批量计算

用户指南

为了无法计算的价值 | 阿里云

用户指南

准备阶段

集群镜像

1. 背景

提交作业或者创建集群时，批量计算将使用您指定的镜像来启动实例。为了方便集群管理和运行状态收集，批量计算提供了预置管理软件的基础镜像。您可以直接使用这些基础镜像，也可以通过在基础镜像内安装业务所需的软件制作自定义镜像。

2. 说明

使用批量计算服务对您所使用镜像的要求是：

- 镜像为批量计算提供的官方基础镜像；
- 镜像为基于基础镜像制作并完成镜像注册（共享）的自定义镜像。

3. 使用

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。

3.1 创建 ECS 实例

请直接点击相应的基础镜像链接：

Ubuntu 14.04	Ubuntu 16.04	CentOS 6.8	CentOS 7.4	Windows 2008	Windows 2012	Windows 2016
link	link	link	link	link	link	link

单击**立即购买**，进入 ECS 购买页面，按照下面的建议配置实例：

- 付费类型选择按量付费；
- 地域与批量计算服务保持一致；
- 可用区选择随机分配；
- 实例规格建议至少 4 核 8GB；
- 系统盘按您所需软件大小选择；
- 选择一个已有 VPC 或创建一个新的 VPC；
- 选择一个已有的安全组或创建一个新安全组。

3.2. 安装用户软件

实例创建出来后，请 重置实例密码 并重启实例。待实例处于运行状态后，通过远程连接登入到该实例。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。

3.3. 创建镜像

请按如下步骤完成镜像创建：

- 待所有程序安装完成后，在 ECS 控制台创建磁盘快照。参考 创建快照 ；
- 待上述快照创建完成后，在 ECS 控制台创建自定义镜像。参考 创建自定义镜像 。

3.4. 镜像验证

自定义镜像制作完成后，以该镜像创建一个 ECS 实例。实例成功启动后进行远程连接，登录后请务必自行检查安装的软件是否能正常运行。如果您使用的是 Linux 操作系统，还需要检测 cloud-init 服务的完整性，请执行命令cloud-init -h查询软件能否正常运行，若抛异常则说明 cloud-init 服务被破坏，需要联系批量计算运维人员对镜像进行修复操作。

3.5. 注册镜像

需要注意的是，您之前所做的全部操作都是在 ECS 之上完成的，若在批量中使用该自定义镜像还需要完成镜像注册的工作。打开批量计算控制台 创建镜像 ，将制作好的 ECS 镜像（以 m- 为前缀）在批量计算进行注册操作，之后提交作业或创建集群均可以使用批量计算镜像（以 img- 为前缀）。

集群网络

1. 背景

批量计算只支持 Vpc 集群的创建，即实例均创建在 Vpc 内。同一个 Vpc 内的集群实例可以通过私网 IP 互联，并且可以访问您在该 Vpc 内的其他阿里云服务。如您需要自建 Server 管理批量计算集群实例，只需在同一 Vpc 内部署相关服务即可。

2. 说明

使用用户 Vpc 主要包含以下四点限制：

- 大小限制：CidrBlock 指定的网段空间必须包含在您指定的 Vpc 网段内；
- 网段限制：CidrBlock 只能在以下三个区间范围内：
  - 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；
- 其他限制：在集群存在期间请不要随意操作批量计算自动创建出的 VSwitch。

3. 使用

您在创建集群或作业时，可以指定在您已有的 Vpc 内创建，此时需要提供用户 Vpc (VpcId) 和 Vpc 内规划给批量计算使用的网段 (Cidrblock) 。当然，如果您还没有 Vpc，也可以只提供 Cidrblock，批量计算会为您创建默认 Vpc。

以下我们将展示通过 SDK 和命令行工具指定用户 Vpc，VpcId为vpc-xxyyzz，CidrBlock为192.168.0.0/16。

3.1. SDK

使用 Python SDK 创建集群指定用户 Vpc 样例：

```
from batchcompute.resources import (
ClusterDescription, Configs, Networks, VPC
)

cluster_desc = ClusterDescription()
configs = Configs()
networks = Networks()
vpc = VPC()

vpc.CidrBlock = '192.168.0.0/16'
vpc.VpcId = 'vpc-xxyyzz'

networks.VPC = vpc
configs.Networks = networks
cluster_desc.Configs = configs
```

3.2. 命令行工具

使用命令行工具创建集群指定用户 Vpc 样例：

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

挂载磁盘

1. 背景

您在创建批量计算计算节点时，因系统盘大小限制，若有较大本地数据需求，建议挂载数据盘。挂载数据盘后，对挂载目录里数据的读写行为将和读写本地数据完全相同。

2. 说明

数据盘挂载主要包含以下四点限制：

- 类型限制：只支持 cloud_efficiency 和 cloud_ssd两种类型数据盘；
- 大小限制：默认最大支持 1000G 数据盘，如有特殊需求请提交工单；
- 约束限制：需保持系统盘和数据盘类型保持一致；
- 系统限制：必须指定 MountPoint，Linux 下可挂载到目录，Windows 下只能挂载到驱动，如 E:。

3. 使用

使用磁盘挂载功能必须同时指定数据盘大小、数据盘类型和挂载点。

以下我们将展示通过 SDK 和命令行工具挂载数据盘样例，数据盘类型为cloud_efficiency；大小为 500G；挂载点为 /home/my-data-disk。

3.1. SDK

使用 Python SDK 创建集群挂载数据盘样例：

```
from batchcompute.resources import (
ClusterDescription, Configs
)

cluster_desc = ClusterDescription()
configs = Configs()

configs.add_data_disk(500, 'cloud_efficiency', '/home/my-data-disk')

cluster_desc.Configs = configs
```

3.2. 命令行工具

使用命令行工具创建集群挂载数据盘样例：

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

说明：

- 系统盘挂载格式： system:系统盘类型:系统盘大小
- 数据盘挂载格式： data:数据盘类型:数据盘大小:挂载点

挂载NAS

1. 背景

绝大部分计算模型下，客户数据直接存储于云端 NAS 里。为了方便客户读写云端计算数据，批量计算根据用户提供的挂载信息，自动将 NAS 的挂载点挂载到本地目录。完成 NAS 挂载后，对挂载目录里数据的读写行为将和读写本地数据完全相同。

2. 说明

- 网络限制：批量计算仅支持专有网络 (Vpc) 类型的挂载点，且集群必须和待挂载的 NAS 在同一个专有网络 (Vpc) 内；
- 文件系统限制：批量计算仅支持NFS类型的 NAS 文件系统；
- 格式限制：不同操作系统挂载略有差异，Windows 在 NAS 文件系统目录后需要加！：
    - Windows 示例：nas://xxx.cn-beijing.nas.aliyuncs.com:/!
    - Linux 示例：nas://xxx.cn-beijing.nas.aliyuncs.com:/

3. 使用

使用 NAS 挂载功能必须同时指定 NAS 源地址 (Source)、本地挂载地址 (Destination) 和是否支持可写位 (WriteSupport)，他们的意义如下：

- Source：以nas://为前缀，后面接上 NAS 挂载点和 NAS 文件系统目录信息，例：nas://xxx.cn-beijing.nas.aliyuncs.com:/!；
- 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，如果用户设置为False 则表示该挂载点不支持写操作；若设置为

True则表示挂载点支持读写操作；写操作会直接将数据写到 NAS 远端服务器上，而不会在本地做缓存。

以下我们将展示通过 SDK 和命令行工具挂载 NAS，其中 NAS 源地址为nas://xxx.cn-beijing.nas.aliyuncs.com:/，本地挂载地址为/home/admin/mydir/，可写位为true。

