批量计算

用户指南

为了无法计算的价值 | 阿里云

用户指南

支持的地域

目前，批量计算服务支持的地域已覆盖亚洲、欧洲、北美洲。全部可用的地域(Region)如下：

国内部分

RegionId	地域
cn-qingdao	华北1（青岛）
cn-beijing	华北2（北京）
cn-zhangjiakou	华北3（张家口）
cn-huhehaote	华北5（呼和浩特）
cn-hangzhou	华东1（杭州）
cn-shanghai	华东2（上海）
cn-shenzhen	华南1（深圳）

国外部分

RegionId	地域
ap-southeast-1	新加坡
eu-central-1	德国（法兰克福）
us-west-1	美国（硅谷）
us-east-1	美国（弗吉尼亚）

如何提交作业

1. 批量计算提交作业的 4 种典型方法

A) 使用命令行

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

该命令会将 test.py 文件打包成 worker.tar.gz 并上传到指定位置，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。

bcs sub 命令格式为 bcs sub <commandLine> [job_name] [options]，更多参数详情参考bcs sub -h命令。

B) 使用控制台提交作业

i. 将 test.py 打包上传到 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
```

使用 OSS 控制台，上传 worker.tar.gz。

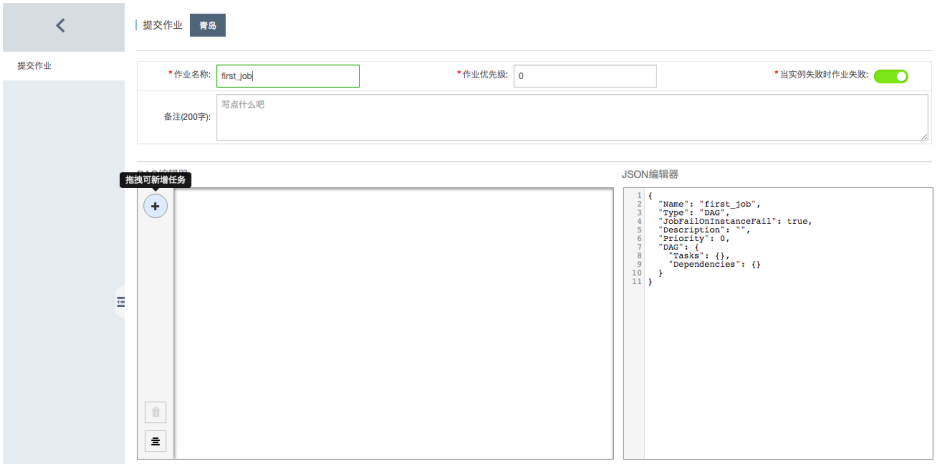
> 如果还没有开通 OSS，请先开通。
>
> 还需要创建 Bucket，假设创建了 Bucket 名称为 mybucket
>
> 然后在这个 Bucket 下创建一个目录: test

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

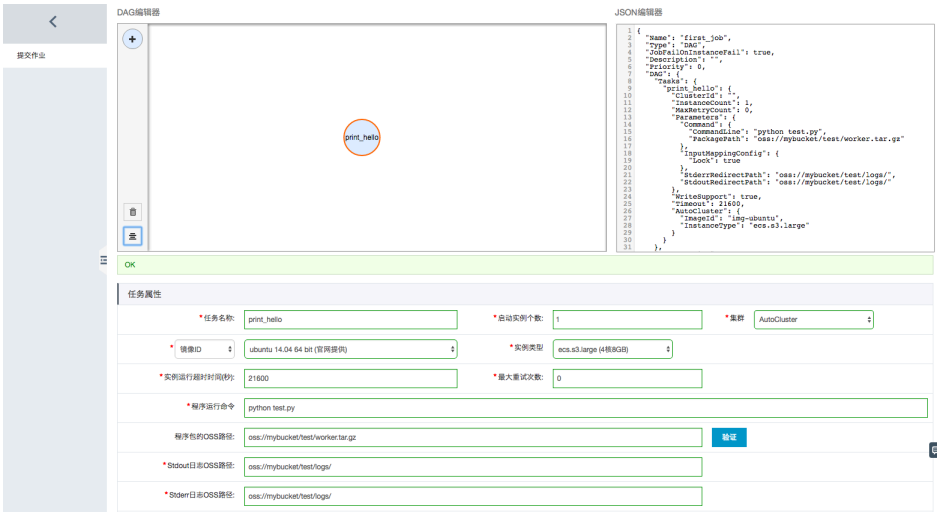
ii. 使用控制台提交作业

打开 提交作业 页面。

按照表单提示，填写作业名称为 first_job



拖拽一个任务，按照下图填写表单, 指定 ECS 镜像 ID 。



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。

C) 使用 Python SDK 提交作业

i. 将 test.py 打包上传到云端 OSS

同上一节。

ii. 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": "ecs.sn1.medium",
"ImageId": "img-ubuntu"
}
}
},
"Dependencies": {}
}
}

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

....
```

iii. 更多关于Python SDK内容

请参考 Python SDK 。

D) 使用 Java SDK 提交作业

i. 将 test.py 打包上传到云端 OSS

同上一节。

ii. 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("ecs.sn1.medium");
// 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.addOutputttMapping("/home/admin/disk2/", "oss://my-bucket/disk2/");
// task.addLogMapping( "/home/admin/a.log","oss://my-bucket/a.log");

return task;
}
}
```

iii. 更多关于 Java SDK 内容

请参考 Java SDK。

2 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。

OSS挂载

对象存储 OSS 提供了与传统文件系统不同的 API，为了支持传统程序的快速迁移，BatchCompute 允许用户将 OSS 目录直接挂载到虚拟机的本地文件系统，应用程序无需针对 OSS 进行特殊编程即可访问 OSS 上的数据。

OSS 挂载分为只读挂载和可写挂载两种：

- 只读挂载用于访问程序的输入数据，通过只读挂载点的数据访问将会被自动转换为 OSS 的访问请求，数据不需要先下载到虚拟机本地。

- 可写挂载目录下的结果数据将被先存放在虚拟机的本地，在作业运行结束时会被自动上传到 OSS 的相应位置。使用可写挂载时请确保虚拟机为结果数据分配了足够的磁盘空间。

1. 挂载数据目录

提交作业的时候，可以配置挂载，请参考如下示例。

A) 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
```

B) 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/"
}
```

C) 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。

InputMapping 和 OutputMapping 不能指定为同样的映射。

2. 挂载程序目录

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。

A) 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);
```

B) 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'
```

C) 使用命令行

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

3. 挂载文件

需求： 我在OSS上有多个不同前缀下的文件，需要挂载到批量计算VM中的同一个目录下，该如何使用；

假设 bucket1 和 bucket2 下有两个文件挂载到 VM 本地的/home/data/下：

- oss://bucket1/data/file1挂载到/home/data/file1
- oss://bucket2/data/file2挂载到/home/data/file2

另外，作业结束后有两个 VM 本地的文件分别上传到 bucket1 和 bucket2：

- /home/output/output1挂载到oss://bucket1/output/file1
- /home/output/output2挂载到oss://bucket2/output/file2

可以参考如下示例：

A) 使用 Java SDK

```
TaskDescription taskDesc = new TaskDescription();
taskDesc.addInputMapping("oss://bucket1/data/file1", "/home/data/file1");
taskDesc.addInputMapping("oss://bucket2/data/file2", "/home/data/file2");
taskDesc.addOutputMapping("/home/output/output1", "oss://bucket1/output/file1");
taskDesc.addOutputMapping("/home/output/output2", "oss://bucket2/output/file2");
```

B) 使用 Python SDK

```
# 只读挂载
job_desc['DAG']['Tasks']['my-task']['InputMapping'] = {
"oss://bucket1/data/file1": "/home/data/file1",
"oss://bucket2/data/file2": "/home/data/file2"
}

