 2.6.6

北京(华北2)：

操作系统	镜像ID	ECS镜像ID	系统详情
Windows	img-windows	m-25gj9pkxg	Windows Server 2008 R2 企业版 64位 中文
Ubuntu	img-ubuntu	m-25useywoa	Ubuntu 14.04 64位, python 2.7, jdk 1.7, Docker 1.10.1
Centos	img-centos	m-25mdm57m0	Centos 6.5 64位 ，python 2.6.6

深圳(华南1)：

操作系统	镜像ID	ECS镜像ID	系统详情
Windows	img-windows	m-94dunmll8	Windows Server 2008 R2 企业版 64位 中文
Ubuntu	img-ubuntu	m-94luqb1cb	Ubuntu 14.04 64位, python 2.7, jdk 1.7, Docker 1.10.1
Centos	img-centos	m-94gjevbrq	Centos 6.5 64位 ，python 2.6.6

杭州(华东1)：

操作系统	镜像ID	ECS镜像ID	系统详情
Windows	img-windows	m-23ubsvddu	Windows Server 2008 R2 企业版 64位 中文
Ubuntu	img-ubuntu	m-2309yqb5n	Ubuntu 14.04 64位, python 2.7, jdk 1.7, Docker 1.10.1
Centos	img-centos	m-23jwitrt7	Centos 6.5 64位 ，python 2.6.6

如果这些镜像无法满足需求，您可参考 **自定义镜像** 步骤，自行制作镜像。

2. 如何使用镜像

(1) 获取可用的镜像

- 使用控制台可用直接查看可用的镜像列表:

- 使用 Java SDK 请看获取镜像列表.
- 使用 Python SDK 请看获取镜像列表.
- 使用命令行： bcs i.

（2）创建集群时指定

- 使用控制台创建集群时指定指定镜像ID：

- 使用 Java SDK 请看 如何创建集群.
- 使用 Python SDK 请看 如何创建集群.
- 使用命令行请看 如何使用集群.

(3) 创建(提交)作业时指定

- 使用控制台创建(提交)作业时指定镜像ID：

- 使用 Java SDK 请看 如何创建(提交)作业.
- 使用 Python SDK 请看 如何创建(提交)作业.
- 使用命令行请看 如何提交作业.

如果官方提供的镜像无法满足您的需求，您可参考以下步骤自行制作镜像(自行安装需要的软件)。**注意：自定义镜像的基础镜像，必须是批量计算服务在云市场发布的指定镜像，不能直接使用ECS公共镜像。**

制作自定义镜像需要以下几步:

1. 购买BatchCompute指定镜像并启动ECS实例。
2. 安装需要的软件。
3. 生成(创建)ECS镜像。
4. 注册BatchCompute镜像。
5. 释放ECS实例。

下面详细介绍各个步骤：

1. 购买BatchCompute指定镜像并启动ECS实例

请直接点击相应的基础镜像链接：

- Windows Server 2008
- Ubuntu 14.04

点击“立即购买”，进入购买页面，按照下面的建议填写表单：

- 付费类型选择“按量付费”。
- 地域与批量计算服务保持一致。
- 可用区随机分配。
- 网络类型选择“经典网络”。
- 选择一个已有的安全组或创建一个新安全组。
- 实例规格建议选择4核8GB即可, 选择 I/O 优化实例（必选）。
- 带宽选择固定带宽并将峰值按需调节（目前流入ECS实例的流量不收费，流出收费）。
- 系统盘默认40G（可以按需选择合适的系统盘大小，请勿添加数据盘)。
- 设置管理员（windows是Administrator，linux是root）密码，并记住密码。

表单填写好后，点击“立即购买”完成实例购买（注意：目前不支持子账户购买）。

实例创建出来后，重置实例密码并重启实例。

待实例处于运行状态后，通过VNC连接，登入到该实例。配置详情参见ECS文档启动实例。

2. 安装需要的软件

您可以安装自己需要的其他应用程序，如用于渲染的maya等。

3. 生成(创建)ECS镜像

待所有程序安装完成后，在ECS控制台中创建磁盘快照。参见ECS文档创建快照。

使用上面创建的快照，在ECS控制台中创建自定义镜像。参见ECS文档创建自定义镜像。

- 注意：请勿使用带有数据盘的ECS实例制作自定义镜像。

4. 注册镜像（支持2种方式，选一种方式即可）

(1) 共享自定义镜像给BatchCompute

将制作的ECS镜像共享给batchcompute@aliyun-inner.com（UID为1190847048572539 ）。参见ECS文档共享镜像。

(2) 注册(创建)BatchCompute镜像（推荐）

打开批量计算控制台创建镜像页面。将制作好的ECS镜像在BatchCompute服务处进行创建镜像操作，之后提交作业或创建集群均可以使用创建的镜像。镜像资源的相关操作可以参见下文。

注意：第一次自行创建镜像资源需要授权，请按照控制台创建镜像页面上的提示，进行一键授权。

除了控制台操作，您也可以使用Python SDK、Java SDK或命令行工具完成注册镜像操作，请查看相关文档，这里不再赘述。

5. 释放ECS实例

镜像制作完成后，需要释放ECS实例，因为是按量付费购买的，如果不释放会继续收费。

登录ECS控制台释放掉该实例即可。

批量计算在深圳(华南1)， 北京(华北2)，杭州（华东1）区域提供以下专属实例。专属实例只支持按量计费模式。

名称	CPU（核）	内存(GB)
bcs.a2.large	4	8
bcs.a2.xlarge	8	16
bcs.a2.3xlarge	16	32

除了专属实例，用户也能通过批量计算直接使用ECS SN1、SN2系列的独享实例。ECS独享实例的详细说明请参考这里。如果需要使用其他实例类型，请提工单咨询。目前批量计算只支持以竞价模式使用ECS实例。

任务程序环境变量

1. BatchCompute为用户任务程序提供以下的环境变量:

变量名	变量值
BATCH_COMPUTE_DAG_JOB_ID	作业ID,视实际情况而定
BATCH_COMPUTE_DAG_TASK_ID	任务名称,视实际情况而定

BATCH_COMPUTE_DAG_INSTANCE_ID	实例ID,视实际情况而定
BATCH_COMPUTE_OSS_HOST	OSS host,视实际情况而定
BATCH_COMPUTE_REGION	区域,视实际情况而定
BATCH_COMPUTE_CLUSTER_ID	cluser id
BATCH_COMPUTE_WORKER_ID	worker id