3.1. SDK

```
from batchcompute.resources import (
ClusterDescription, Configs, Mounts
)

cluster_desc = ClusterDescription()
configs = Configs()
mounts = Mounts()

# For Linux
nas_entry = {
'Source': 'nas://xxx.cn-beijing.nas.aliyuncs.com:/', # Windows 需在最后加 "!"
'Destination': '/home/admin/mydir/',
'WriteSupport': true,
}

mounts.Entries = [nas_entry]
configs.Mounts = mounts
cluster_desc.Configs = configs
```

挂载OSS

1. 背景

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。

2. 说明

挂载类型：

- 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。

命名限制：

- 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。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。

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（在挂载的时候增加 nolock 选项）。但是有些特定的应用程序如 3ds MAX 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。

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。

修改限制：

- 在应用程序运行期间，不应该修改 OSS 上已经挂载的数据，否则可能引起冲突。

访问权限：

- 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。所有对挂载目录的修改操作，都会缓存在本地，您的应用程序可以读取到修改后的内容，但这些数据不会同步到 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。

3. 使用

3.1. 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。

3.1.1 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
```

3.1.2 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/"
}
```

3.1.3 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。

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

3.2 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。

3.2.1 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);
```

3.2.2 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'
```

3.2.3 使用命令行

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

3.3. 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：

3.3.1 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");
```

3.3.2 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"
}
```

环境变量

1. 背景

批量计算为客户提供了系统环境变量和自定义环境变量功能，您可以在代码中直接使用这些环境变量。

2. 说明

目前批量计算支持两种类型运行环境，您可以按需使用：

- VM 环境变量：
  - 作业 ID：BATCH_COMPUTE_DAG_JOB_ID
  - 任务名称：BATCH_COMPUTE_DAG_TASK_ID
  - 实例 ID：BATCH_COMPUTE_DAG_INSTANCE_ID
  - OSS Host：BATCH_COMPUTE_OSS_HOST
  - 服务区域：BATCH_COMPUTE_REGION
  - 集群 ID：BATCH_COMPUTE_CLUSTER_ID
  - 虚拟机 ID：BATCH_COMPUTE_WORKER_ID
- Docker 运行环境：
  - root用户：USER
  - 工作目录：PWD
  - 软件路径：PATH
  - 家目录：HOME
  - 作业 ID：BATCH_COMPUTE_DAG_JOB_ID
  - 任务名称：BATCH_COMPUTE_DAG_TASK_ID
  - 实例 ID：BATCH_COMPUTE_DAG_INSTANCE_ID
  - OSS Host：BATCH_COMPUTE_OSS_HOST
  - 区域：BATCH_COMPUTE_REGION

3. 使用

3.1 SDK

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

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

4. 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。

4.1. SDK

代码片段:

```
env = {
'k1': 'v1',
'k2': 'v2'
}

job_desc['DAG']['Tasks']['my-task']['Parameters']['Command']['EnvVars']=env
```

4.2. 命令行工具

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

消息通知

1. 背景

批量计算使用 MNS 来实现消息通知。用户负责主题 (Topic) 的创建、管理和订阅，并在创建集群或提交作业时指定相关配置。批量计算会依据配置向指定的主题推送消息，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。

2. 说明

2.1. 支持的事件

目前批量计算支持两类事件，您可以按需配置：

- 集群事件：
  - 集群已删除：OnClusterDeleted；

- 实例已创建：OnInstanceCreated；
- 实例已运行：OnInstanceActive；

- 作业事件：
  - 作业等待中：OnJobWaiting；
  - 作业运行中：OnJobRunning；
  - 作业已停止：OnJobStopped；
  - 作业已结束：OnJobFinished；
  - 作业已失败：OnJobFailed；
  - 任务等待中：OnTaskWaiting；
  - 任务运行中：OnTaskRunning；
  - 任务停止中：OnTaskStopped；
  - 任务已结束：OnTaskFinished；
  - 任务已失败：OnTaskFailed；
  - 实例等待中：OnInstanceWaiting；
  - 实例运行中：OnInstanceRunning；
  - 实例已停止：OnInstanceStopped；
  - 实例已结束：OnInstanceFinished；
  - 实例已失败：OnInstanceFailed；
  - 优先级改变：OnPriorityChange。

2.2. 消息格式

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"
}
```

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-
000000005818421800000815000000000D.task.0",
"StderrRedirectPath": "oss://bucket/tests/a44c0ad8-a003-11e6-8f8e-fefec0a80e06/logs/stdout.job-
000000005818421800000815000000000D.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"
}
```

3. 使用

使用消息队列必须同时指定 MNS 主题名称 (Name)、MNS 私网 Endpoint (Endpoint) 和关注的事件 (Events)。

以下我们将展示通过 SDK 展示如何使用 MNS 消息队列，其中 MNS 主题名称为test，MNS私网 Endpoint 为 http://xxx.mns.cn-beijing.aliyuncs.com/，关注的事件为OnClusterDeleted、OnInstanceCreated和 OnInstanceActive。

3.1. SDK

```
from batchcompute.resources import (
ClusterDescription, Notification, Topic
)

cluster_desc = ClusterDescription()
notification = Notification()
topic = Topic()

topic.Name = 'test'
topic.Endpoint = 'http://xxx.mns.cn-beijing.aliyuncs.com/'
topic.Events = ["OnClusterDeleted", "OnInstanceCreated", "OnInstanceActive"]

notification.Topic = topic
cluster_desc.Notification = notification
```

Docker 支持

背景知识

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。

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：

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
```

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。

CR 镜像管理

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 SDK 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。

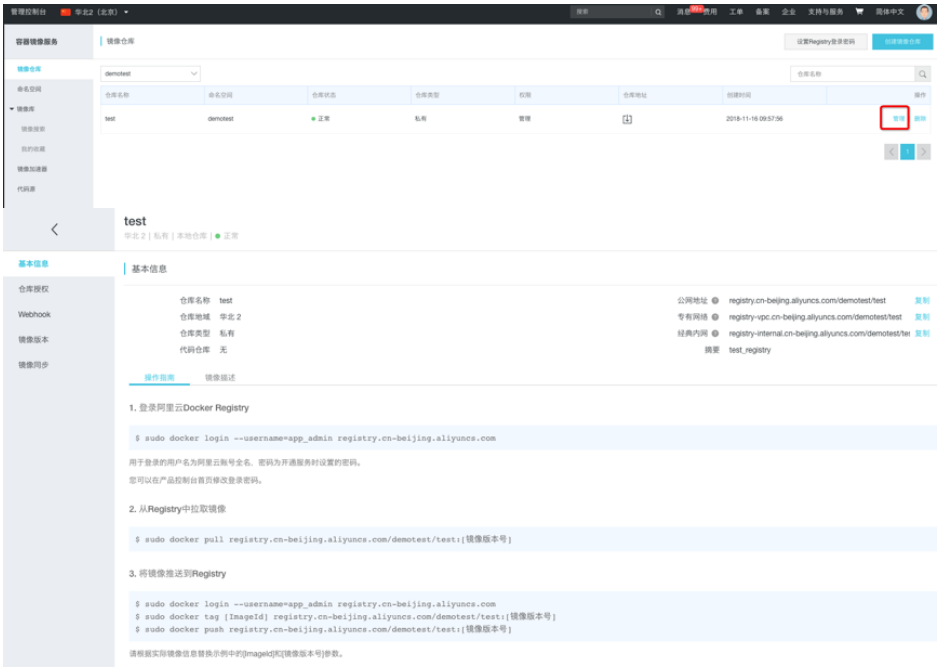
1.2 创建名字空间域

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

登录到镜像所在的 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。