# 可写挂载
job_desc['DAG']['Tasks']['my-task']['OutputMapping'] = {
"/home/output/output1": "oss://bucket1/output/file1",
"/home/output/output2": "oss://bucket2/output/file2"
}
```

4. InputMapping 挂载限制说明

A) OSS上存储对象的命名

- 单个文件名最长为 255 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。

B) 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 \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，如 \127.0.0.1\ossdata\bucket\dir\local\file，Linux 操作系统中不存在这个问题。

C) 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。

D) 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。

E) 文件挂载

将 OSS 上的单独文件挂载到 VM 本地时，需要保证所有文件到的本地目录不能时另外文件挂载或者目录挂载的父目录。

F) 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。

NAS挂载

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 相关信息使用阿里云 NAS 服务。

1. 相关概念

从安全、性能以及便捷性来考虑，目前批量计算仅支持在专有网络中使用阿里云 NAS，以下是专有网络和阿里云 NAS 的一些基本概念：

- 专有网络 (VPC) : 专有网络 VPC（Virtual Private Cloud）是用户基于阿里云创建的自定义私有网络，不同的专有网络之间二层逻辑隔离，用户可以在自己创建的专有网络内创建和管理云产品实例，比如 ECS、负载均衡、RDS、NAS 等。
- 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。
- 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，由于批量计算只支持在专有网络中使用 NAS，创建挂载点时需要指定挂载点类型为专有网络。

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、VPC 其他相关内容，请参考：NAS 官方文档、专有网络官方文档。

2. 使用 NAS

在批量计算中使用 NAS，需要按照如下步骤进行操作。

A) 数据准备

以下专有网络名、文件系统名、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。

- 创建专有网络：参考 创建专有网络 新建专有网络，其 ID 由专有网络自动生成，假设为vpc-m5egk1jsm3qkbxxxxxxxx。
- 创建交换机：参考创建交换机 创建一个新的交换机，其 ID 由专有网络自动生成，假设为vsw-2zeue3c2rciybxxxxxxxx，注意最好将交换机的网段设置为专有网络的子网。
- 创建 NAS 文件系统：按照之前的文档在 NAS 控制台创建文件系统，其 ID 由 NAS 自动生成，假设为0266e49fea，目前批量计算仅支持 NFS 类型的 NAS 文件系统。
- 创建挂载点：选择挂载点类型为专有网络，并选择上面步骤创建的专有网络和交换机，如果没有特殊需求，选择默认的权限组即可，创建完成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。

B) 使用用户专有网络创建集群

用户程序要在批量计算集群节点中访问存储在 NAS 中的文件，需要在创建批量计算集群时同时指定VpcId和CidrBlock，将批量计算集群创建到用户专有网络中。

- VpcId： 用户NAS挂载点所在的专有网络ID，用户需要将示例中的ID替换为用户的实际值；
- CidrBlock：批量计算集群在专有网络的私网范围，一般为专有网络网段的子网段，用户按照自己的网段规划填入合适的值，详情参考网段规划。

i. 使用 Python SDK 创建

```
# 以下是创建集群时指定专有网络和集群的网段规划

cluster_desc['Configs']['Networks']['VPC']['VpcId'] = 'vpc-m5egk1jsm3qkbxxxxxxxx'
cluster_desc['Configs']['NetWorks']['VPC']['CidrBlock'] = '192.168.0.0/20'

# 用户也可以在提交AutoJob时指定专有网络和集群的网段规划
job_desc['DAG']['Tasks']['my-task']['AutoCluster']['Configs']['Networks']['VPC']['VpcId'] = 'vpc-m5egk1jsm3qkbxxxxxxxx'
job_desc['DAG']['Tasks']['my-task']['AutoCluster']['Configs']['Networks']['VPC']['CidrBlock'] = '192.168.0.0/20'
```

ii. 使用 JAVA SDK 创建

```
Configs configs = new Configs();
Networks networks = new Networks();
VPC vpc = new VPC();
vpc.setCidrBlock("192.168.0.0/20");
vpc.setVpcId("vpc-m5egk1jsm3qkbxxxxxxxx");
networks.setVpc(vpc);
configs.setNetworks(networks);


//创建集群时指定 NAS 文件系统和权限组信息
```

```
ClusterDescription clusterDescription = new ClusterDescription();
clusterDescription.setConfigs(configs);

//用户也可以在提交 AutoJob 时指定 NAS 文件系统和权限组信息
AutoCluster autoCluster = new AutoCluster();
autoCluster.setConfigs(configs);
```

C) 指定挂载点

批量计算根据用户提供的挂载信息自动将 NAS 的挂载点挂载为本地目录，需要用户在集群描述或作业描述的Mounts.Entries中指定 NAS 挂载点到本地目录的映射。

- Source：以nas://为前缀，后面接上 NAS 挂载点和 NAS 文件系统目录信息，注意：
  - 批量计算的挂载支持多种来源，为了区分来源，目前 NAS NFS 文件系统需要以nas://作为前缀。
  - Windows 挂载由于 Windows NFS client的行为，需要在最后加上感叹号。
  - 用户也可以在用户程序中自行挂载（上节中创建集群到用户专有网络的步骤必不可少），具体请参考 手动挂载 。
- Destination：批量计算集群节点内的本地目录，用户不需要事先创建该目录，批量计算会自动为用户创建该目录。
- WriteSupport： 是否支持可写，如果用户指定为False，会使用批量计算独有的分布式缓存来提高访问 NAS 的性能。

i. 使用 Python SDK 挂载

```
# 集群级别挂载
# For Linux
cluster_desc['Configs']['Mounts']['Entries'] = {
'Source': 'nas://0266e49fea-yio75.cn-beijing.nas.aliyuncs.com:/',
'Destination': '/home/admin/mydir/',
'WriteSupport': true,
}

# For Windows
cluster_desc['Configs']['Mounts']['Entries'] = {
'Source': 'nas://0266e49fea-yio75.cn-beijing.nas.aliyuncs.com:/!',
'Destination': 'E:',
'WriteSupport': true,
}

# 作业级别挂载
# 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': 'E:',
'WriteSupport': true,
}
```

ii. 使用 JAVA SDK 挂载

```
MountEntry mountEntry = new MountEntry();
mountEntry.setSource("nas://0266e49fea-yio75.cn-beijing.nas.aliyuncs.com:/");
mountEntry.setDestination("/home/admin/mydir/");
mountEntry.setWriteSupport(true);
mounts.addEntries(mountEntry);
```

3. NAS挂载注意事项

A) 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 。

B) 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。

挂载磁盘

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）。

A) 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": "cloud_efficiency",
"Size": 50
},
"DataDisk": {
"Type": "cloud_efficiency",
"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_ssd，DataDisk 的 Typ e也必须是 cloud_ssd。
- 如果 Type 为空，系统会自动选择默认的磁盘类型，使用更为简单。
- 数据盘必须指定 MountPoint，Linux 下可以挂载到目录，Windows 下只能挂载到驱动,如 E 盘：” E:”

B) 命令行提交作业时指定磁盘

```
bcs sub "echo 123" --disk system:cloud_efficiency:40,data:cloud_efficiency:500:/home/disk1
```

系统盘配置格式: system:[cloud_efficiency|cloud_ssd|cloud|ephemeral|default]:[40-500], 举例: system:cloud_efficiency:40, 表示系统盘挂载 40GB 的高效云盘.

数据盘配置格式: data:[cloud_efficiency|cloud_ssd|cloud|ephemeral|default]:[5-2000]:[mount-point], 举例: data:cloud_efficiency:500:/home/disk1, 表示挂载一个 500GB 的高效云盘作为数据盘, window 下只能挂载到驱动,如E盘:” data:cloud_efficiency:500:E” .

如果选择 default，系统会自动选择默认的磁盘类型，对磁盘性能没有特殊要求的应用，建议使用默

认类型。

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

另外，可以只指定系统盘：

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

当然，也可以只指定数据盘：

```
bcs sub "echo 123" --disk data:default:500:/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。如果是使用 BCS 开头的批量计算专有实例类型，磁盘类型只能选择 ephemeral。ECS 实例类型的磁盘类型选择请参考实例规格族。

使用镜像

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 默认提供以下三种系统镜像供用户直接使用（支持经典网络和 VPC 网络）：

操作系统	镜像ID	系统详情
Windows	img-windows-vpc	Windows Server 2008 R2 企业版 64位 中文
Ubuntu	img-ubuntu-vpc	Ubuntu 14.04 64位, python 2.7, jdk 1.7, Docker 17.0.1

Centos	img-centos-vpc	Centos 6.8 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

A) 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 请参考**获取镜像列表** 。
- 使用 Python SDK 请参考**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**。

自定义镜像

如果 **官方提供的镜像**无法满足您的需求，请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 5 步:

1. 购买 BatchCompute 基础镜像并启动 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：

	ubuntu 14.04	centos 6.8	windows server2008 R2
华东1	link	link	link
华东2	link	link	link
华北1	link	link	link
华北2	link	link	link
华南1	link	link	link

单击**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**专有网络。**
- 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。
- 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。**注意： 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，单击**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 软件，基因数据分析工具等。

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。参考 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。参考 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

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图。

> 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。

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。

您只需要登录 **ECS 控制台** 释放掉该实例即可。

实例类型

批量计算全面支持 ECS 企业级实例规格，用户可以通过按量或者竞价的形式调用。如果需要使用其他实例类型，请提工单咨询。

在深圳(华南1)， 北京(华北2)，杭州（华东1）区域提供的原有专属实例（规格名以 bcs 开头）不再对新客户售卖，原有客户可以继续使用。专属实例规格描述如下表。

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

环境变量

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:

A) 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']
```

B) 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。

A) 使用 Python SDK

代码片段:

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

B) 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);
```

C) 使用命令行工具

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

用户限额

限额配置用于批量计算服务的管理，需要说明的是，集群层面的实例为虚拟机，而作业层面的实例为一个具体的任务。默认限额如下表所示，如有特殊需求，请提交工单。

集群层面

限额种类	默认值
单用户最大集群数	50
单集群内最大可创建的实例组数	1
单实例组最大可申请的实例数	500
单用户最大并发实例数	50

作业层面

限额种类	默认值
单用户最大作业数	500
单用户单作业最大任务数	30

单用户单任务最大实例数	500
单用户单实例最大超时时间	24小时

使用集群

集群使用场景介绍

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	提交作业时指定集群 ID

	InstanceType，还有需要的机器台数	

2. 如何使用 Cluster

A) 使用 Python SDK

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

B) 使用 Java SDK

以下是代码片段:

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

C) 使用命令行

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

3. 如何使用 AutoCluster

A) 使用 Python SDK

```
...
autoCluster = {
'ImageId': 'img-ubuntu',
'InstanceType': 'ecs.sn1.medium'
}
...
job_desc['DAG']['Tasks']['my-task']['AutoCluster'] = autoCluster
...
```

B) 使用 Java SDK

以下是代码片段:

```
AutoCluster autoCluster = new AutoCluster();
autoCluster.setImageId("img-ubuntu");
```

```
autoCluster.setInstanceType("ecs.sn1.medium");

TaskDescription desc = new TaskDescription();

desc.setAutoCluster(autoCluster);
```

C) 使用命令行

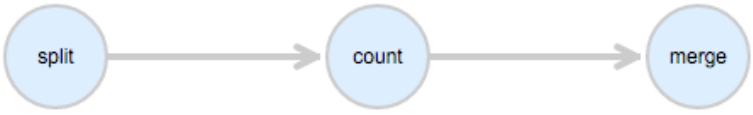
```
# 使用 Auto Cluster
bcs sub "python main.py" -c img=img-ubuntu:type=ecs.sn1.medium
```

多任务

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:



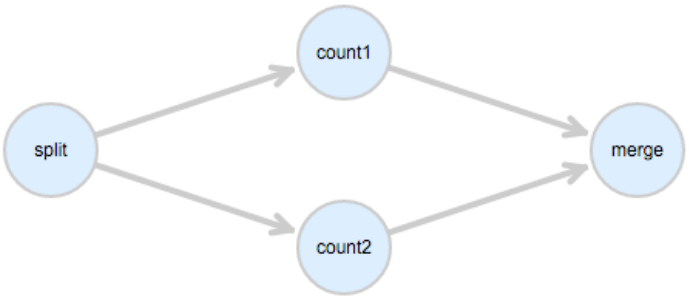
在 job description json 中这样描述:

```
{
"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。

并发

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": "ecs.sn1.medium"
},
"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)
...
```

完整示例请参考 **快速开始** 。

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

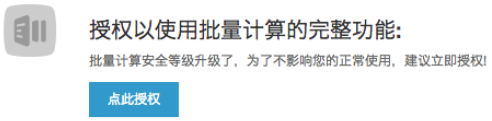
A) 开通消息服务

B) 创建 MNS 主题

C) 创建 MNS 主题订阅

D) 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，控制台会出现这个提示:



如果已经授权，过请忽略。

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

A) 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，参考 如何获取 Endpoint

B) 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，参考 如何获取Endpoint。

4. 消息格式

消息格式目前支持 json string。

A) 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"
}
```

B) 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-
000000005818421800000815000000D.task.0",
"StderrRedirectPath": "oss://bucket/tests/a44c0ad8-a003-11e6-8f8e-fefec0a80e06/logs/stdout.job-
```

```
00000000581842180000081500000000D.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"
}
```

竞价型资源

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。

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。

A) 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);
```

B) 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
```

C) 使用命令行提交作业

示例 1：提交 spot 作业

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

示例 2： 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
```

示例 3： 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

A) 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");
```

B) 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
```

C) 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
```

VPC支持

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。

也增加了支持在用户 VPC 内创建集群的功能，这个功能允许用户指定自己账号下的 VPC，会在用户账号下的 VPC 内去创建集群，这样的好处是，方便集群与用户账号下已有的云服务资源进行连通，例如 NAS，RDS 等。在使用这个功能的时候需要配置除了上述 Configs.Networks.VPC.CidrBlock 字段外，还需要配置 Configs.Networks.VPC.VpcId 字段用来标识用户账号下的 VPC 的 VpcId。这里需要注意的是，在集群创建过程中，我们会在用户指定的 VPC 内创建 VSwitch 等资源，用户需要保证在集群存在期间不要随意操作这些自动创建出的 VSwitch，否则会影响集群的正常运行。另外，在这种模式下，多个集群可以共享同一个 CIDR 网段（即多个集群或者作业可以设置同一个网段，它们共享这个网段内的网络资源），但是用户需要注意的是，在这个 CIDR 网段内不要创建 VSwitch 等资源。

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

以下样例请更新到最新版本的SDK或工具再使用。

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
vpc.setVpcId("vpc-xxyyzz"); //如果想要使用用户 VPC 功能，需要设置此字段
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
vpc.setVpcId("vpc-xxyyzz"); //如果想要使用用户 VPC 功能，需要设置此字段
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"
cluster_desc.Configs.Networks.VPC.VpcId = "vpc-xxyyzz" # 如果想要使用用户VPC功能，需要设置此字段
...
```

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"
task_desc.AutoCluster.Configs.Networks.VPC.VpcId = "vpc-xxyyzz" # 如果想要使用用户 VPC 功能，需要设置此字段
...
```

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

创建集群时指定:

```
bcs cc myCluster --vpc_cidr_block 192.168.0.0/16
bcs cc myCluster --vpc_cidr_block 192.168.0.0/16 --vpc_id vpc-xxyyzz
```

创建包月集群

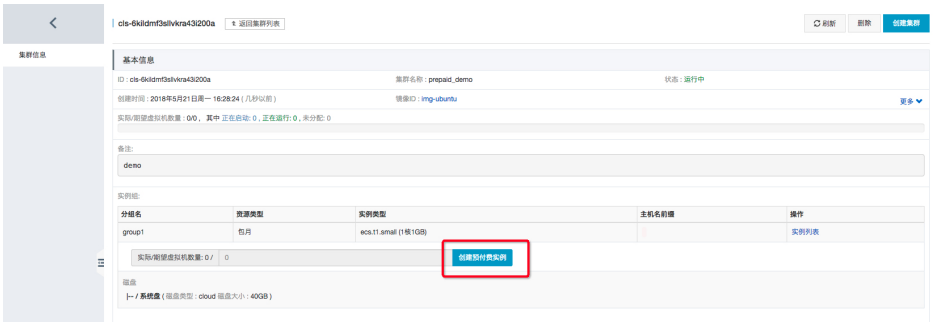
1. 创建集群

包月集群目前仅针对有限用户开放，如果需要使用请提工单申请。

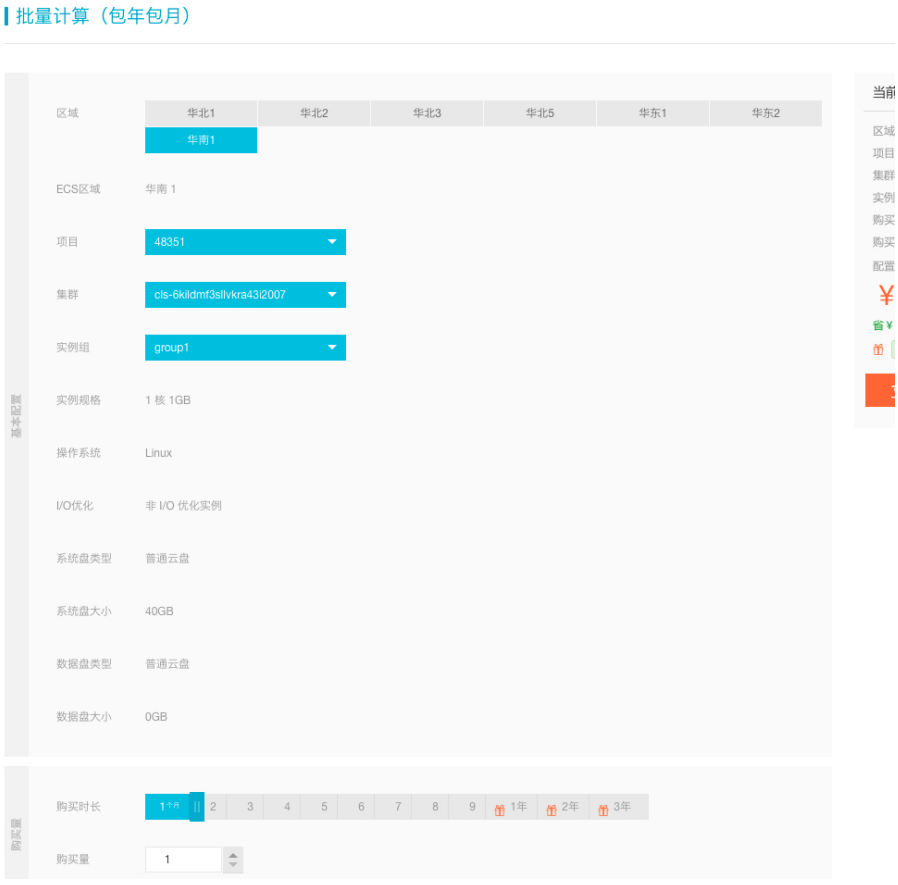
创建集群时，group 的资源类型选择“包月”。期望虚拟机数量会自动设置为0，并且不能修改。

填写好表单后，点击提交按钮创建。

2. 购买包月实例



进入刚才创建的集群详情页，点击”创建预付费实例”，到购买页面创建。



选择好配置后，点击”购买”按钮，按提示操作即可。

Docker

前言

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。

也可以自定义制作一个 Docker 镜像，上传到阿里云的容器镜像服务仓库中或者使用 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