程序运行在 docker 容器中的环境变量稍有不同：

变量名	变量值
USER	root
PWD	/batchcompute/workdir
PATH	/sbin:/usr/sbin:/bin:/usr/bin, 注意没有/usr/local/bin; 如果要设置PATH，需要在提交作业时在EnvVars字段中指定
HOME	/root
BATCH_COMPUTE_DAG_JOB_ID	作业ID,视实际情况而定
BATCH_COMPUTE_DAG_TASK_ID	任务名称,视实际情况而定
BATCH_COMPUTE_DAG_INSTANCE_ID	实例ID,视实际情况而定
BATCH_COMPUTE_OSS_HOST	OSS host,视实际情况而定
BATCH_COMPUTE_REGION	区域,视实际情况而定

2. 如何使用

用户只需在任务运行程序中从环境变量中获取即可, 举例:

(1) python 程序中使用环境变量:

```
task_id = os.environ['BATCH_COMPUTE_DAG_TASK_ID']
instance_id = os.environ['BATCH_COMPUTE_DAG_INSTANCE_ID']
```

(2) java 程序中使用环境变量:

```
String taskId = System.getenv("BATCH_COMPUTE_DAG_TASK_ID");
String instanceId = System.getenv("BATCH_COMPUTE_DAG_INSTANCE_ID");
```

3. 自定义环境变量

除了系统提供的环境变量，你也可以在提交作业的时候设置新的环境变量。

(1) 使用 Python SDK

代码片段:

```
env = {
'k1': 'v1',
'k2': 'v2'
}
...
job_desc['DAG']['Tasks']['my-task']['Parameters']['Command']['EnvVars']=env
...
```

(2) 使用 Java SDK

代码片段:

```
Command cmd= new Command();
cmd.addEnvVars("k1","v1");
cmd.addEnvVars("k2","v2");
...

TaskDescription desc = TaskDescription();
Parameters parmas = new Parameters();
params.setCommand(cmd);
...
desc.setParameters(params);
```

(3) 使用命令行工具:

```
bcs sub "python main.py" -e k1:v1,k2:v2
```

默认限额配置如下表所示，如果对用户限额设置有更高的需求，请提交工单。

Quota项	默认值
单用户最大集群（Cluster）数（不包含已删除的Cluster）	5
单集群最大实例组（Group）数	1
单实例组最大申请的虚拟机资源数	20
单用户最大并发虚拟机实例数	20
单用户最大作业（Job）数（不包含已删除的Job）	100

单用户单作业最大任务（Task）数	30
单用户单任务最大实例（Instance）数	500
单用户单Instance最大超时时间（超时Instance会自动fail）	6小时

使用集群

使用AutoCluster还是Cluster？

用户提交作业时，如果指定一个Cluster ID，那么作业的任务运行时会被调度到这个Cluster中运行。如果没有指定集群，则可以使用AutoCluster配置，指定镜像和实例类型即可。任务运行时会自动创建相应的Cluster，运行完成后自动释放掉。

什么情况下应该使用Cluster？

如果你有很多作业(Job)要处理，可以考虑使用Cluster。

比如有100个作业要运行，你可以创建一个10台VM的Cluster，将100个作业全部提交到这个集群，然后只需等待即可, 系统会在每个任务完成后自动调度下一个任务运行。全部运行完成后，你需要手动释放(删除)掉集群。这样可以节省时间和费用。

什么情况下使用AutoCluster?

AutoCluster在提交作业时指定需要的实例数和实例规格，实际运行任务的时候系统自动创建集群，运行任务完成后自动释放。不在乎等待时间长，或者作业较少情况下，可以使用AutoCluster。

1. 区别

.	AutoCluster	Cluster
创建	作业启动时自动创建	需要事先创建集群，创建集群时需要指定ImageId和InstanceType，还有需要的机器台数。
释放	作业完成后自动释放	需要手动删除. 如果您不再使用集群，请删除，不然会一直收费。
使用	提交作业时指定ImageId和InstanceType，还有需要的机器台数	提交作业时指定集群ID

2. 如何使用 Cluster

(1) 使用 Python SDK

```
# 使用clusterId就无需使用AutoCluster
job_desc['DAG']['Tasks']['my-task']['ClusterId'] = "cls-xxxxxxx"
```

(2) 使用 Java SDK

以下是代码片段:

```
// 使用clusterId就无需使用AutoCluster
desc.setClusterId("cls-xxxxxxx");
```

(3) 使用命令行

```
# 使用 clusterId
bcs sub "python main.py" -c cls-0101010299123
```

3. 如何使用 AutoCluster

(1) 使用 Python SDK

```
...
autoCluster = {
'ImageId': 'img-ubuntu',
'InstanceType': 'bcs.a2.large'
}
...
job_desc['DAG']['Tasks']['my-task']['AutoCluster'] = autoCluster
...
```

(2) 使用 Java SDK

以下是代码片段:

```
AutoCluster autoCluster = new AutoCluster();
autoCluster.setImageId("img-ubuntu");
autoCluster.setInstanceType("bcs.a2.large");

TaskDescription desc = new TaskDescription();
```

```
desc.setAutoCluster(autoCluster);
```

(3) 使用命令行

```
# 使用 Auto Cluster
bcs sub "python main.py" -c img=img-ubuntu:type=bcs.a2.large
```

批量计算服务支持一个作业包含多个任务，任务之间可以有DAG依赖关系。

即前面的任务运行完成(Finished)后, 后面的任务才开始运行。

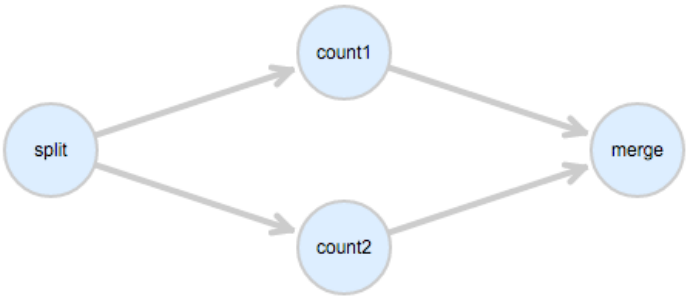
例1:

在 job description json 中这样描述:

```
{
  "Name": "my-job",
  "DAG": {
    ...
    "Dependencies": {
      "split": ["count"],
      "count": ["merge"]
    }
  }
}
```

- split运行完成后，count开始运行，count完成后，merge才开始运行。
- merge运行完成，整个作业结束。

例2:



在 job description json 中这样描述:

```
{
  "Name": "my-job",
  "DAG": {
    ...
    "Dependencies": {
      "split": ["count1","count2"],
      "count1": ["merge"],
      "count2": ["merge"]
    }
  }
}
```

- split运行完成后，count1和count2同时开始运行，count1和count2都完成后，merge才开始运行。
- merge运行完成，整个作业结束。