```
root@iZ2zecs0w4zpyoqkchr2tyZ:~# docker tag 7e7a7a066cc3 registry.cn-beijing.aliyuncs.com/demotest/test:0.1
root@iZ2zecs0w4zpyoqkchr2tyZ:~# docker images
REPOSITORY                                                    TAG                 IMAGE ID            CREATED             SIZE
registry                                                      2                   a1857487b127        6 months ago        297 MB
golang                                                        1.7-alpine          974aa102bae2        16 months ago       241 MB
registry.cn-beijing.aliyuncs.com/demotest/test                0.1                 7e7a7a066cc3        2 years ago         561 MB
localhost:5000/bio-speedseq                                   0.1.0               7e7a7a066cc3        2 years ago         561 MB
registry-vpc.cn-beijing.aliyuncs.com/batchcompute_test/bio-speedseq   0.1.0               7e7a7a066cc3        2 years ago         561 MB
registry-vpc.cn-beijing.aliyuncs.com/batchcompute_test/bj-test        0.1.0               7e7a7a066cc3        2 years ago         561 MB
```

2.3 推送镜像

```
root@iZ2zecs0w4zpyoqkchr2tyZ:~# sudo docker push registry.cn-beijing.aliyuncs.com/demotest/test:0.1
sudo: unable to resolve host iZ2zecs0w4zpyoqkchr2tyZ
The push refers to a repository [registry.cn-beijing.aliyuncs.com/demotest/test]
5f70bf18a086: Pushed
b364fdaa8ca8: Pushed
ee0f7ecceedf: Pushed
66b61bf624c7: Pushed
1255f0fad851: Pushed
3bd5a069ac09: Pushed
fef0f9958347: Pushed
0.1: digest: sha256:51b44d509bd3c1738dc1a437a27fa2242264fcb7d225262a93b8bdfdbb45f654 size: 1993
root@iZ2zecs0w4zpyoqkchr2tyZ:~#
```

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

OSS 镜像管理

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

```
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
```

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}
```

集群管理

按需集群

1. 背景

按需集群是指您按日常业务量变化，可按需调节集群规模的集群类型。该类型集群适用于绝大部分业务波峰明显的场景，阿里云批量计算服务依托强大的弹性计算能力，在渲染、基因、金融、游戏、科学计算等领域有极为广阔的应用。

2. 限制

各账号按需实例均存在配额限制，您可以直接打开批量计算控制台，切换到对应区域查看您可使用的实例类型。若您有超过 50 台以上的实例需求，需要先提交工单来提高配额。

3. 使用

使用竞价实例必须针对以下几个参数进行设置：

- 资源类型（ResourceType）：按需实例的前提是资源类型必须设置为OnDemand；
- 实例类型（InstanceType）：按需集群的实例类型，您可以在批量计算控制台查看。

以下我们将展示通过 SDK 创建集群，ResourceType 设置为OnDemand，InstanceType 设置为 ecs.sn1ne.xlarge。

3.1 SDK

使用 Python SDK 创建集群设置按需实例样例：

```
from batchcompute.resources import (
ClusterDescription, GroupDescription
)
```

```
cluster_desc = ClusterDescription()
group_desc = GroupDescription()

group_desc.ResourceType = 'OnDemand'
group_desc.InstanceType = 'ecs.sn1ne.xlarge'

cluster_desc.add_group('group_name', group_desc)
```

竞价集群

1. 背景

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。它的定价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价格浮动的特性，在合适的时间购买该竞价实例资源，从而降低计算成本。

2. 限制

使用竞价实例必须明确以下几点：

- 竞价实例的核时单价会浮动，会影响成本的精确预估；
- 竞价实例存在被系统释放的可能。

3. 使用

使用竞价实例必须针对以下几个参数进行设置：

- 资源类型（ResourceType）：使用竞价实例的前提是资源类型必须设置为Spot；
- 竞价策略（SpotStrategy）：设置为SpotAsPriceGo表示跟随当前系统出价；设置为SpotWithPriceLimit表示是否分配机器取决于当前价格与您出价情况；
- 竞价上限（SpotPriceLimit）：只有 SpotStrategy 被设置为SpotWithPriceLimit才有效，支持最大 3 位小数。

以下我们将展示通过 SDK 和命令行工具创建集群并设置竞价实例，ResourceType 设置为 Spot，SpotStrategy 设置为SpotWithPriceLimit，SpotPriceLimit 设置为 0.1。

3.1 SDK

使用 Python SDK 创建集群设置竞价实例样例：

```
from batchcompute.resources import (
ClusterDescription, GroupDescription
)

cluster_desc = ClusterDescription()
group_desc = GroupDescription()

group_desc.ResourceType = 'Spot'
group_desc.SpotStrategy = 'SpotWithPriceLimit'
group_desc.SpotPriceLimit = 0.1

cluster_desc.add_group('group_name', group_desc)
```

3.2 命令行工具

使用命令行工具创建集群设置竞价实例样例：

```
bcs cc cluster_1 -t ecs.sn1.large --resource_type Spot --spot_price_limit 0.1
```

修改集群竞价单价：

```
bcs uc cls-xxxx --spot_price_limit 0.1
```

包月集群

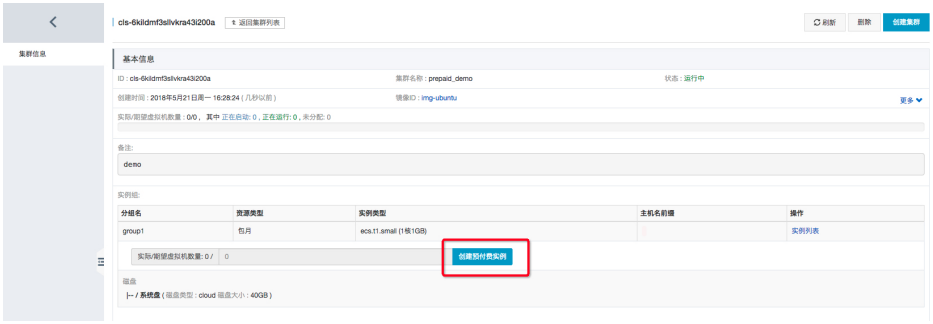
1. 创建集群

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。

批量计算（包年包月）

选择好配置后，点击”购买”按钮，按提示操作即可。

作业管理

任务类型

1. 背景

一个作业（Job）由一组任务（Task）及其依赖关系组成，每个任务可以有一个或多个执行实例（Instance）。具体详情看名词解释。目前的任务类型分为两种：并发任务和 DAG(Directed Acyclic Graph)任务。

2. 任务概述

2.1 并发任务

作业中的一个任务可以指定在多个实例上运行程序，这些实例运行的任务程序都是一样的，但是可以处理不同的数据。

2.2 DAG任务

作业中的多个任务之间可以有 DAG 依赖关系。即前面的任务运行完成后, 后面的任务才开始运行。

3. 任务实现

这两种任务是在提交的 Job 中指定相关字段实现的，下面以 Python SDK 为例给出实现方式，代码的完整程序见快速开始。

3.1 并发任务实现

在提交的 Job 中，填写 InstanceCount 字段。指明任务需要的实例数。该字段就是实现任务的并发功能。

```
from batchcompute.resources import (
JobDescription, TaskDescription, DAG
)
# create my_task
my_task = TaskDescription()
my_task.InstanceCount = 3 #指定需要实例数:3台VM
```

如果并发任务需要处理不同片段的数据，这个时候在需要运行的任务程序中 使用环境变量: BATCH_COMPUTE_DAG_INSTANCE_ID（实例 ID）来区分，就可以处理不同片段的数据。下面的示例程序是快速开始的count代码，假设输入数据已经放在oss中。您需要下载oss的sdk。