以 AutoCluster 模式为例，对比VM 方式和Docker方式的使用过程。

> VM 方式。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 Windows）；系统运行任务时，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持久化存储，然后销毁 VM，准备执行下一个任务。
>
> Docker 方式：用户提交作业运行任务时，会先启动 VM 来支持 Docker 的系统镜像（如：支持 Docker 的 Ubuntu）；然后从容器镜像服务仓库或者 OSS 下载指定的 Docker 镜像，并在该 VM 中启动，用户的任务将在该 Docker 容器内运行；任务完成后，结果上传到指定的持久化存储，然后销毁 VM，准备执行下一个任务。

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（例如，官网提供的 Ubuntu），还需指定自定义 Docker 镜像。
程序运行平台	支持 Windows 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

A) 请到 Docker 官网下载安装

- 在 Windows/Mac 上安装 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。

- linux 上请自行到 官网下载安装 。

注意： 确保安装的 Docker 版本 >= 1.10, 否则会有兼容问题。

B) 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. 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。

本地调试

如果希望使用 Docker 镜像进行本地调试， 可以根据本节内容操作。

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
```

参数解释：

```
-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: 制作的 docker 镜像名称，如 myubuntu

your_command:命令行及参数
```

例如，本地程序路径为： /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 对应 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：

- 路径 /home/admin/log-count/ 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。

Docker镜像制作

docker 镜像制作主要有两种方式 Dockerfile 和 快速制作方式。

1. Dockerfile 制作镜像

本例中我们采用 Dockerfile的形式 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

2. 快速制作镜像

2.1 运行基础镜像容器

```
docker run -it ubuntu
```

该命令将以 root 身份进入 ubuntu：

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

安装软件，比如：

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

安装结束退出：

```
exit
```

2.2 制作镜像

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

找到对应的CONTAINER ID ， 例如： 41570524e867

```
docker commit 41570524e867 myubuntu
```

完成后，可以使用以下命令查看是否成功。

```
docker images
```

Docker镜像上传到容器镜像仓库

阿里云容器镜像服务（Container Registry）提供安全的应用镜像托管能力，精确的镜像安全扫描功能，稳定的国内外镜像构建服务，便捷的镜像授权功能，方便用户进行镜像全生命周期管理。有关容器镜像服务的详细介绍请参考其官方文档。

目前批量计算的用户也可以在 API 和 GDK 中通过配置相关镜像信息使用阿里云的容器镜像服务存放制作成功的镜像。

1. 准备工作

1.1 开通容器镜像服务

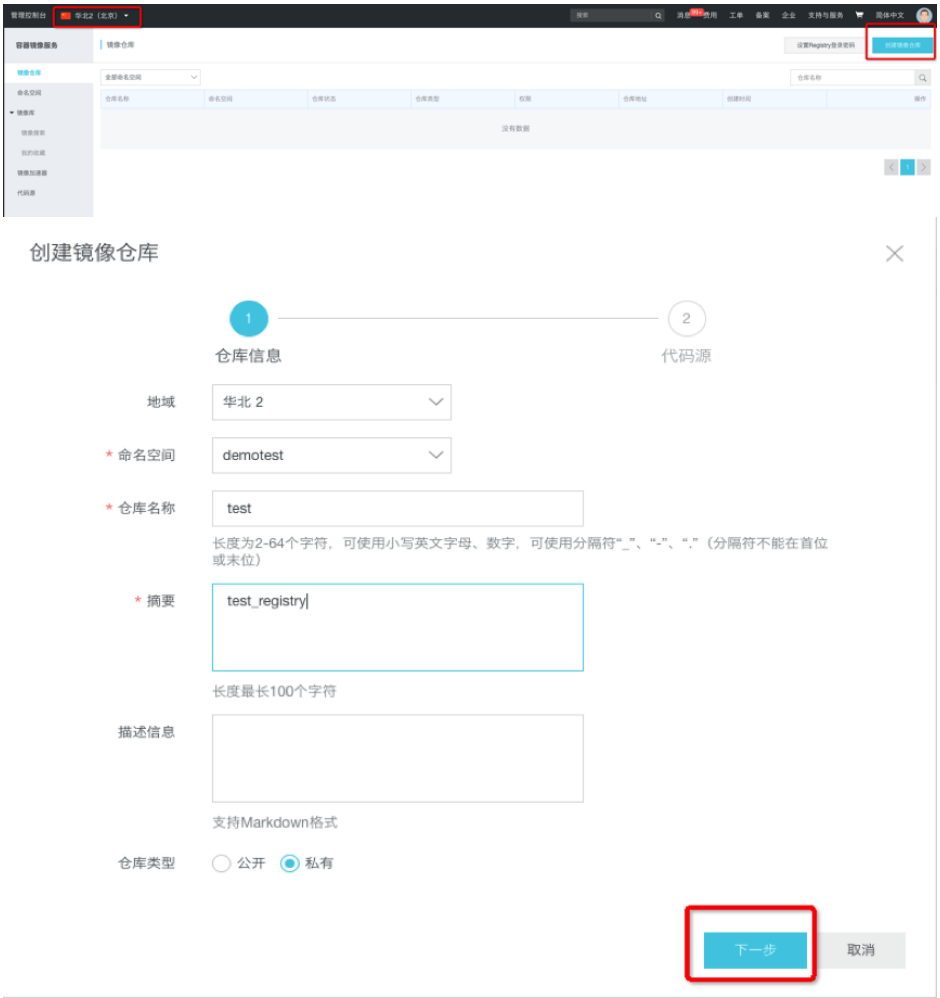
登录到容器镜像服务控制台，首次登录需要设置Registry的登录密码，开通过程请参考文档。

1.2 创建名字空间域

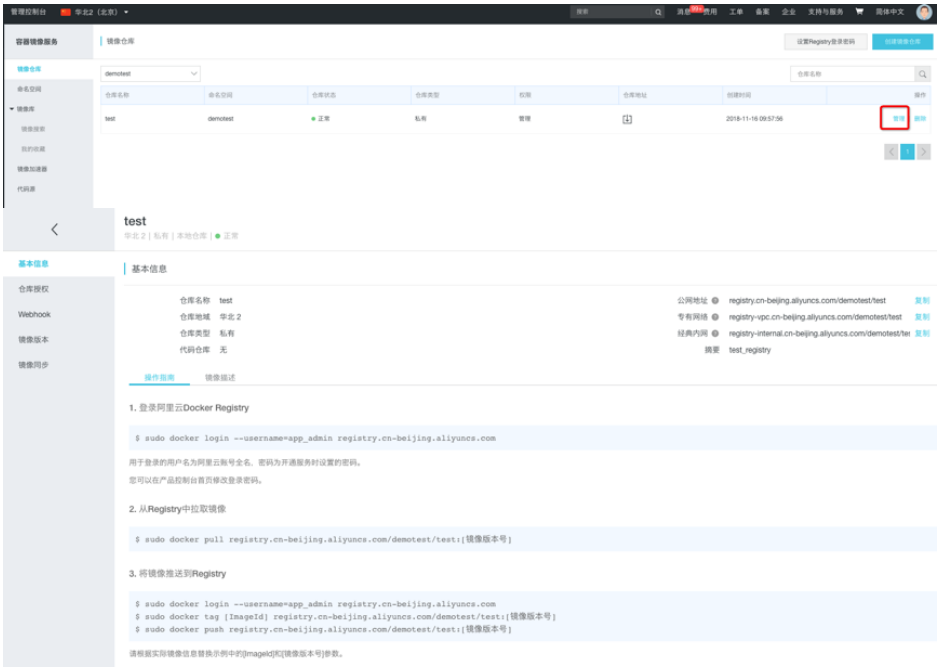
容器镜像服务开通后，首次使用需要在控制台创建名字空间域，名字空间域创建过程参考文档。

1.3 创建仓库

仓库作为一些镜像的集合，推荐将一个镜像或镜像的不同版本放置在一个仓库中。仓库的使用注意事项参考文档。仓库的创建过程如下：



镜像仓库创建完成后，点击管理可以获取详细的仓库信息，如仓库的所属的region，以及仓库的地址；包括仓库的登录、推送镜像以及拉取镜像的方式。详细信息如图：



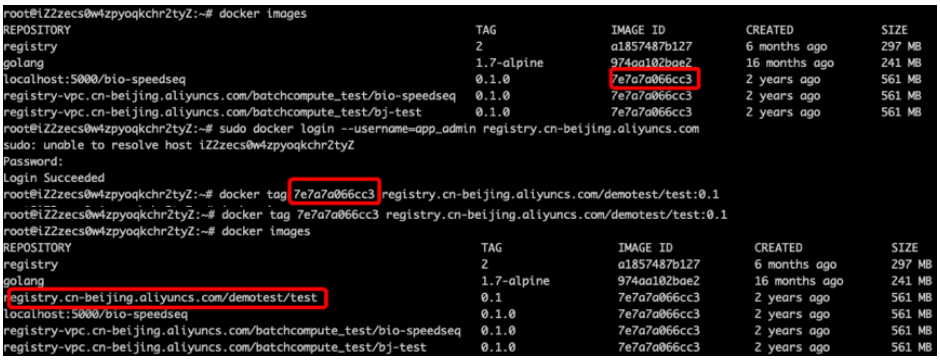
2. 推送镜像

2.1 登录 Registry

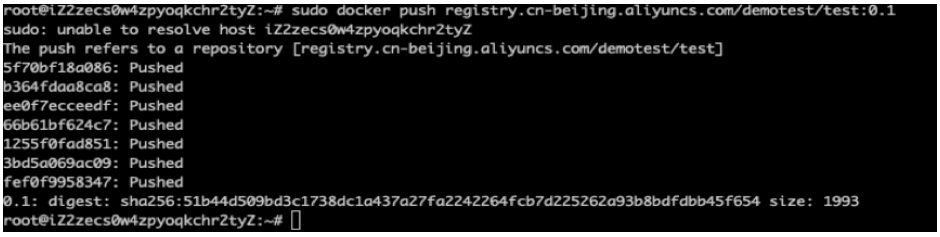
登录到存放镜像的 ECS 实例或者 服务器，在实例内或者服务器上登录到 Registry。登录方式从仓库的详细页面获取。登录密码为开始容器镜像服务时设置的 Registry 登录密码。

2.2 修改镜像名称

修改镜像名称，命名格式为 仓库地址/名字空间/仓库名称:镜像版本号。注意仓库地址和登录仓库地址保持一致。如本实例中，采用公网(registry.cn-beijing.aliyuncs.comregistry.cn-beijing.aliyuncs.com)方式登录，则修改镜像名称时仓库地址也必须是该登录地址；若登录地址为VPC登录，则镜像名称中的地址也必须采用VPC地址，否则无法正常推送镜像到容器服务。



2.3 推送镜像



在容器镜像服务控制台查看最新推送的镜像信息。

推荐使用容器镜像服务的模式推送 Docker 镜像。

3. 作业提交

支持容器镜像模式下的docker 作业提交请参考提交作业实例

Docker镜像上传到OSS

若需要将制作的 Docker 镜像上传到 OSS，需要按如下步骤操作。

安装 OSS Docker Registry 2

假设 docker 存储到 OSS 的目录路径为oss://your-bucket/dockers/，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：

i. 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-shenzhen
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	阿里云的 access key id
your_access_key_secret	阿里云的 access key secret
your-bucket	阿里云的 bucket
oss-cn-shenzhen	bucket 所在的 region

关于 OSS 配置的详细信息请参见 Docker 官方文档 。

ii. 运行命令安装

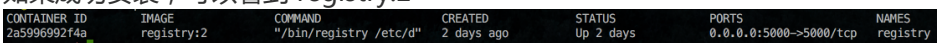
```
docker pull registry:2
docker run -v `pwd`/config.yml:/etc/docker/registry/config.yml -p 5000:5000 --name registry -d registry:2
```

- 注意：region 使用 oss-cn-shenzhen, 表示使用华南1(深圳) 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。

iii.查看结果

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

如果成功安装，可以看到 registry:2



镜像上传 OSS

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

注意：

1. 要用 localhost:5000/ 作为前缀，用其他的字符串无法上传。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。
3. 检验镜像上传是否成功, 可以使用 OSS 控制台查看是否有这个目录: oss://your-bucket/dockers/docker/registry/v2/repositories/myubuntu/，使用 Docker 时，对应参数填写如下：
    - BATCH_COMPUTE_DOCKER_REGISTRY_OSS_PATH：oss://your-bucket/dockers
    - BATCH_COMPUTE_DOCKER_IMAGE：localhost:5000/myubuntu:xxxx （ xxxx 为 myubuntu 的版本号)

Docker作业提交

BatchCompute 中, 提交作业时使用 docker 与普通 VM 作业基本相同，只是在设置作业参数时有稍微区别。

1. DAG 作业

1.1 使用支持 Docker 的 ImageId

需要将集群描述的 ImageId 指定为 BatchCompute 的公共镜像的 Id（支持 Docker 镜像, ID 为 img-ubuntu）。

AutoCluster 集群 ImageId 设置代码：