一个作业(Job)中可以有多个任务(Task)，一个任务可以指定在多个实例(Instance)上运行程序。

如何并发？

请看下面 job description json 例子：

```
{
"DAG": {
...
"Tasks": {
...
"count": {
"InstanceCount": 3, //指定需要实例数：3台VM
"LogMapping": {},
"AutoCluster": {
"ResourceType": "OnDemand",
"ImageId": "img-ubuntu",
"InstanceType": "bcs.a2.large"
},
"Parameters": {
"Command": {
```

```
"EnvVars": {},
"CommandLine": "python count.py",
"PackagePath": "oss://your-bucket/log-count/worker.tar.gz"
},
"InputMappingConfig": {
"Lock": true
},
"StdoutRedirectPath": "oss://your-bucket/log-count/logs/",
"StderrRedirectPath": "oss://your-bucket/log-count/logs/"
},
"OutputMapping": {},
"MaxRetryCount": 0,
"Timeout": 21600,
"InputMapping": {}
}
}
},
"Description": "batchcompute job",
"Priority": 0,
"JobFailOnInstanceFail": true,
"Type": "DAG",
"Name": "log-count"
}
```

任务count中配置了InstanceCount为3， 表示需要实例数3台, 即在3台VM上运行这个任务的程序。

并发处理不同片段的数据

3台VM上运行的任务程序都是一样的，如何让它处理不同的数据呢？

在任务程序中使用环境变量: BATCH_COMPUTE_DAG_INSTANCE_ID(实例ID) 来区分，可以处理不同片段的数据。

以下是 count.py 代码片段:

```
...
# instance_id: should start from 0
instance_id = os.environ['BATCH_COMPUTE_DAG_INSTANCE_ID']

...

filename = 'part_%s.txt' % instance_id
...

# 1. download a part
oss_tool.download_file('%s/%s/%s.txt' % (pre, split_results, instance_id ), filename)



...
# 3. upload result to oss
upload_to = '%s/count_results/%s.json' % (pre, instance_id )
```

```
print('upload to %s' % upload_to)
oss_tool.put_data(json.dumps(m), upload_to)
...
```

完整例子请看快速开始例子。

Docker

BatchCompute除了支持把软件直接安装到ECS镜像，还支持通过Docker镜像部署应用程序。

您可以自定义制作一个Docker镜像，使用registry工具上传到阿里云OSS，然后您可以指定您的作业的任务在这个镜像中运行。

1. BatchCompute对Docker支持的原理

这里要和普通VM支持对比说明一下：

(1) 先说使用普通VM, 用户提交作业，每个作业可以有多个任务，每个任务指定一个镜像（支持Linux 和 Window）,系统运行这个任务时，会根据指定的镜像启动VM，用户任务将运行在这个VM上。任务完成后结果会被上传到指定的OSS目录，VM销毁。然后执行下一个任务。

(2) 使用Docker：每个任务也可以指定使用一个Docker容器镜像来运行。运行task时，会先启动一个VM运行支持Docker的系统镜像（如：支持Docker的Ubuntu）,然后会从OSS上下载你指定的Docker镜像，在这个VM中启动起来，用户的任务将运行在这个Docker容器内。任务完成后结果上传到指定的OSS目录，VM销毁。然后执行下一个任务。

> 目前一个VM，只支持运行一个Docker镜像。

```
# 使用VM:
---
|-- job
|-- task
|-- VM (用户指定的VM，支持Windows和Linux)
|-- program (用户程序)

# 使用Docker模式:
---
|-- job
|-- task
|-- VM (支持docker的Ubuntu)
|-- Docker-Container(用户指定的Docker的容器镜像)
|-- program (用户程序)
```

2. 使用Docker和不使用Docker有什么区别

-	*不使用Docker*	*使用 Docker*
使用镜像	指定ECS镜像ID	指定支持Docker Container的ECS镜像ID（官网提供的Ubuntu），还需指定自定义Docker镜像。
程序运行平台	支持Window和Linux	支持Linux
本地调试	不支持本地调试	镜像在本地制作，支持本地调试

3. 安装Docker

(1) 在 Ubuntu 上安装(推荐使用)

```
sudo apt-get update  #更新软件源
sudo apt-get install docker.io #安装docker


#检查是否成功
sudo docker ps
```

可以看到类似下面的结果：

> Ubuntu下安装好docker后，每次都要加sudo运行比较影响效率。有没有好办法？

您可以通过加入docker组，即可免sudo。

```
sudo gpasswd -a ${USER} docker # 将当前用户加入docker组， 如果没有docker组，加一个：sudo groupadd docker
sudo service docker restart # 重启docker服务，然后注销后重新登录系统即可
```

(2) 在 window/Mac 上安装

从官网下载docker toolbox:

https://www.docker.com/docker-toolbox

安装完成后会有2个快捷方式：

> Kitematic: 用来管理docker container的图形化界面

Docker Quickstart Terminal: 可以快速启动 docker 命令行界面。

(3) 配置加速器

使用加速器将会提升您在国内获取Docker官方镜像的速度:阿里云容器服务开发者平台

4. 制作Docker镜像

本例中我们将制作一个 Ubuntu 镜像，内置python。镜像名称：myubuntu。

新建一个目录 dockerUbuntu,结构如下:

```
dockerUbuntu
|-- Dockerfile
```

文件 Dockerfile 的内容：

```
FROM ubuntu:14.04

# 这里要替换 your_name 为您的名字, 和your_email 为您的Email
MAINTAINER your_name <your_email>

# 更新源
RUN apt-get update

# 清除缓存
RUN apt-get autoclean

# 安装python
RUN apt-get install -y python

# 启动时运行这个命令
CMD ["/bin/bash"]
```

运行以下命令，build镜像:

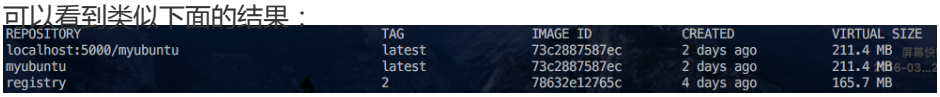
```
cd dockerUbuntu             #进入 dockerUbuntu 目录
docker build -t myubuntu ./ #正式build, 命名为 myubuntu
```

- 注意：docker 命令在ubuntu中默认需要加sudo才能运行，而在Mac/Windows中，需要从“Docker Quickstart Terminal”中启动的命令行工具中运行。

build 完成后, 运行以下命令查看:

```
docker images
```

可以看到类似下面的结果：



REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
localhost:5000/myubuntu	latest	73c2887587ec	2 days ago	211.4 MB
myubuntu	latest	73c2887587ec	2 days ago	211.4 MB
registry	2	78632e12765c	4 days ago	165.7 MB