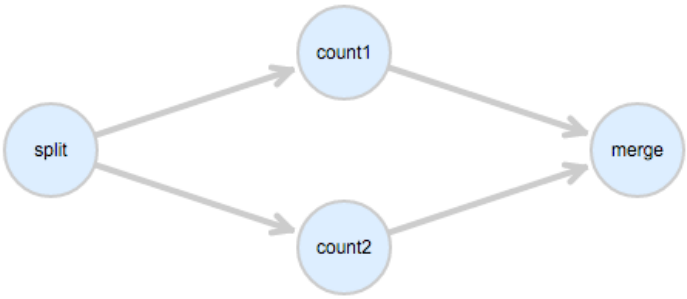
```
import oss2 #oss sdk
from conf import conf
import os
```

```
import json

endpoint = os.environ.get('BATCH_COMPUTE_OSS_HOST') #OSS Host
auth = oss2.Auth(conf['access_key_id'], conf['access_key_secret'])

def download_file(oss_path, filename):
(bucket, key) = parse_oss_path(oss_path)
bucket_tool = oss2.Bucket(auth, endpoint, bucket)
bucket_tool.get_object_to_file(key, filename)

def upload_file(filename, oss_path):
(bucket, key) = parse_oss_path(oss_path)
bucket_tool = oss2.Bucket(auth, endpoint, bucket)
bucket_tool.put_object_from_file(key,filename)

def put_data(data, oss_path):
(bucket, key) = parse_oss_path(oss_path)
bucket_tool = oss2.Bucket(auth, endpoint, bucket)
bucket_tool.put_object(key, data)

def parse_oss_path(oss_path):
s = oss_path[len('oss://'):]
[bucket, key] = s.split('/',1)
return (bucket,key)

def main():
# instance_id: should be start from 0
instance_id = os.environ['BATCH_COMPUTE_DAG_INSTANCE_ID']
data_path = conf['data_path']
split_results = 'split_results'
filename = 'part_%s.txt' % instance_id
pre = data_path[0: data_path.rfind('/')]
print('download form: %s/%s/' % (pre, split_results))

# 1. download a part
download_file('%s/%s/%s.txt' % (pre, split_results, instance_id ), filename)
# 2. parse, calculate
with open(filename) as f:
txt = f.read()
m = {
'INFO': 0,
'WARN': 0,
'ERROR': 0,
'DEBUG': 0
}
for k in m:
m[k] = len(re.findall(k, txt))
print(m)
# 3. upload result to oss
upload_to = '%s/count_results/%s.json' % (pre, instance_id )
print('upload to %s' % upload_to)
put_data(json.dumps(m), upload_to)
```

3.2 DAG任务实现

在提交的job中，填写 Dependencies 字段。指明任务之间的依赖关系。下面的图中，首先理清各个任务之间

的依赖关系，count1 和 count2 是并行的任务，它们依赖 split 任务，merge任务依赖 count1 和 count2。



依据上面的依赖关系，在Job中可以这样描述：

```
from batchcompute.resources import (
JobDescription, TaskDescription, DAG, AutoCluster
)

job_desc = JobDescription()
#以下省略task的描述内容
split = TaskDescription()
count1 = TaskDescription()
count2 = TaskDescription()
merge = TaskDescription()

task_dag = DAG()
task_dag.add_task(task_name="split", task=split)
task_dag.add_task(task_name="count1", task=count1)
task_dag.add_task(task_name="count2", task=count2)
task_dag.add_task(task_name="merge", task=merge)
task_dag.Dependencies = {
'split': ['count1', 'count2'],
'count1': ['merge'],
'count2': ['merge']
}

job_desc.DAG = task_dag
```

整个作业的任务执行顺序是：

- 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。

作业类型

1 背景

批量计算依据用户使用集群方式的区别，将作业分为固定集群作业，AutoCluster作业，AttachCluster作业三种类型。下面分别介绍三种作业的优缺点，您可依据业务需求选择相应的作业类型。

2 作业概述

2.1 固定集群作业

优点：集群支持分布式缓存，适用大规模作业的场景

缺点：需要用户花费精力管理集群资源生命周期

2.2 AutoCluster作业

优点：用户无需花费精力管理集群，由批量计算自动管理集群生命周期

缺点：集群间无法共享分布式缓存数据，访问存储压力大，占用资源多，适用小规模作业的场景

2.3 AttachCluster作业

批量计算最新推出的作业类型。使用方法请参考：AttachCluster最佳实践

优点：

- 支持资源弹性伸缩，用户无需管理集群生命周期
- 分布式缓存提高IO性能，减少带宽占用
- 极大减少集群数量，节省资源
- 减少作业等待时间，使用方式简单，迁移成本小。

缺点：用户首次使用需创建一个集群，然后才能提交作业。

3 使用区别

3.1 作业管理

	AttachCluster	AutoCluster	固定集群
创建	需要事先创建集群，在AutoCluster配置里填写当前task需要的配置。	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	手动创建的集群需要自己删除	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	提交作业时需要在AutoCluster字段中指定集群ID	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

3.2 作业格式

- 固定集群模式，直接填写已经创建好的集群

```
from batchcompute.resources import (
JobDescription, TaskDescription
)
job_desc = JobDescription()

task_desc = TaskDescription()
task_desc.ClusterId = "cls-xxxx"
task_desc.AutoCluster.ClusterId = ""

job_desc.DAG.add_task ("cluster", task_desc)
```

- Auto-Cluster模式

```
from batchcompute.resources import (
JobDescription, TaskDescription
)
job_desc = JobDescription()

task_desc = TaskDescription()
task_desc.ClusterId = "
task_desc.AutoCluster.ClusterId = ""

job_desc.DAG.add_task ("auto_cluster", task_desc)
```

- Attach-Cluster模式：

```
from batchcompute.resources import (
JobDescription, TaskDescription
)
job_desc = JobDescription()

task_desc = TaskDescription()
task_desc.ClusterId = ""
task_desc.AutoCluster.ClusterId = "cls-xxxx"

job_desc.DAG.add_task ("attach_cluster", task_desc)
```

3.3 适用场景

固定集群作业

如果您有很多作业要处理，可以考虑使用固定集群。 比如提交100个作业，您可以创建一个10台VM的固定集群，将100个作业全部提交到这个集群即可, 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，你需要手动释放掉集群。

AutoCluster作业

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。

AttachCluster作业

您有很多AutoCluster的作业时（成千上万级别），需要选用AttachCluster。提交作业时需要指定AutoCluster配置中的集群信息，作业运行的时候会被Attach到指定集群上。集群会自动申请和释放所需资源。

提交作业

1. 提交作业的方法

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。

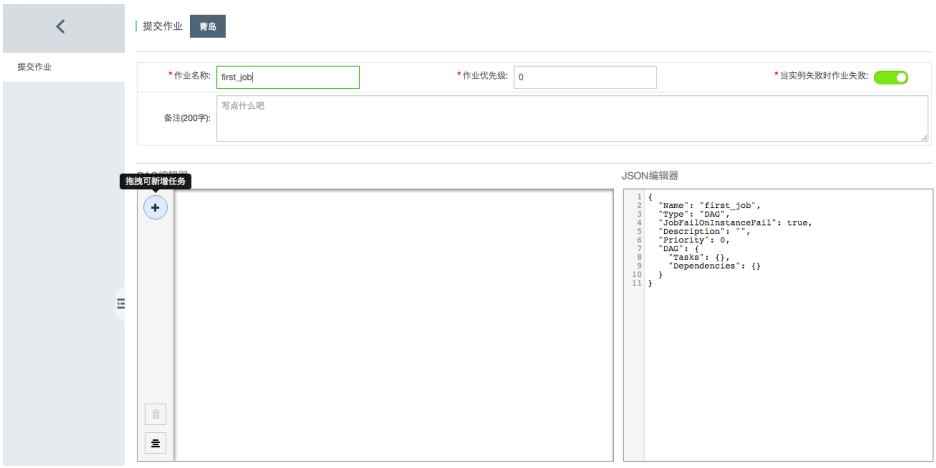
> 如果还没有开通 OSS，请先开通。
>
> 还需要创建 Bucket，假设创建了 Bucket 名称为 mybucket
>
> 然后在这个 Bucket 下创建一个目录: test