```
//设置AutoCluster集群ImageID
AutoCluster autoCluster = new AutoCluster();
//设置集群镜像信息ECSImageId 在不同region可能会发生变化
//autoCluster.setECSImageId("m-wz9dk5nao5z3fw6bo9k6");
```

```
//建议使用setImageId接口设置
autoCluster.setImageId("img-ubuntu");
```

固定集群 ImageId 设置代码：

```
ClusterDescription desc = new ClusterDescription();
desc.setName("cluster_test");
desc.setImageId("img-ubuntu");
```

1.2 Docker 镜像在 OSS

需要 DAG 作业环境变量(EnvVars)中增加如下两个参数如下，具体 EnvVars 变量在 DAG 作业中的位置，参考API文档。

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。

示例代码：

```
//oss registry模式
cmd.addEnvVars("BATCH_COMPUTE_DOCKER_IMAGE", "localhost:5000/yuorBucket/dockers:0.1");//镜像名称:版本;
cmd.addEnvVars("BATCH_COMPUTE_DOCKER_REGISTRY_OSS_PATH", "oss://your-bucket/dockers");//设置OSS地址
```

1.3 Docker 镜像在容器镜像仓库

需要 DAG 作业 Docker 变量设置容器镜像仓库地址以及镜像版本号如下，具体 Docker 变量在 DAG 作业中的位置，参考API文档。

示例代码

```
//容器镜像模式
Command.Docker docker = new Command.Docker();
docker.setImage("registry.cn-beijing.aliyuncs.com/demotest/test:0.1");
cmd.setDocker(docker);
```

1.4 创建作业的源码

JAVA 源码参考作业创建SDK描述。

2. APP 作业

APP在创建的时候指定是VM 运行还是 docker运行作业，因此需要在创建APP时配置，APP描述中设置docker配置。Docker 镜像在 OSS 和镜像在容器镜像仓库 参数相同，只是设置方法存在差异。

2.1 Docker 镜像在 OSS

示例代码

```
//设置docker oss registry 模式
AppDescription.Docker docker = new AppDescription.Docker();
docker.setImage("localhost:5000/yuorBucket/dockers:0.1");//镜像信息
docker.setRegistryOSSPath("oss://your-bucket/dockers");//OSS 地址
desc.setDocker(docker);
```

2.2 Docker 镜像在 容器镜像仓库

示例代码

```
//设置docker oss registry 模式
AppDescription.Docker docker = new AppDescription.Docker();

//设置docker 容器镜像服务 模式
docker.setImage("registry.cn-beijing.aliyuncs.com/demotest/test:0.1");//镜像信息

desc.setDocker(docker);
```

2.3 创建APP的源码

JAVA 源码参考APP创建SDK描述。

docker作业示例

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:

A) 上传数据文件到 OSS

下载本示例所需的数据: log-count-data.txt

将 log-count-data.txt 上传到:

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

- your-bucket 表示对应的 bucket，本示例假设 region 为: cn-shenzhen。
- 上传数据到OSS，请参考 OSS 上传文档 。

B) 准备任务程序

本示例的作业程序使用 python 编写， 下载本示例所需程序: log-count.tar.gz

解压到如下目录：

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

解压后的目录结构如下：

```
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

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。

A) 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，请参考 多任务支持 。

B) 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，具体请参考 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：

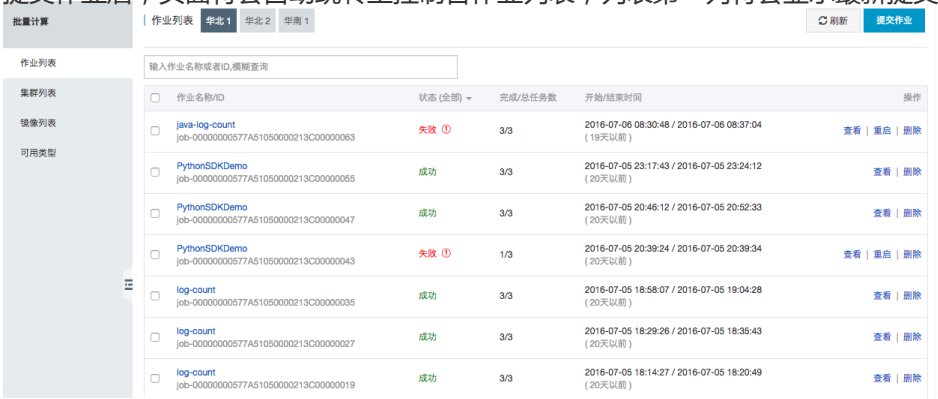
```
{"INFO": 2460, "WARN": 2448, "DEBUG": 2509, "ERROR": 2583}
```