制作docker镜像，除了Dockerfile这种方式外，还有更加直观的制作方式。

5. 将docker镜像上传到OSS

使用BatchCompute Docker服务，需要将制作的Docker镜像myubuntu上传到OSS，系统将会从OSS下载这个镜像来运行您的任务程序。

(1) 安装 OSS Docker Registry 2

假设您想要将docker存储到OSS的目录路径为:

```
oss://your-bucket/dockers/
```

我们将利用Docker Registry 2官方镜像创建一个私有镜像仓库，需要配置了OSS的Access Key ID, Access Key Secret, Region, Bucket等信息。

具体安装步骤如下：

先在当前目录生成文件config.yml

```
version: 0.1
log:
level: debug
storage:
oss:
accesskeyid: your_access_key_id
accesskeysecret: your_access_key_secret
region: oss-cn-qingdao
bucket: your-bucket
rootdirectory: dockers
secure: false
internal: false
http:
addr: 0.0.0.0:5000
```

其中的变量需要替换：

参数	描述
your_access_key_id	您的 access key id
your_access_key_secret	您的 access key secret
your-bucket	您的 bucket

关于OSS配置的详细信息请参见 Docker官方文档

然后运行下面的命令安装

```
docker pull registry:2
docker run -v `pwd`/config.yml:/etc/docker/registry/config.yml -p 5000:5000 --name registry -d registry:2
```

- 注意：这里region使用 oss-cn-qingdao, 表示使用青岛region的OSS，而后面提交作业也需要提交到相应的region才能正常工作。

完成后，可以运行以下命令查看：

```
docker ps      #查看运行的container
```

如果成功安装，可以看到 registry:2

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
2a5996992f4a        registry:2          "/bin/registry /etc/d"   2 days ago          Up 2 days           0.0.0.0:5000->5000/tcp   registry
```

(2) 运行下面命令，将镜像上传到 OSS:

```
docker tag myubuntu localhost:5000/myubuntu
docker push localhost:5000/myubuntu
```

注意：

1. 要用 localhost:5000/ 作为前缀，用其他的字符串上传不了。5000 端口是第(1)步中 -p 5000:5000 中(冒号前的5000)指定的。
2. 您制作的镜像名称为 localhost:5000/myubuntu,而不是 myubuntu。

检验是否上传成功, 可以直接使用OSS控制台查看是否有这个目录: oss://your-bucket/dockers/docker/registry/v2/repositories/myubuntu/

6. 使用Docker注意事项:

Docker container运行时，用户为root，path环境变量默认为：/sbin:/usr/sbin:/bin:/usr/bin。注意没有/usr/local/bin

PWD环境变量如果没有设置，则为’/batchcompute/workdir’。用户的程序包始终会被解压到/batchcompute/workdir。

使用ClusterID提交任务时， 因为Docker registry一旦启动后就不停止，因此提交到一个cluster中的所有job，其 BATCH_COMPUTE_DOCKER_REGISTRY_OSS_PATH 必须相同。

目前 InputMapping，OutputMapping 不能同时挂载到同一个目录。

BatchCompute启动Docker容器时使用—privileged=false模式，所以不允许在docker容器中启动docker容器。

Docker 提交作业例子

步骤

- 提交作业简介
- 作业准备
    - 上传数据文件到OSS
    - 准备任务程序
- 提交作业
    - 编写作业配置
    - 提交命令
- 查看作业运行状态
- 查看运行结果

1. 提交作业简介

在 BatchCompute 中, 提交作业时使用 docker 与普通作业基本相同, 只有2点区别：

(1) 使用支持Docker的ImageId

您只需要将任务描述的 ImageId 指定为 BatchCompute 的公共镜像的Id（支持Docker的镜像, ID为img-ubuntu），或者使用设置了该ImageId的 Cluster的ClusterId.

(2) 在task描述的环境变量(EnvVars)中增加如下两个参数:

字段名称	描述	是否可选
BATCH_COMPUTE_DOCKER_IMAGE	Docker镜像名称	可选
BATCH_COMPUTE_DOCKER_REGISTRY_OSS_PATH	Docker镜像在 OSS-Registry 中的存储路径	可选

- 如果没有 BATCH_COMPUTE_DOCKER_IMAGE 参数,表示不使用 docker ,这时

BATCH_COMPUTE_DOCKER_REGISTRY_OSS_PATH 将被忽略。
- 如果有 BATCH_COMPUTE_DOCKER_IMAGE, 则表示使用 docker。

2. 作业准备

本作业程序使用python编写，目的是统计一个日志文件中“INFO”,” WARN”,” ERROR”,” DEBUG”出现的次数。

该作业包含3个任务: split, count 和 merge

- split 任务会把日志文件分成 3 份。
- count 任务会统计每份日志文件中“INFO”,” WARN”,” ERROR”,” DEBUG”出现的次数 (count 任务需要配置InstanceCount为3，表示同时启动3个 count 任务)。
- merge 任务会把 count 的结果统一合并起来。

DAG图例:

(1) 上传数据文件到OSS

下载本例子所需的数据: log-count-data.txt

将 log-count-data.txt 上传到:

```
oss://your-bucket/log-count/log-count-data.txt
```

- your-bucket如表示您自己创建的bucket，本例子假设region为: cn-shenzhen.
- 如何上传到OSS，请参考OSS上传文档。

(2) 准备任务程序

本例子的作业程序是使用python编写的， 下载本例子所需的程序: log-count.tar.gz

使用下面的目录解压：

```
mkdir log-count && tar -xvf log-count.tar.gz -C log-count
```

解压后的log-count/目录结构如下

```
```

```
 log-count
|-- conf.py # 配置
|-- split.py # split 任务程序
|-- count.py # count 任务程序
|-- merge.py # merge 任务程序
```

- 注意：不需要改动程序

2. 提交作业

提交作业可以使用 python sdk 或者 java sdk, 或者控制台提交，本例子使用命令行工具提交。

(1) 编写作业配置

在log-count的父目录下创建一个文件: job.cfg(此文件要与log-count目录平级), 内容如下：

```
[DEFAULT]
job_name=log-count
description=demo
pack=./log-count/
deps=split->count;count->merge

[split]
cmd=python split.py

[count]
cmd=python count.py
nodes=3

[merge]
cmd=python merge.py
```

这里描述了一个多任务的作业，任务的执行顺序是 split->count->merge。

- 关于cfg格式的描述，请看多任务支持

(2) 提交命令

```
bcs sub --file job.cfg -r oss://your-bucket/log-count/:/home/input -w oss://your-bucket/log-count/:/home/output --docker localhost:5000/myubuntu@oss://your-bucket/dockers/
```

- -r 和 -w 表示只读挂载和可写映射，具体请看这里: OSS挂载
- 同一个oss路径，可以挂载到不同的本地目录。但是不同的oss路径是不能挂载到同一个本地目录的，一定要注意。
- —docker 表示使用docker，格式: image_name@storage_oss_path, 会自动将docker名称和仓库地址配置到环境变量。

- 注意：bcs 使用的region，一定要和docker所在region一致。

4. 查看作业运行状态