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

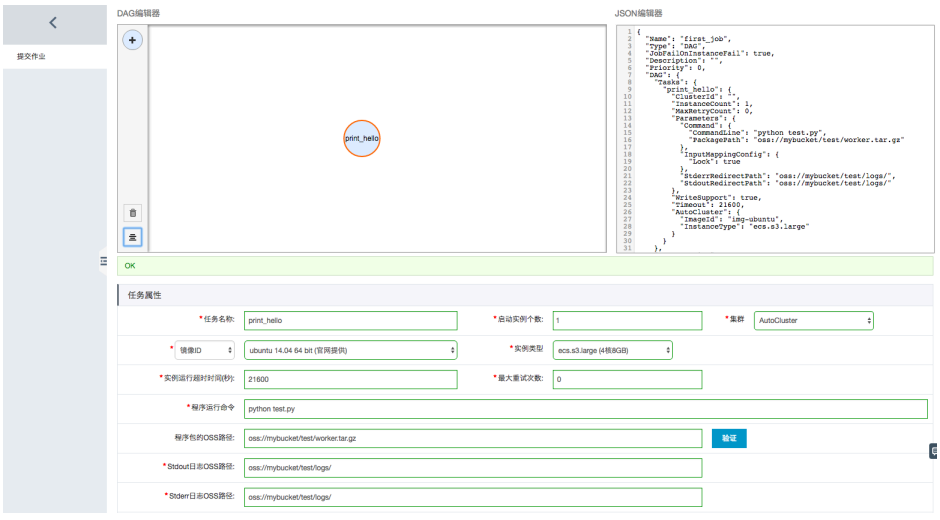
ii. 使用控制台提交作业

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。

批量删除作业

1 背景

由于历史原因，提交了大量的批量计算 DAG 作业，但是作业没有及时清理；或者单次提交了大量的作业，需要做清理操作。目前批量计算控制台支持单页批量删除，但若存在几十页作业的情况，清理动作也比较麻烦。因此提供了一个批量清理批量计算作业的工作。

2 工具

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import multiprocessing
import threading
import sys
PY2 = sys.version_info[0] == 2
if PY2:
import Queue as bcQueue
else:
import queue as bcQueue
import json
import time
from batchcompute import Client, ClientError
import argparse
from functools import wraps

def retryWrapper(func):
@wraps(func)
def wrapper(*args,**kwargs):
index = 0
while True:
try:
res = func(*args,**kwargs)
break
except ClientError as e:
status = e.get_status_code()
if status >= 400 and status < 500:
```

```
raise Exception(str(e))
if status >= 500 and status < 600:
if index > 6:
raise Exception(str(e))
else:
time.sleep(0.5 * pow(2,index))
index += 1
except Exception as e:
if index > 6:
raise Exception(str(e))
else:
time.sleep(0.5 * pow(2,index))
index += 1
return res
return wrapper

class ShowProcess():
"""
显示处理进度的类
调用该类相关函数即可实现处理进度的显示
"""
i = 0 # 当前的处理进度
max_steps = 0 # 总共需要处理的次数
max_arrow = 50 #进度条的长度
infoDone = 'done'

# 初始化函数，需要知道总共的处理次数
def __init__(self, max_steps, infoDone = 'Done'):
self.max_steps = max_steps
self.i = 0
self.infoDone = infoDone

# 显示函数，根据当前的处理进度i显示进度
# 效果为[>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>]100.00%
def show_process(self, i=None):
if i is not None:
self.i = i
else:
self.i += 1
num_arrow = int(self.i * self.max_arrow / self.max_steps) #计算显示多少个'>'
num_line = self.max_arrow - num_arrow #计算显示多少个'-'
percent = self.i * 100.0 / self.max_steps #计算完成进度，格式为xx.xx%
process_bar = '[' + '>' * num_arrow + '-' * num_line + ']'\
+ '%.2f' % percent + '%' + '\r' #带输出的字符串，'\r'表示不换行回到最左边
sys.stdout.write(process_bar) #这两句打印字符到终端
sys.stdout.flush()
if self.i >= self.max_steps:
self.close()

def close(self):
print('')
print(self.infoDone)
self.i = 0

class DeleteJob(threading.Thread):
def __init__(self, client, jobQueue, error_queue):
```

```
threading.Thread.__init__(self)
self.jobQueue = jobQueue
self.error_queue = error_queue
self.client = client

@retryWrapper
def delteSpecJob(self, jobId):
self.client.delete_job(jobId)

def run(self):
while True:
try:
jobId = self.jobQueue.get(block=False)
self.delteSpecJob(jobId)
self.jobQueue.task_done()
except bcQueue.Empty:
break
except Exception as e:
self.jobQueue.task_done()
self.error_queue.put('Delte Job(%s) exception %s' % (jobId, str(e)))

class DeleteJobs:
def __init__(self, client, days):
self.client = client
self.days = days

@retryWrapper
def listSpecIdxJob(self, marker, max_item):
response = self.client.list_jobs(marker, max_item)
return response

def listAndDeleteJobs(self):
marker = ""
max_item = 100
index = 0
jobQueue = bcQueue.Queue(0)
error_queue = bcQueue.Queue(0)

print("Begin List Spec Jobs...")
while marker or index == 0:
response = self.listSpecIdxJob(marker, max_item)
marker = response.NextMarker
for job in response.Items:
if job.State == "Running" or job.State == "Waiting":
continue
if self.days == 0:
jobQueue.put(job.Id)
else:
createTime = job.CreationTime.strftime("%Y-%m-%d %H:%M:%S.%f")
timestap = int(time.mktime(time.strptime(createTime, "%Y-%m-%d %H:%M:%S.%f")))
detaTime = self.days * 3600 * 24
if int(time.time()) - timestap >= detaTime:
jobQueue.put(job.Id)
# print "jobId: %s" % job.Id
index += 1
```

```
print("List Spec Jobs Finish...")
threadnum = multiprocessing.cpu_count()
size = jobQueue.qsize()
if size < threadnum:
threadnum = size

thread_pool = []
for threadnum in range(threadnum):
current = DeleteJob(self.client, jobQueue, error_queue)
thread_pool.append(current)
current.start()
print("Begin Delete Jobs...")
process_bar = ShowProcess(size, 'OK')
totalSize = size
while size != 0:
size = jobQueue.qsize()
process_bar.show_process(totalSize - size)
time.sleep(0.5)

jobQueue.join()
for thread in thread_pool:
thread.join()

errs = []
while True:
try:
error = error_queue.get(block=False)
errs.append(error)
except bcQueue.Empty:
break
return errs

if __name__ == "__main__":
parser = argparse.ArgumentParser(
formatter_class = argparse.ArgumentDefaultsHelpFormatter,
description = 'python scripyt for Delete Batchcompute jobs',
usage='deleteJobs.py <positional argument> [<args>]',
)
region_surpport = ["cn-beijing", "cn-zhangjiakou", "cn-hangzhou", "cn-huhehaote", "cn-shanghai", "cn-shenzhen"]

parser.add_argument('--region', action='store', type = str, choices=region_surpport, help = 'the region info.')
parser.add_argument('--id', action='store', type = str, required=True, help = 'keyid.')
parser.add_argument('--key', action='store', type = str, required=True, help = 'key secrete.')
parser.add_argument('--day', action='store', type=int, required=True, help = 'delete the jobs which creating time
before the spec day.')
args = parser.parse_args()

BatchHost = "batchcompute.%s.aliyuncs.com" % args.region
client = Client(BatchHost, args.id, args.key)

deleteJobs = DeleteJobs(client, args.day)
err = deleteJobs.listAndDeleteJobs()
if len(err) != 0 :
a = ""
print("\n".join(err))
else:
```