控制台操作

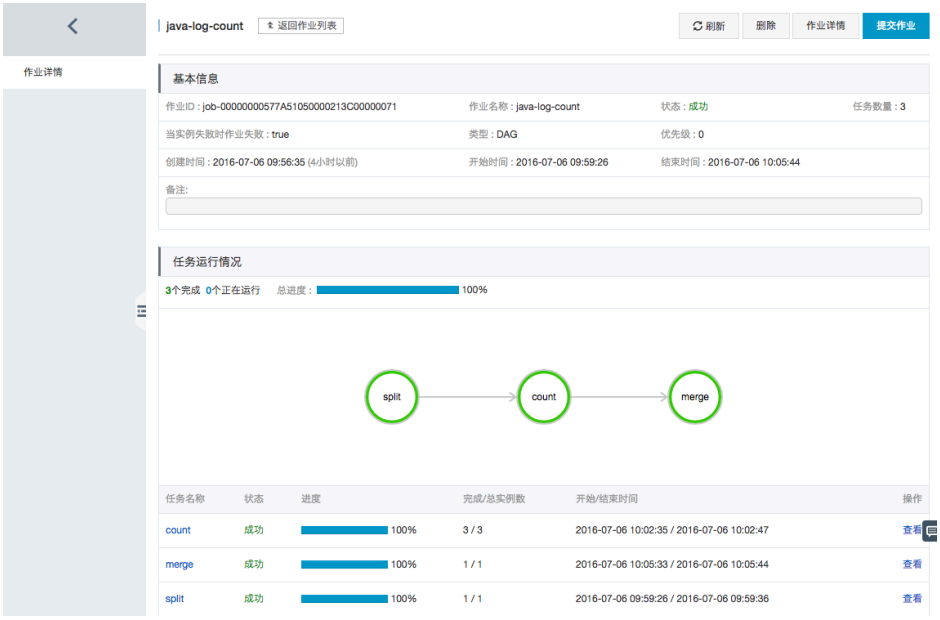
作业管理

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。

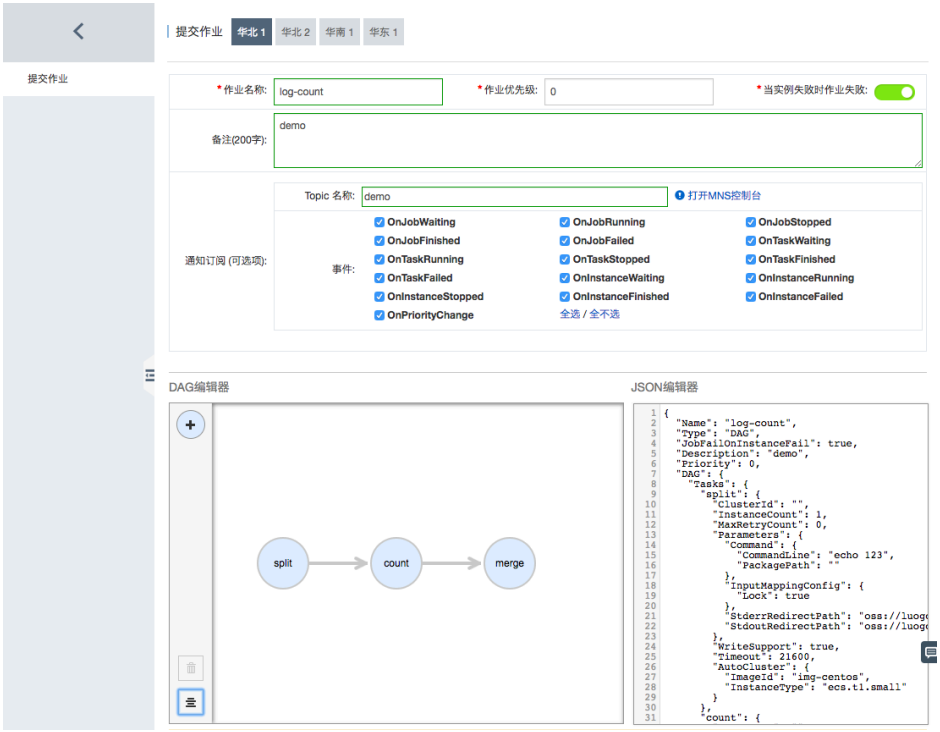


通过点击作业名称或者右侧“查看”选项，你可以看到作业的基本信息和任务运行情况。



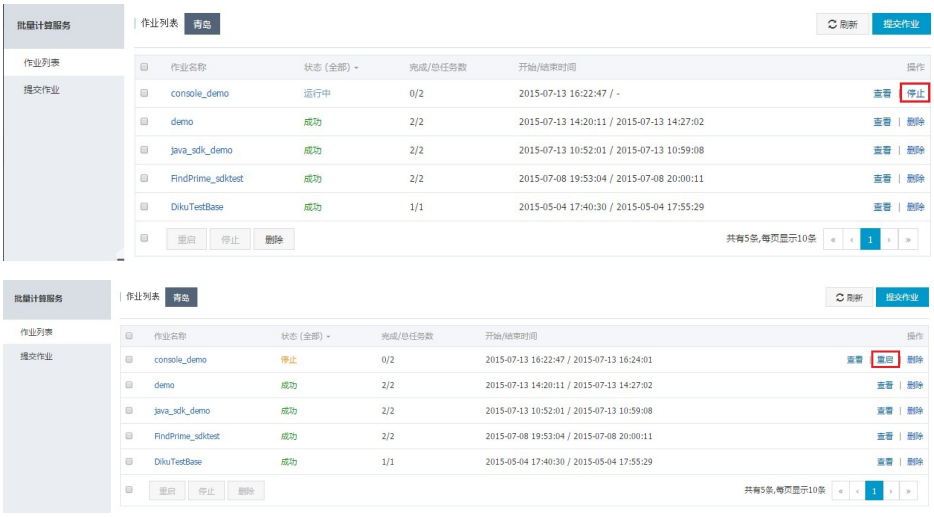
点击任务名称或者右侧“查看”选项，还可以看到任务的基本信息和实例运行情况。其中，左下部为实例编号，将鼠标悬置于各个编号上方，可以看到各个实例详情，并且可以下载实例运行日志。