```
bcs j  # 获取作业列表, 每次获取作业列表后都会将列表缓存下来，一般第一个即是你刚才提交的作业
bcs ch 1 # 查看缓存中第一个作业的状态
bcs log 1 # 查看缓存中第一个作业日志
```

5. 查看结果

Job结束后，可以使用以下命令查看存在OSS中的结果。

```
bcs oss cat oss://your-bucket/log-count/merge_result.txt
```

内容应该如下：

```
{"INFO": 2460, "WARN": 2448, "DEBUG": 2509, "ERROR": 2583}
```

如果你想要使用Docker镜像本地调试一下程序， 可以根据本节内容操作，如果不需要本地调试，请跳过本节。

1. 任务程序可以使用的变量说明

在 BatchCompute 中,运行在 docker 容器中的环境和不使用 docker 容器时的环境变量稍微不同, 具体请看环境变量

2. 本地测试命令

在制作完成 docker 镜像后,您可以使用如下的命令进行本地测试。

```
docker run -it -v /home/local_folder:/batchcompute/workdir
-e BATCH_COMPUTE_DAG_INSTANCE_ID=<your_instance_id>
-e BATCH_COMPUTE_DAG_TASK_ID=<your_task_name>
-e BATCH_COMPUTE_DAG_JOB_ID=job-0000000000
-e BATCH_COMPUTE_OSS_HOST=<your_oss_host>
your_docker_image_name your_command
```

其中

```
-v /home/local_folder:/batchcompute/workdir 表示挂载本地/home/local_folder目录到 docker容器镜像中的
/batchcompute/workdir 目录

-e key=value 表示指定环境变量
```

```
your_task_name 作业中 task 的名称

your_job_name: 作业的名称

your_instance_id: 任务实例ID，从0开始递增的整数， 如这个任务你要启动3个实例来运行，则id分别为0,1,2

your_oss_host: OSS主机名（域名,应包含region信息，且不带"http://"前缀)

your_docker_image_name: 您制作的 docker 镜像名称,如 myubuntu

your_command:命令行及参数
```

举例:

假设您的本地程序路径： /home/admin/log-count/

```
docker run -it -v /home/admin/log-count/:/batchcompute/workdir -e BATCH_COMPUTE_INSTANCE_ID=0 -e
BATCH_COMPUTE_TASK_ID=split -e BATCH_COMPUTE_JOB_ID=job-0000000000 -e
BATCH_COMPUTE_OSS_HOST=oss-cn-shenzhen.aliyuncs.com myubuntu python /batchcompute/workdir/split.py
```

这个命令是在本地运行 myubuntu 这个docker镜像,将本地目录/home/admin/log-count/挂载到docker镜像的/batchcompute/workdir/目录，并在这个镜像里运行python /batchcompute/workdir/split.py命令。

注意：

- 本地的/home/admin/log-count/目录是程序所在目录， 目录中应当有split.py。
- BATCH_COMPUTE_INSTANCE_ID 从0开始，假如你配置该任务启动3个实例，则 BATCH_COMPUTE_INSTANCE_ID 分别为0,1,2。

除了使用Dockerfile的方式外，你还可以使用更直观的方式来制作镜像.

1. 运行基础镜像容器

```
docker run -it ubuntu
```

然后你会发现你已经以root身份进入ubuntu

```
root@0bab204d8f9b:/#
```

你可以在这里安装你想要的软件, 比如：

```
apt-get install python -y
apt-get install openjdk-7-jdk
....
```

安装完成后， 退出

```
exit
```

2. 制作镜像

```
docker ps -n 1  #列出最新container
```

找到对应的CONTAINER ID ， 如： 41570524e867

```
docker commit 41570524e867 myubuntu
```

完成后，可以使用以下命令查看是否成功。

```
docker images
```

消息通知

批量计算服务（BatchCompute）使用MNS提供的主题模式来实现消息通知。用户负责主题（Topic）的创建、管理和订阅，并在使用BatchCompute创建集群或提交作业时指定主题相关的配置。BatchCompute依据配置向指定用户主题推送消息。用户可在MNS控制台配置URL、队列、邮件和短信四种方式获取消息通知。目前，BatchCompute主要支持两大类消息事件，即集群事件和作业事件。

1. 前期准备

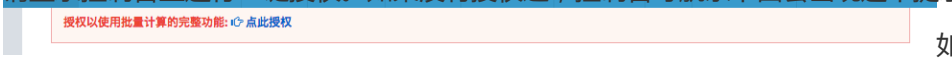
(1) 开通消息服务

(2) 创建MNS主题

(3) 创建MNS主题订阅

(4) 授权BatchCompute推送消息

请登录控制台上进行一键授权。如果没有授权过，控制台导航条下面会出现这个提示:



如果已经授权过请忽略本条。

2. 计费相关

消息通知产生的费用统一由**消息服务结算**，批量计算不再额外收取。

3. 消息类型

3.1 集群事件

使用SDK或控制台创建集群（cluster）时，可以配置如下类型消息事件。

```
{
"Notification": {
"Topic": {
"Name": "test-topic",
"Endpoint": "http://[UserId].mns.[Region].aliyuncs.com/",
"Events": [
"OnClusterDeleted",
"OnInstanceCreated",
"OnInstanceActive"
]
}
}
}
```

字段	说明
Name	MNS主题名称
Endpoint	MNS私网Endpoint，如何获取Endpoint。

3.2 作业事件

使用SDK或控制台创建作业（job）时，可以配置如下类型消息事件。

```
{
"Notification": {
"Topic": {
"Name": "test-topic",
"Endpoint": "http://[UserId].mns.[Region].aliyuncs.com/",
"Events": [
"OnJobWaiting",
"OnJobRunning",
"OnJobStopped",
"OnJobFinished",
"OnJobFailed",
"OnTaskWaiting",
"OnTaskRunning",
"OnTaskStopped",
"OnTaskFinished",
"OnTaskFailed",
```

```
"OnInstanceWaiting",
"OnInstanceRunning",
"OnInstanceStopped",
"OnInstanceFinished",
"OnInstanceFailed",
"OnPriorityChange"
]
}
}
}
```

字段	说明
Name	MNS主题名称
Endpoint	MNS私网Endpoint，如何获取Endpoint。

4. 消息格式

消息格式目前支持json string。

4.1 集群事件

适用于OnClusterDeleted

```
{
"Category": "Cluster",
"ClusterId": "cls-hr2rbl6qt5gki7392b8001",
"ClusterName": "test-cluster",
"CreationTime": "2016-11-01T15:25:02.837728Z",
"State": "Deleted",
"Event": "OnClusterDeleted"
}
```

适用于OnInstanceCreated/OnInstanceActive