```
print("delete Jobs success!!!")
```

3 执行删除动作

3.1 准备工作

复制脚本到执行机器，执行机器 必须安装批量计算的SDK:

```
pip install batchcompute
```

若执行机器已经安装批量计算的 SDK ，则建议更新到最新版本

```
pip install --upgrade batchcompute
```

准备好阿里云 AK 信息，且该 AK 已经开通批量计算相关权限

执行机器安装好 Python，建议 Python 2.7及以上

3.2 执行脚本

```
python delete.py --region cn-beijing --id xxxx --key xxx --day 10
```

- id 表示 AK的 ID 信息
- key 表示 AK的 KEY 信息
- day 表示保留最近 N 天的意思，
- 示例表示：删除北京 region 10天前创建的非等待和运行状态的作业

3.3 执行结果

```
Begin List Spec Jobs...
List Spec Jobs Finihs...
[>>>>>>>>>----------------------------------------]19.47%
```

APP管理

前言

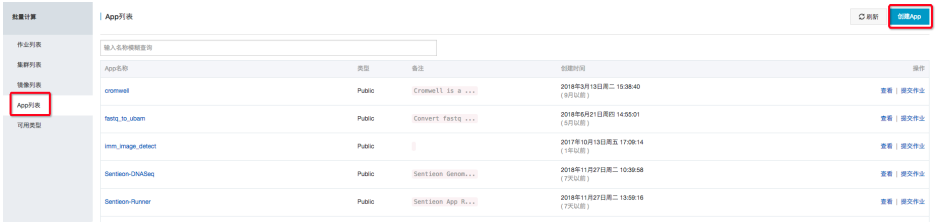
App 是什么

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。



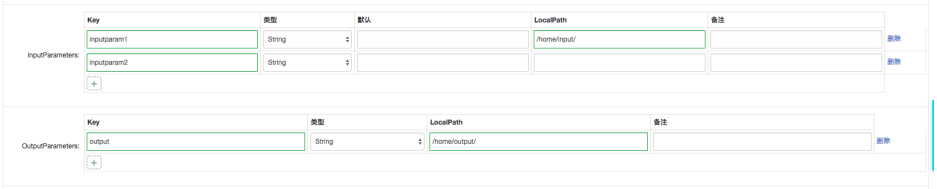
在批量计算控制台，通过 App 列表的创建 App 按钮进入创建页面，各参数的含义如下：

- 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。

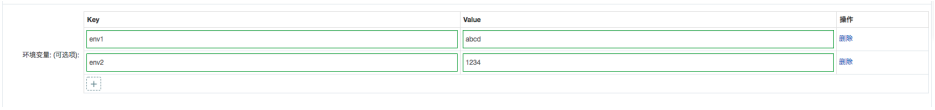
- InputParameters：App 的自定义输入参数。创建 App 时可以自定义一个或多个 String 或 Number 类型的入参。如果是来自 OSS 的输入数据，还可以通过指定 LocalPath 和本地路径做映射。
- OutputParameters：App 的自定义输出数据的路径。创建 App 时可以自定义一个或多个 String 或 Number 类型的输出参数。同样的，如果是来自 OSS 的路径，可以通过指定 LocalPath 和本地路径做映射。



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。



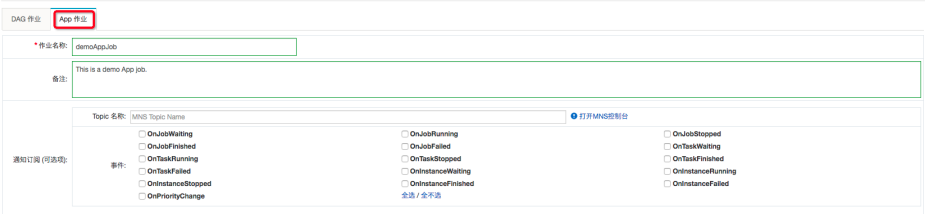
关于环境变量的使用，参考 环境变量 中的详细介绍。

如何提交App作业

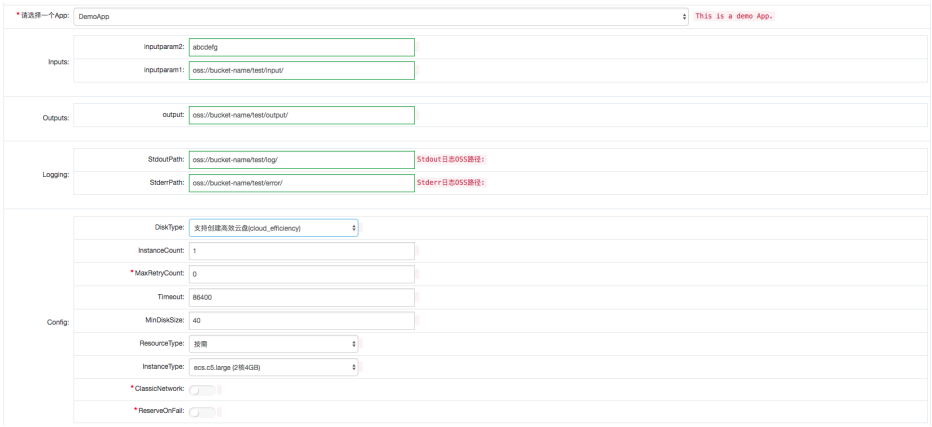
通过批量计算控制台作业列表的提交作业按钮进入作业提交页面。

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())
```