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”。



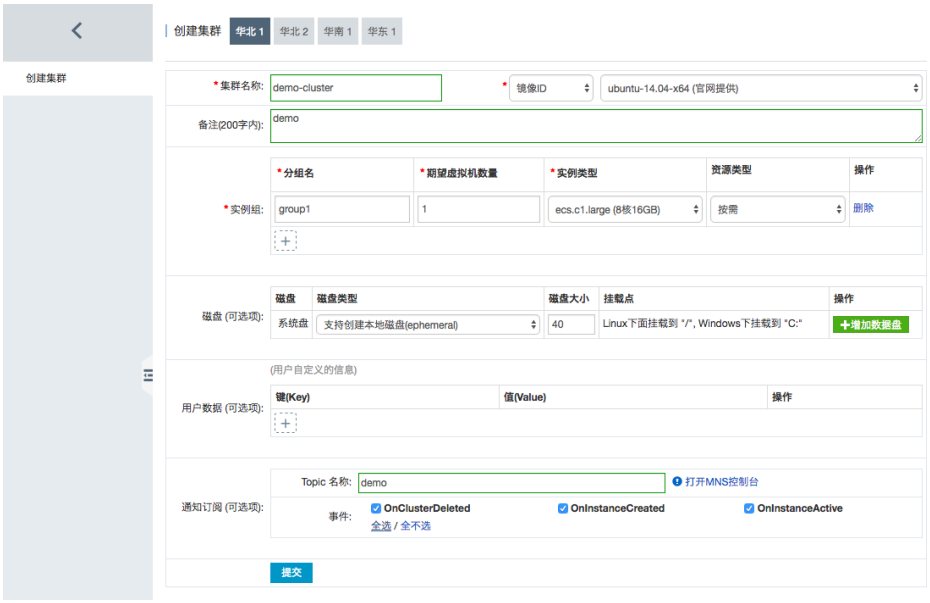
当作业状态为“停止”、“成功”或“失败”时，可以进行删除操作。

同时，还支持批量操作“重启”、“删除”或者“运行”。

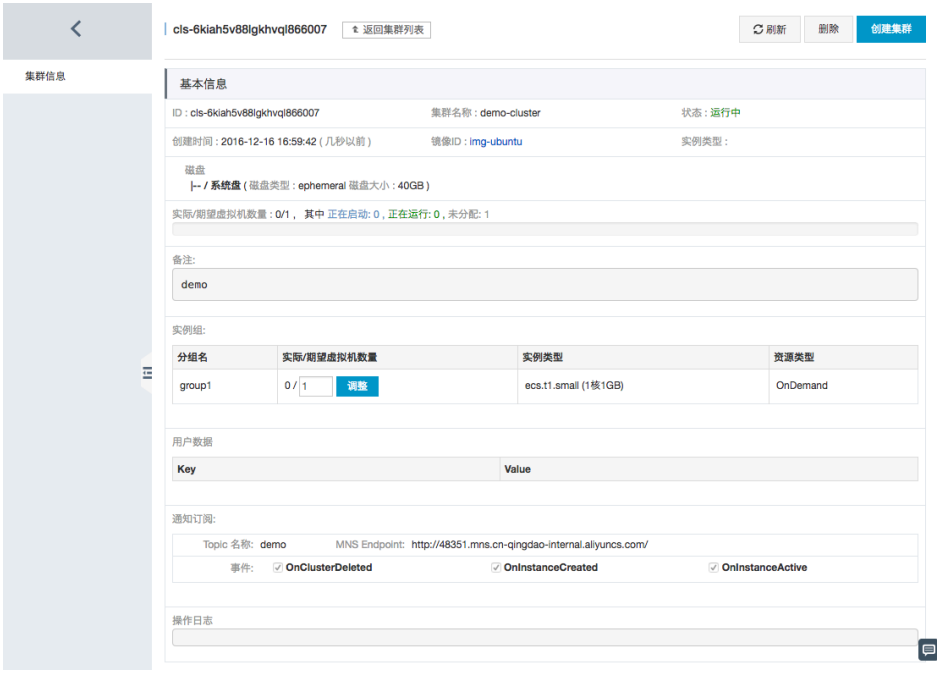
集群管理

1. 集群列表管理

2. 创建集群

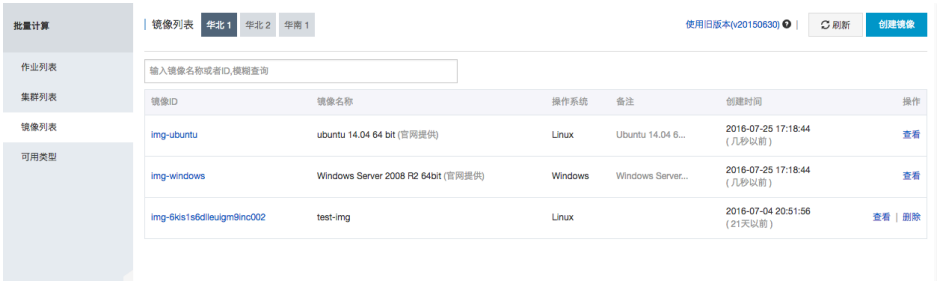


3. 查看集群详细情况



镜像管理

1. 镜像列表



2. 镜像详情

3. 创建镜像

App

前言

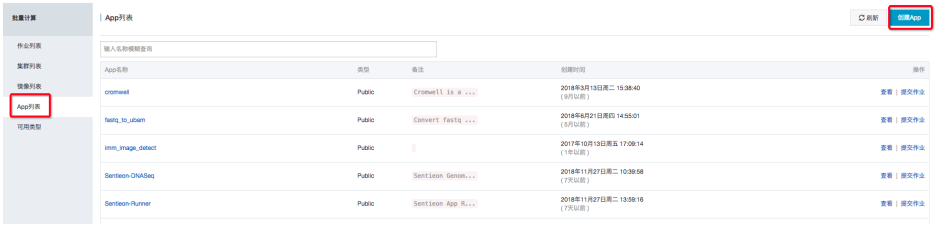
App 是什么

App 是批量计算中资源配置的模板，包括使用什么镜像、什么实例类型、VM 个数。镜像中封装了运行作业的程序或算法，使用 App 提交作业，只需要指定输入数据和输出路径即可以运行作业，而不用关心上述的资源配置以及程序运行的细节。

- 工作原理
  - 创建 App：创建 App 时，将运行作业需要的软件或脚本安装在自定义的镜像中，并设置资源的默认配置，以及输入输出的格式。
  - 提交 App 作业：提交作业时，按照上述资源配置启动虚拟机镜像或 Docker 镜像，使用用户输入的数据运行软件或脚本，并将输出结果存储在用户指定的持久化存储中。
- 运行环境
  - 镜像类型：App 允许用户通过自定义虚拟机镜像（VM 镜像）或者 Docker 镜像的方式对运行环境进行高度定制。
  - 操作系统：App 可以支持 Windows 和 Linux 操作系统。
- 持久化存储
  - 当前 App 支持对象存储 OSS 作为输入输出数据的持久化存储，后续会支持 NAS。
  - 用户的程序、自定义 Docker 镜像、作业的运行日志存储在 OSS 中。
- App的分类
  - 公有 App: 批量计算官方提供，按照 Region 部署，对 Region 内的所有用户都可见，所有用户都可以提交公有 App 的作业。
  - 私有 App: 用户自己创建，只有创建者可见，并且可以对 App 做删除和修改操作，以及提交作业。

如何创建 App

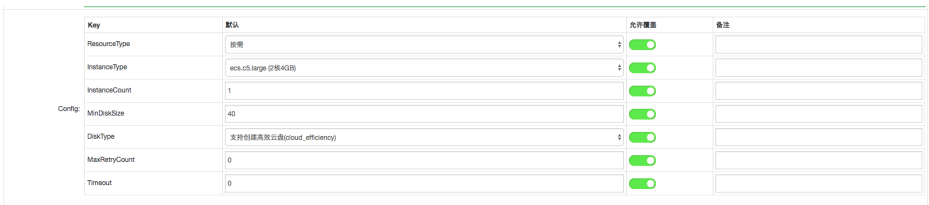
下面以控制台操作为例，介绍如何创建一个 App。



在批量计算控制台，通过 App 列表的创建 App 按钮进入创建页面，各参数的含义如下：

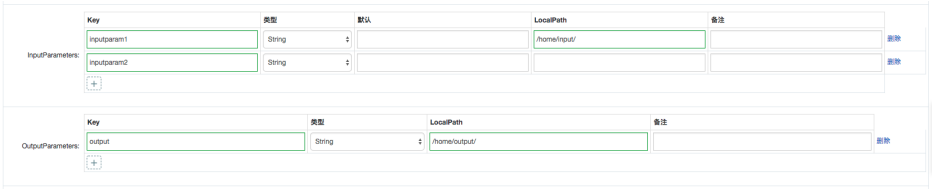
- Name：App 的名称【必填参数】。
- Daemonize：在 App 执行时，是否只需要启动一次并保持后台运行，而不是每次都要重新启动。默认值 False【必填参数】。
- CommandLine：执行 App 时的命令行，可以是运行一个程序或脚本【必填参数】。
- 镜像类型：App 的运行环境，可以是 VM，或者是 Docker【必填参数】。
    - VM：虚拟机镜像，需要填充镜像 ID，可以是批量计算官方提供的镜像类型，也可以是自定义的镜像。
    - Docker：docker 镜像，需要填充 Docke 镜像名称和 Registry 路径。
- Description：App 的描述信息【选填参数】。

- Config：App 的配置信息【选填参数】，包含了如下参数：
    - ResourceType：资源类型，有按需、包月和竞价类型。
    - InstanceType：App 运行的实例类型。
    - InstanceCount：App 运行的实例个数。
    - MinDiskSize：App 运行的最小系统盘大小。
    - DiskType：App 运行的盘类型。
    - MaxRetryCount：App 的某个实例失败后最大的重试次数。
    - Timeout：App 的实例运行的超时时间。



注意：以上 Config 中的所有参数，创建 App 时通过默认值指定为某种配置，但是可以设置允许覆盖（OverWritable），表示在提交这个 App 的作业时是否可以用新的配置覆盖原有配置。提交作业时如果用户不指定这些参数，任务运行时会使用默认配置，创建 AutoCluster，运行完成后自动释放掉。对于允许覆盖的（Overwriteable）参数，提交作业时可以使用自定义的配置来覆盖默认值。

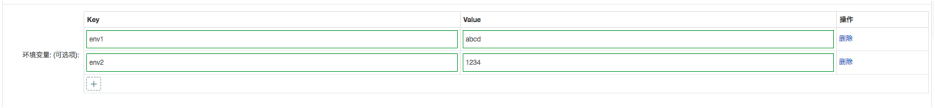
- InputParameters：App 的自定义输入参数。创建 App 时可以自定义一个或多个 String 或 Number 类型的入参。如果是来自 OSS 的输入数据，还可以通过指定 LocalPath 和本地路径做映射。
- OutputParameters：App 的自定义输出数据的路径。创建 App 时可以自定义一个或多个 String 或 Number 类型的输出参数。同样的，如果是来自 OSS 的路径，可以通过指定 LocalPath 和本地路径做映射。



注意：对于输入输出参数，可以用环境变量的方式来使用，比如：

- 命令行访问：在 App 的命令行中采用 ${inputparam} 的方式传入每一个 input/output 参数，对于有 LocalPath 的路径，在命令行执行时，批量计算会将其替换成本地路径。
- 代码中使用：如果 App 的命令行是执行一段代码，可以在代码中用相关的接口获取环境变量。

- EnvVars：环境变量。使用 map<String, String> 的形式提供的环境变量，可以在 App 运行中使用。



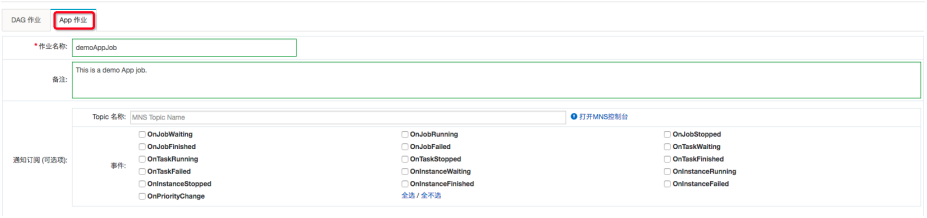
关于环境变量的使用，参考 **环境变量** 中的详细介绍。

如何提交App作业

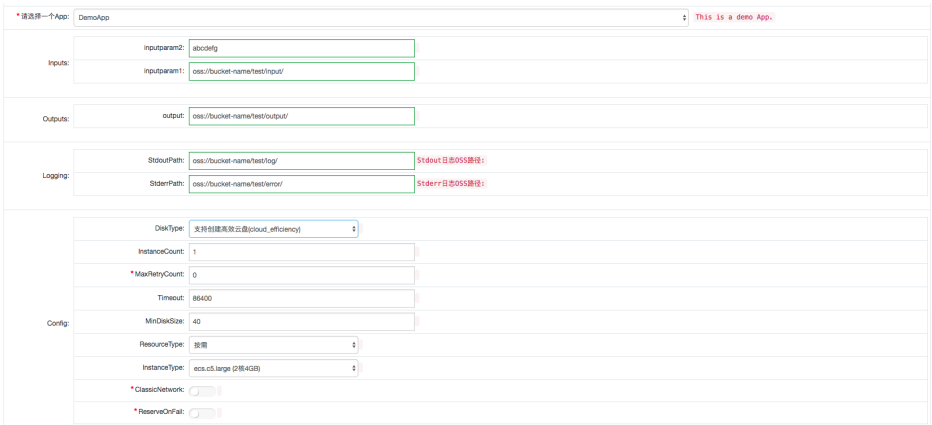
通过批量计算控制台作业列表的提交作业按钮进入作业提交页面。

在作业提交页面选择 App 作业，并选择要使用的 App。各参数的含义如下：

- 作业名称：作业的名称【必填参数】。
- 备注：作业的备注信息【选填参数】。
- 通知订阅：消息通知配置，用户指定消息事件 Notification【选填参数】。