```
{
"Category": "Cluster",
"ClusterId": "cls-hr2rbl6qt5gki7392b8001",
"Group": "group1",
"InstanceId": "i-wz9c51g2s6zsrtnqi4fa",
"InnerIpAddress": "10.45.168.26",
"Hints": "",
"State": "Starting",
"CreationTime": "2016-11-01T15:25:02.837728Z",
"Event": "OnInstanceCreated"
}
```

4.2 作业事件

适用于OnJobWaiting/OnJobRunning/OnJobStopped/OnJobFinished/OnJobFailed

```
{
"Category": "Job",
"JobId": "job-000000005852472000077E900007257",
"JobName": "test-job",
"Event": "OnJobWaiting",
"State": "Waiting",
"CreationTime": "2016-11-01T15:25:02.837728Z",
"StartTime": "2016-11-01T15:35:02.837728Z",
"EndTime": "2016-11-01T15:45:02.837728Z",
"Message": ""
}
```

适用于OnTaskWaiting/OnTaskRunning/OnTaskStopped/OnTaskFinished/OnTaskFailed

```
{
"Category": "Job",
"JobId": "job-000000005852472000077E900007257",
"Task": "Echo",
"Event": "OnTaskWaiting",
"State": "Waiting",
"StartTime": "2016-11-01T15:35:02.837728Z",
"EndTime": "2016-11-01T15:45:02.837728Z"
}
```

适用于
OnInstanceWaiting/OnInstanceRunning/OnInstanceStopped/OnInstanceFinished/OnInstanceFailed

```
{
"Category": "Job",
"JobId": "job-000000005852472000077E900007257",
"Task": "Echo",
"InstanceId": "0",
"Event": "OnInstanceWaiting",
"State": "Waiting",
"StartTime": "2016-11-01T15:35:02.837728Z",
"EndTime": "2016-11-01T15:45:02.837728Z",
"RetryCount": "0",
"Progress": "0",
"StdoutRedirectPath": "oss://bucket/tests/a44c0ad8-a003-11e6-8f8e-fefec0a80e06/logs/stderr.job-
0000000058184218000008150000000D.task.0",
"StderrRedirectPath": "oss://bucket/tests/a44c0ad8-a003-11e6-8f8e-fefec0a80e06/logs/stdout.job-
0000000058184218000008150000000D.task.0",
"ExitCode": "0",
"ErrorCode": "",
"ErrorMessage": "",
"Detail": ""
}
```

适用于OnPriorityChange

```
{
"Category": "Job",
"JobId": "job-00000000585247200000077E900007257",
"JobName": "test-job",
"Event": "OnPriorityChange",
"State": "Waiting",
"CreationTime": "2016-11-01T15:45:02.837728Z",
"StartTime": "2016-11-01T15:55:02.837728Z",
"EndTime": "2016-11-01T15:57:02.837728Z",
"Message": "",
"From": "10",
"To": "20"
}
```

竞价实例是专为降低用户ECS计算成本而设计的一种按需实例。竞价实例的价格由阿里云ECS设置，并会根据市场上针对该实例的供需变化而浮动。因此，用户可充分利用竞价实例的价格浮动特性，在合适的时间购买该竞价实例资源，从而降低计算成本，并在整体成本下降的前提下，提升业务在该时间周期内的吞吐量。了解竞价实例更多细节

在批量计算中使用竞价型实例与ECS计费一致，不需要支付额外费用。目前批量计算支持的竞价实例类型请参考这里。

在批量计算中可以通过以下方式使用竞价型实例：

1. 提交作业

相关配置说明:

参数	描述
ResourceType	资源类型，可选范围:[Spot,OnDemand], 这里设置为Spot。
InstanceType	Spot 实例类型，可以通过 getQuota()方法获取可用的Spot实例类型列表。
SpotStrategy	取值范围: SpotWithPriceLimit：设置上限价格的竞价实例; SpotAsPriceGo：系统自动出价，最高不超过按量付费价格。
SpotPriceLimit	实例的每小时最高价格。支持最大 3 位小数，SpotStrategy 为 SpotWithPriceLimit 生效。

(1). 使用 Java SDK 提交作业

```
TaskDescription taskDesc = new TaskDescription();
```

```
AutoCluster autoCluster = new AutoCluster();
autoCluster.setResourceType("Spot");
autoCluster.setInstanceType("ecs.sn1.large");

//可选配置
autoCluster.setSpotStrategy("SpotWithPriceLimit");
autoCluster.setSpotPriceLimit(0.6f);
```

(2). 使用 Python SDK 提交作业

```
job_desc['DAG']['Tasks']['my-task']['AutoCluster']['ResourceType'] = 'Spot'
job_desc['DAG']['Tasks']['my-task']['AutoCluster']['InstanceType'] = 'ecs.sn1.large'

# 以下配置可选
job_desc['DAG']['Tasks']['my-task']['AutoCluster']['SpotStrategy'] = 'SpotWithPriceLimit'
job_desc['DAG']['Tasks']['my-task']['AutoCluster']['SpotPriceLimit'] = 0.6 # 0.6元
```

(3). 使用命令行提交作业

举例0：提交spot作业

```
bcs sub "python demo.py" --resource_type Spot -t ecs.sn1.large
```

举例1： 限制每小时最高价格0.6元

```
bcs sub "python demo.py" --resource_type Spot -t ecs.sn1.large --spot_price_limit 0.6
```

举例2： spot_price_limit设置为0，表示系统自动出价，最高不超过按量付费价格

```
bcs sub "python demo.py" --resource_type Spot -t ecs.sn1.large --spot_price_limit 0
```

2. 创建集群和修改集群

(1). 使用 Java sdk

```
GroupDescription group = new GroupDescription();
group.setDesiredVMCount(3);
group.setInstanceType("ecs");
group.setResourceType("OnDemand");

//可选配置
group.setSpotPriceLimit(0.6f);
group.setSpotStrategy("SpotWithPriceLimit");
```

(2). 使用 python sdk

```
group_desc['ResourceType'] = 'Spot'
group_desc['InstanceType'] = 'ecs.sn1.large'

# 以下配置可选
group_desc['SpotStrategy'] = 'SpotWithPriceLimit'
group_desc['SpotPriceLimit'] = 0.6 # 0.6元
```

(3). 使用命令行

创建集群时设置竞价策略：