运维监控

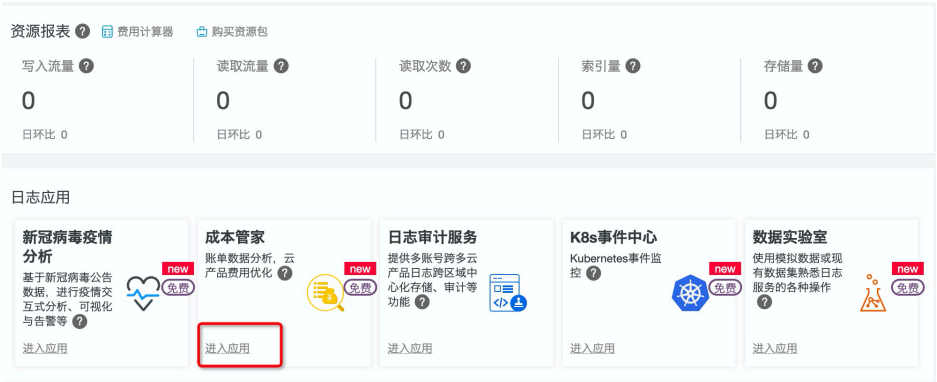
费用查询

1. 背景

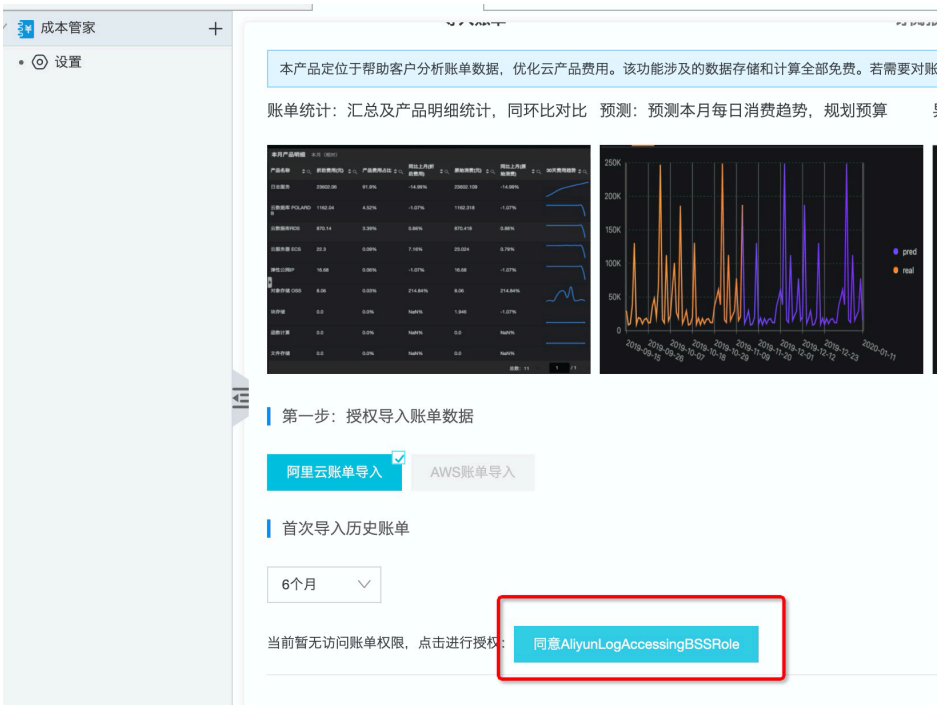
批量计算费用默认在**费用中心**可以查询，同时在费用中心可以看到各个云产品的费用情况；但是暂时无法看到单个云产品费用使用趋势，以及最近一周使用情况的分析。最近日志服务推出成本管家，通过成本管家可以自定义分析成本情况。目前批量计算已经将推出模板分析批量计算产生的费用信息。

2. 开通成本管家

登录到**日志服务控制台**；如下图



点击进入成本管家应用，授权 SLS 获取费用中心的账单信息



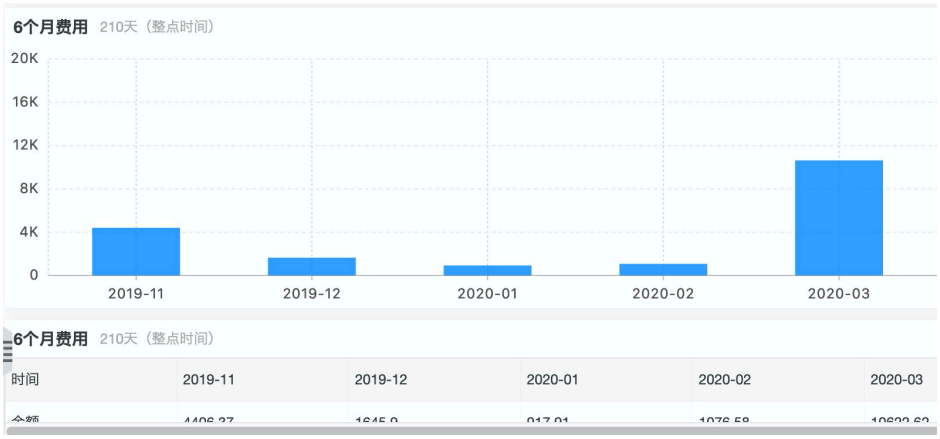
进入“下一步”，设置报告接收地址

设置完成即可以在左侧导航栏 “ECS”、“OSS”以及“批量计算”的费用信息

3. 批量计算费用统计介绍

- 一周费用统计情况

- 半年费用统计情况



- Top 100 固定集群费用(所有 region)

- Top 100 作业(AutoCluster 作业)费用(所有 region 作业)

- 各 region 费用情况分析

其中 费用类型 “cls”，“job” 分别表示固定集群费用 和 AutoCluster 作业费用

访问实例

批量计算固定集群申请的实例信息默认不支持公网访问能力，若希望能够登录到机器做操作做些操作可以通过以下途径，临时访问方式和生产访问形式；临时访问支持做些简单的操作、如问题debug等；若存在数据拷贝则无法通过临时方案，需要走生产方案。通过以上两种方式登录的实例一定是固定集群的申请实例，autocluster 申请的实例未对外开放登录的方式。