- App 信息：首选要选择提交作业的 App，然后填充 App 的信息，分为四个方面：
  - Inputs：App 作业的输入参数，具体参数由 App 定义，可以是 OSS 路径，也可以是其他自定义参数。
  - OutPuts：App 作业的输出参数，具体参数由 App 定义，可以是 OSS 路径也可以是其他自定义参数。
  - Logging：App 作业的日志路径。
    - StdoutPath：App 作业的标准输出日志的 OSS 路径。
    - StderrPath：App 作业的标准错误日志的 OSS 路径。
  - Config：
    - App定义中的7个配置项，可使用默认配置，也可以根据需求自定义。
    - ClassicNetwork：是否使用经典网络，推荐使用 VPC，所以这里默认值是 False。
    - ReserveOnFail：当任务失败时，运行作业的集群环境是否需要保留，用来调查问题，默认 False，可以根据自己的需求打开。



作业提交成功后，可以在批量计算控制台查看作业的运行状态，等待作业完成。

参考示例

下面使用 Python SDK 的方式，创建一个 App，作用是拷贝一个输入路径 inputparam1 的内容到一个输出路径 output，并打印自定义的环境变量，App 相关定义如下：

- 输入信息
  - inputparam1：自定义输入参数1，OSS 路径，映射到本地的 /home/input/ 目录。
  - inputparam2：自定义输入参数2，非 OSS 路径，这里无具体含义，用于演示输入参数用法。
- 输出信息
  - outout：自定义输出参数，OSS 路径，映射到本地的 /home/output/ 目录。
- 环境变量：
  - env1：自定义环境变量，这里无具体含义，用于演示环境变量用法。

使用 Python SDK 创建基于 VM 镜像的 App

```
#encoding=utf-8
import sys
from batchcompute import Client, ClientError
from batchcompute import CN_BEIJING as REGION
from batchcompute.resources import (
JobDescription, TaskDescription, DAG, AutoCluster, GroupDescription, ClusterDescription, AppDescription
)
ACCESS_KEY_ID='xxxx' # 填写您的 ACCESS_KEY_ID
ACCESS_KEY_SECRET='xxxx' # 填写您的 ACCESS_KEY_SECRET

def main():
try:
client = Client(REGION, ACCESS_KEY_ID, ACCESS_KEY_SECRET)

app_desc = {
"Name":"DemoApp_vm",
"Daemonize":False,
"VM":{
"ECSImageId":"img-ubuntu-vpc",
},
"CommandLine":"/bin/bash -c 'cp -r ${inputparam1} ${output} && echo ${inputparam2} && echo $env1'",
"InputParameters":{
"inputparam1":{
"Description":"",
"Type":"String",
"Default":"",
"LocalPath":"/home/input/"
},
"inputparam2":{
```

```
"Description":"",
"Type":"String",
"Default":"",
"LocalPath":""
}
},
"OutputParameters":{
"output":{
"Description":"",
"Type":"String",
"LocalPath":"/home/output/"
}
},
"Config":{
"ResourceType":{
"Default":"OnDemand",
"Overwritable":True
},
"InstanceType":{
"Description":"",
"Default":"ecs.c5.large",
"Overwritable":True
},
"InstanceCount":{
"Description":"",
"Default":1,
"Overwritable":True
},
"MinDiskSize":{
"Description":"",
"Default":40,
"Overwritable":True
},
"DiskType":{
"Description":"",
"Default":"cloud_efficiency",
"Overwritable":True
},
"MaxRetryCount":{
"Description":"",
"Default":0,
"Overwritable":True
},
"Timeout":{
"Description":"",
"Default":86400,
"Overwritable":True
}
},
"EnvVars":{
"env1":"abcd"
}
}

appName = client.create_app(app_desc).Name
print('App created: %s' % appName)
```

```
except ClientError, e:
print (e.get_status_code(), e.get_code(), e.get_requestid(), e.get_msg())
if __name__ == '__main__':
sys.exit(main())
```

使用 Python SDK 创建基于 Docker 镜像的 App

```
#encoding=utf-8
import sys
from batchcompute import Client, ClientError
from batchcompute import CN_BEIJING as REGION
from batchcompute.resources import (
JobDescription, TaskDescription, DAG, AutoCluster, GroupDescription, ClusterDescription, AppDescription
)
ACCESS_KEY_ID='xxxx' # 填写您的 ACCESS_KEY_SECRET
ACCESS_KEY_SECRET='xxxxxx' # 填写您的 ACCESS_KEY_SECRET

def main():
try:
client = Client(REGION, ACCESS_KEY_ID, ACCESS_KEY_SECRET)

app_desc = {
"Name":"DemoApp_Docker",
"Daemonize":False,
"Docker":{
"Image":"localhost:5000/dockerimagename",
"RegistryOSSPath":"oss://bucket-name/dockers/"
},
"CommandLine":"/bin/bash -c 'cp -r ${inputparam1} ${output} && echo ${inputparam2} && echo $env1'",
"InputParameters":{
"inputparam1":{
"Description":"",
"Type":"String",
"Default":"",
"LocalPath":"/home/input/"
},
"inputparam2":{
"Description":"",
"Type":"String",
"Default":"",
"LocalPath":""
}
},
"OutputParameters":{
"output":{
"Description":"",
"Type":"String",
"LocalPath":"/home/output/"
}
},
"Config":{
"ResourceType":{
"Default":"OnDemand",
"Overwritable":True
```

```
},
"InstanceType":{
"Description":"",
"Default":"ecs.c5.large",
"Overwritable":True
},
"InstanceCount":{
"Description":"",
"Default":1,
"Overwritable":True
},
"MinDiskSize":{
"Description":"",
"Default":40,
"Overwritable":True
},
"DiskType":{
"Description":"",
"Default":"cloud_efficiency",
"Overwritable":True
},
"MaxRetryCount":{
"Description":"",
"Default":0,
"Overwritable":True
},
"Timeout":{
"Description":"",
"Default":86400,
"Overwritable":True
}
},
"EnvVars":{
"env1":"abcd"
}
}

appName = client.create_app(app_desc).Name
print('App created: %s' % appName)
except ClientError, e:
print (e.get_status_code(), e.get_code(), e.get_requestid(), e.get_msg())
if __name__ == '__main__':
sys.exit(main())
```

使用 Python SDK 提交 App 作业

```
#encoding=utf-8
import sys
from batchcompute import Client, ClientError
from batchcompute import CN_BEIJING as REGION
from batchcompute.resources import (
JobDescription, TaskDescription, DAG, AutoCluster, GroupDescription, ClusterDescription, AppDescription,
AppJobDescription
)
```

```
ACCESS_KEY_ID='xxxx' # 填写您的 ACCESS_KEY_ID
ACCESS_KEY_SECRET='xxxx' # 填写您的 ACCESS_KEY_SECRET
LOG_PATH = 'oss://bucket-name/bc_test/log/'
ERR_PATH = 'oss://bucket-name/bc_test/error/'

def main():
try:
client = Client(REGION, ACCESS_KEY_ID, ACCESS_KEY_SECRET)
job_desc = AppJobDescription()

# Create job description.
job_desc.Name = "App_demo_test_job_docker"
job_desc.Description = "Test of create app job."
job_desc.Type = 'App'

job_desc.App.AppName = 'DemoApp_Docker'

job_desc.App.Config.InstanceType = "ecs.sn2.medium"

job_desc.App.Inputs["inputparam1"] = "oss://bucket-name/bc_test/input/";
job_desc.App.Inputs["inputparam2"] = "abcd1234";
job_desc.App.Outputs["output"] = "oss://bucket-name/bc_test/output/"

job_desc.App.Logging.StdoutPath = LOG_PATH
job_desc.App.Logging.StderrPath = ERR_PATH

job_id = client.create_job(job_desc).Id
print('App job created: %s' % job_id)
except ClientError, e:
print (e.get_status_code(), e.get_code(), e.get_requestid(), e.get_msg())
if __name__ == '__main__':
sys.exit(main())
```