```
bcs cc cluster_1 -t ecs.sn1.large --resource_type Spot --spot_price_limit 0.6
```

修改集群竞价策略：

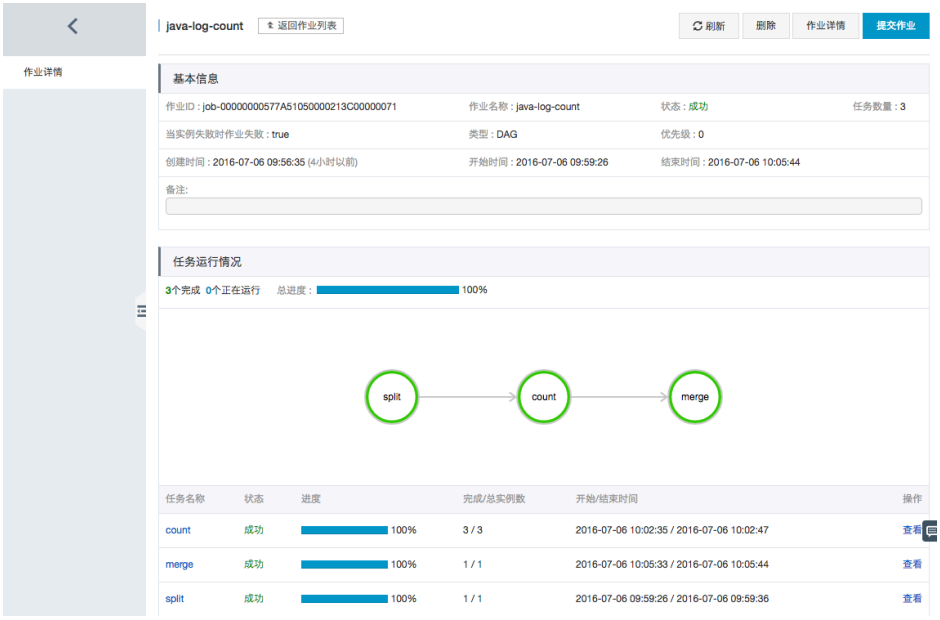
```
bcs uc cls-xxxx --spot_price_limit 0.8
```

控制台操作

1. 查询作业状态

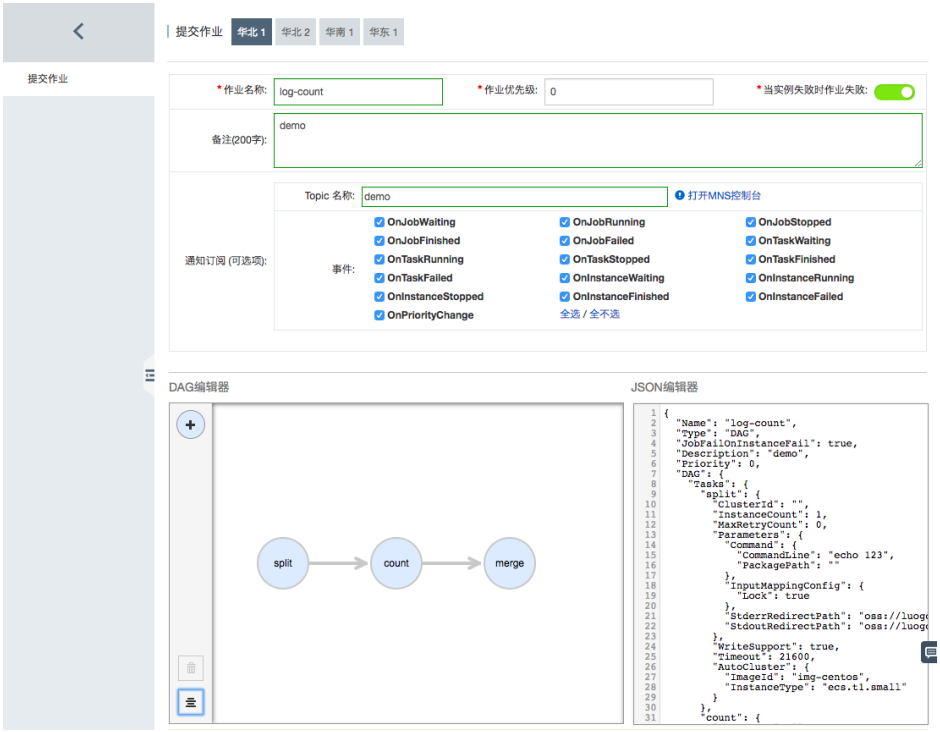
提交作业后，页面将会自动跳转至控制台作业列表，列表第一列将会显示最新提交成功的作业。

通过点击作业名称或者右侧“查看”选项，你可以看到作业的基本信息和任务运行情况。



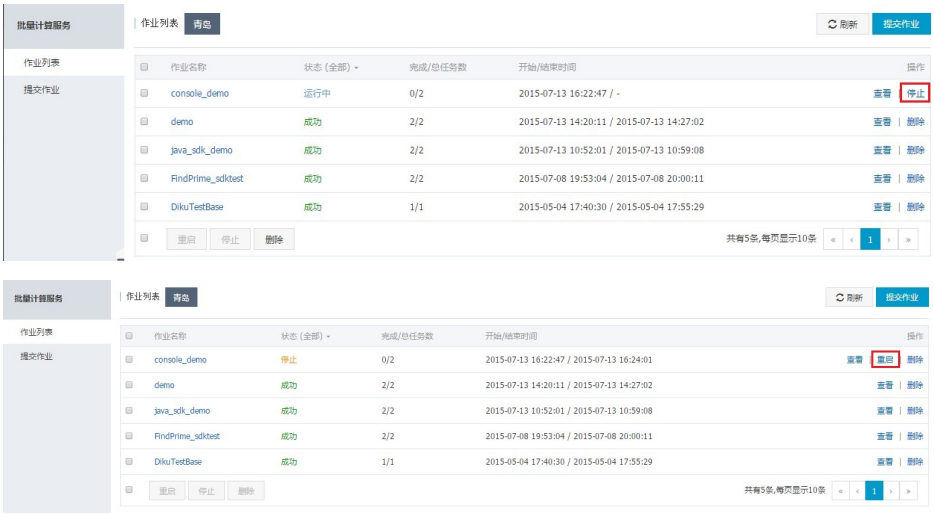
点击任务名称或者右侧“查看”选项，还可以看到任务的基本信息和实例运行情况。其中，左下部为实例编号，将鼠标悬置于各个编号上方，可以看到各个实例详情，并且可以下载实例运行日志。

2. 提交作业



3. 管理作业

作业提交成功后，作业为“等待中”或“运行”状态时，如果有需要可以点击右边的“停止”，之后也可以进行“重启”。

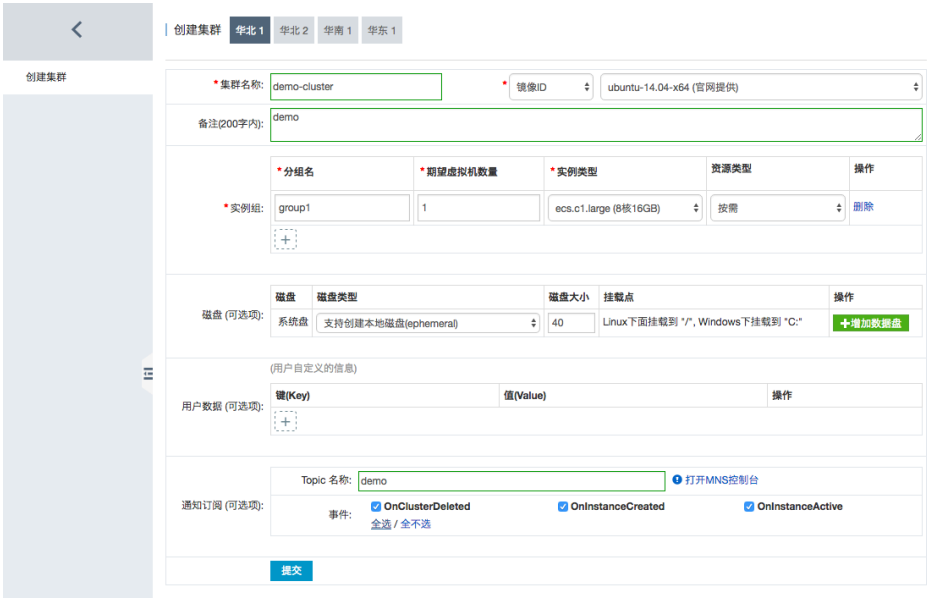


当作业状态为“停止”、“成功”或“失败”时，可以进行删除操作。

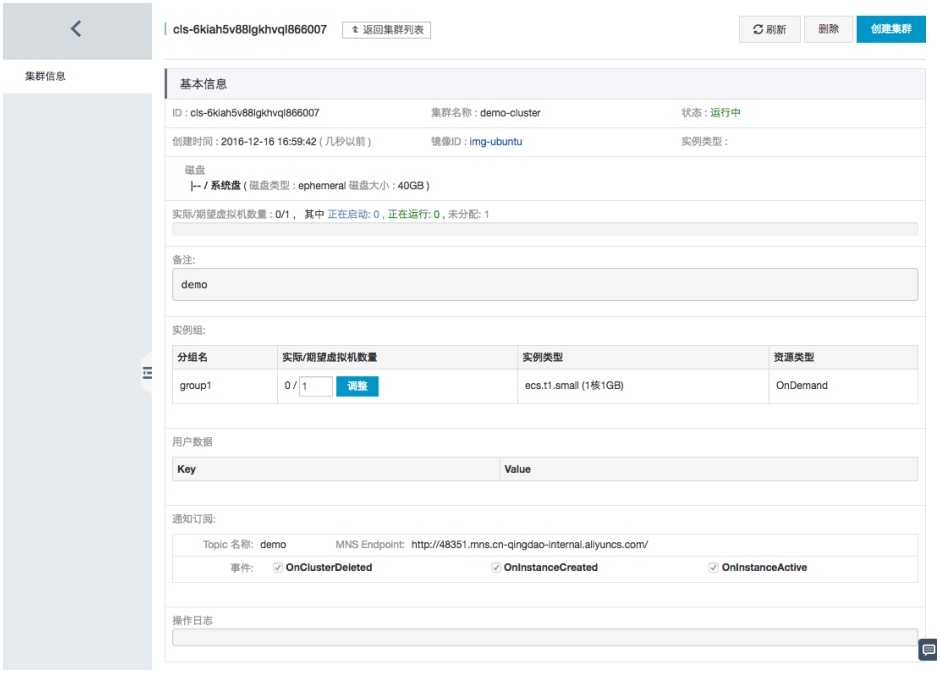
同时，还支持批量操作“重启”、“删除”或者“运行”。

1. 集群列表管理

2. 创建集群



3. 查看集群详细情况



1. 镜像列表

2. 镜像详情

3. 创建镜像

第一次创建镜像，需要授权。

授权完成后，会自动跳转回创建镜像页面，即可创建镜像。

不同用户在不同region的可用类型是不一样的。

> 专有网络VPC（Virtual Private Cloud）是您基于阿里云构建的一个隔离的网络环境，专有网络之间逻辑上彻底隔离。**了解专有网络更多细节**

BatchCompute在创建集群或作业的时候，可以指定集群创建在VPC环境内（和原有经典网络配置互斥），然后用户程序在VPC内的集群中运行访问其他云产品的程序时需要使用该云产品在VPC环境内的入口，可以参考相关云产品的文档，或者工单询问我们。集群描述中的Configs.Networks.VPC内的CidrBlock字段标识了用户想要设置的BatchCompute集群所在的网段，即，集群内的实例均在该网段内。VPC的网段仅支持私网网段，即10.0.0.0/8、172.16.0.0/12 和 192.168.0.0/16 以及其子网，因为下层对VSwitch的划分限制，要求网段掩码在12-24，所以，用户在设置CidrBlock字段需要选择10.0.0.0/12 - 10.0.0.0/24、172.16.0.0/12 -

172.16.0.0/24、192.168.0.0/16 - 192.168.0.0/24中包含的网段，否则会造成集群无法正常创建。

BatchCompute集群VPC环境下不支持bcs实例规格（格式如 bcs.xx.xxx）。

1. 使用 Java SDK

创建集群时指定:

```
ClusterDescription clusterDescription = new ClusterDescription();
Configs cfgs = new Configs();
Networks nw = new Networks();
VPC vpc = new VPC();
vpc.setCidrBlock("192.168.0.1/16"); //设置网段
nw.setVpc(vpc);
cfgs.setNetworks(nw);
clusterDescription.setConfigs(cfgs);
...
```

创建作业时指定：

```
TaskDescription taskDescription = new TaskDescription();
Configs cfgs = new Configs();
Networks nw = new Networks();
VPC vpc = new VPC();
vpc.setCidrBlock("192.168.0.1/16"); //设置网段
nw.setVpc(vpc);
cfgs.setNetworks(nw);
AutoCluster autoCluster = new AutoCluster();
taskDescription.setAutoCluster(autoCluster);
...
```

2. 使用 Python SDK

创建集群时指定:

```
from batchcompute.resources import ClusterDescription

cluster_desc = ClusterDescription()
cluster_desc.Configs.Networks.VPC.CidrBlock = "192.168.0.1/16"
...
```

创建作业时指定：

```
from batchcompute.resources import TaskDescription

task_desc = TaskDescription()
task_desc.AutoCluster.Configs.Networks.VPC.CidrBlock = "192.168.0.1/16"
...
```

3. 使用命令行工具

创建作业时指定:

```
bcs sub "python test.py" --vpc_cidr_block 192.168.0.0/16
```

创建集群时指定:

```
bcs cc myCluster --vpc_cidr_block 192.168.0.0/16
```