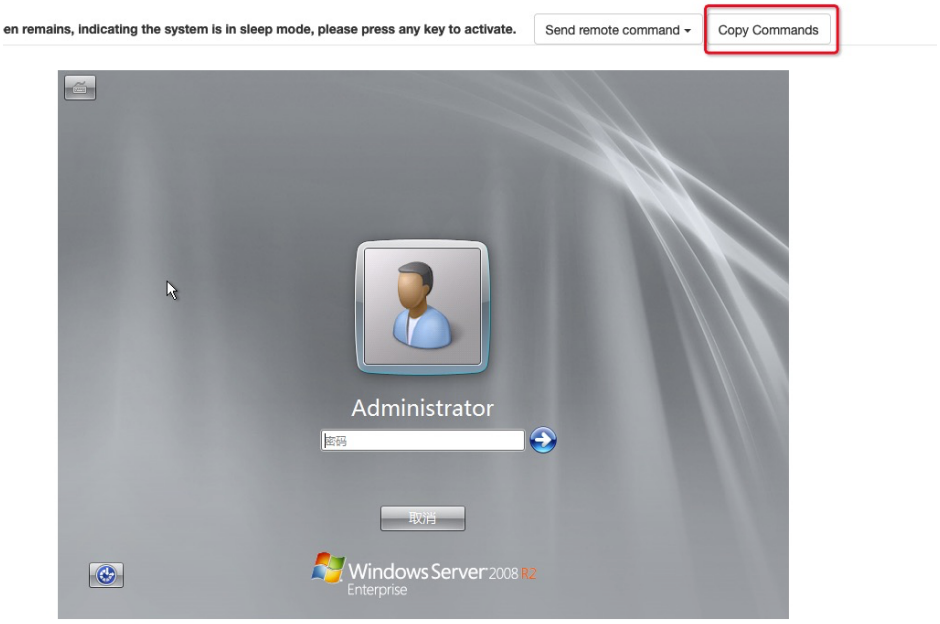
1. 临时访问

1.1. 登录批量计算控制台并找到对应的集群

1.2. 查看集群详细信息找到实例组

1.3. 查看实例列表

1.4. 在“密码”列获登录密码，点击进入远程登录



1.5. 登录成功后即可进行操作

2. 生产访问

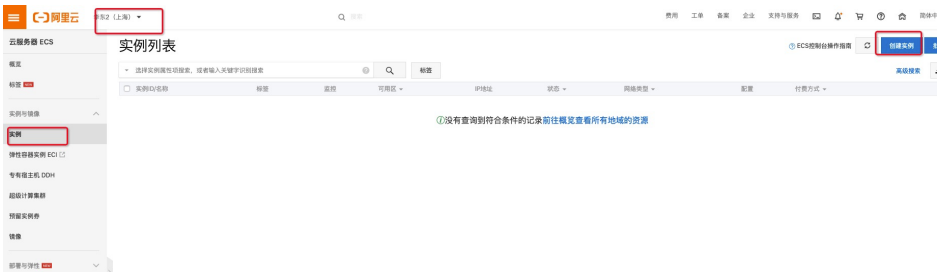
2.1. 登录 VPC 控制台

2.2. 创建 VPC 网络，若有请忽略

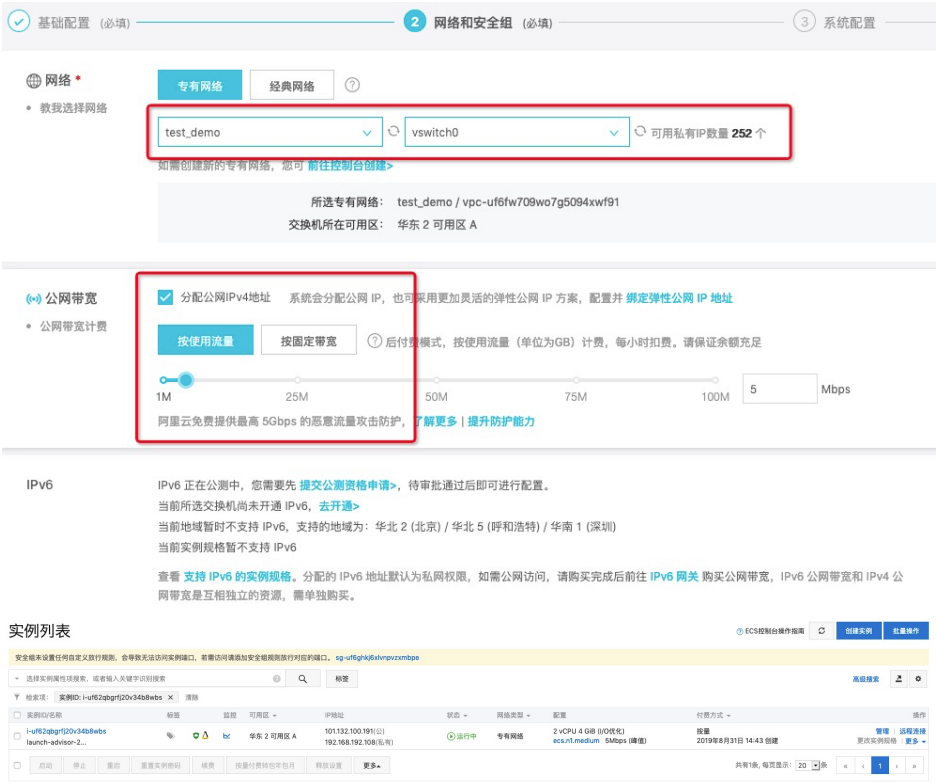
- 网段以及交换机规格根据业务需要进行划分。


创建专有网络
如何搭建专有网络
专有网络
地域
华东2（上海）
名称
test_demo
9/128
IPv4网段
推荐网段
高级配置网段
192.168.0.0/16
一旦创建成功，网段不能修改
描述
0/256
资源组
默认资源组
交换机
名称
vswitch0
8/128
可用区
上海 可用区A
可用区资源
ECS
RDS
SLB
IPv4网段
192
168
192
0
24
一旦创建成功，网段不能修改
可用IP数
确定
取消

2.3 创建 ECS 跳板机



- 跳板机镜像选择，若批量计算运行环境为 Windows 镜像，则此处也是 Windows 镜像；若批量计算运行环境为 Linux 或者 Centos 镜像，则此处可以是任意镜像；
- 跳板机网络必须在新创的 VPC 或者指定的 VPC 中；
- 跳板机公网 IP，公网带宽以及收费模式根据自身业务规划进行选择；
- 跳转机的安全组设置需要放开批量计算实例的网段以及涉及服务的端口；
- 为了生产的高可靠性，可以开多台机器。



2.4 创建批量计算集群

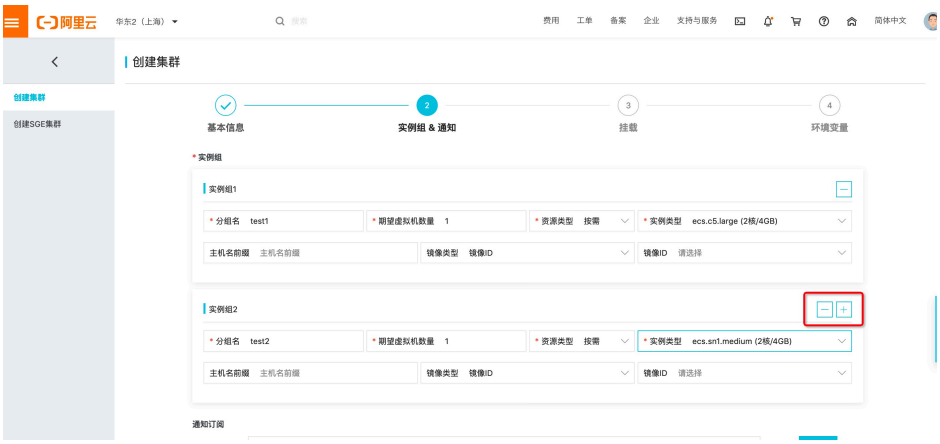
2.4.1 设置集群基本信息

- 设置集群名称；
- 选择集群镜像，支持官方镜像选择和自定义镜像；
- 设置启动脚本，Linux 系统支持 shell 和 Python；Windows镜像支持 Bat 和 Python；

- 启动脚本可以在镜像中，也可以通过 2.4.3 介绍的挂载方式挂载到虚拟机中；实例中启动脚本就是存放到 Oss 上，通过挂载的方式挂载到虚拟机的 “/root/test/“目录下；
- 启动脚本可以保证在节点内所有挂载操作成功之后再执行启动脚本；每个节点都是先挂载后执行启动脚本的顺序进行，节点间无顺序性的保证；根据机器的启动时间来进行，先启动的机器先执行以上动作，后启动的机器后执行以上动作；
- 启动脚本在集群内每个节点都会被执行。

2.4.2 实例组设置

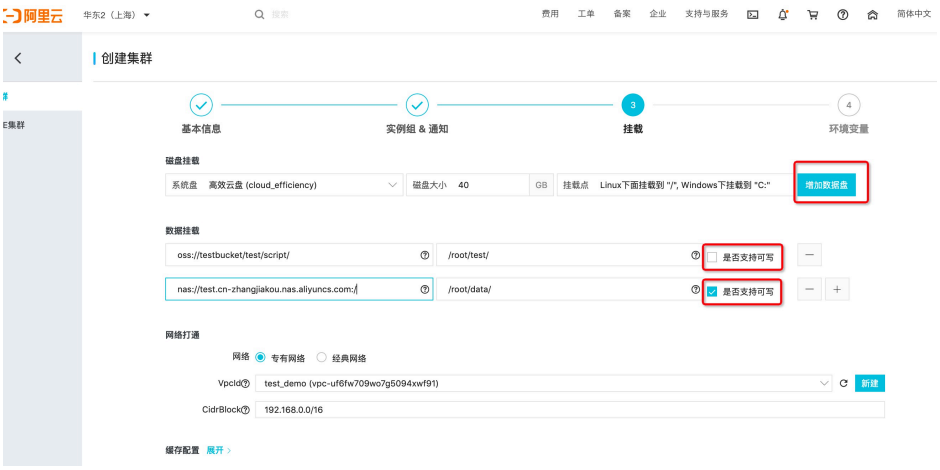
- 以组为单位进行设置资源类型、实例类型、实例数量、以及镜像相关信息；
- 组与组之间配置可以不同，建议生产环境上单个集群内可以设置多个组，实现实例类型多样化缓解资源利用高峰期的资源紧张问题；
- 通过调整实例数量实现集群规模的伸缩。



2.4.3 磁盘以及网络设置

- 设置系统盘类型以及大小(针对集群内每台机器)；
- 设置数据盘大小(针对集群内每台机器、类型必须和系统盘保持一致)，最大32TB；
- 设置 OSS 挂载点，参考示例填写方式；本地挂载点可以在镜像中不存在的路径，批量计算会主动创建；Windows系统本地挂载点必须是不存在的盘符，如不能是 C 盘；OSS 属于只读挂载，挂载成功后只能进行读操作暂不支持写操作；

- 设置 NAS 挂载，注意格式要求；本地挂载点要求和 OSS 的保持一致；NAS 挂载支持读写操作；
- 每个节点的挂载操作可以保证在启动脚本执行之前进行挂载；
- 若您在自定义镜像中设置开机启动你的程序，则无法保证您的程序启动时以上挂载一定完成，建议您的启动程序通过集群的启动脚本来完成而不是通过开机启动项来做配置；
- 网络设置，默认经典网络；若需要和 2.3 章节的中 ECS 保持通讯则此处必须和 ECS 在同一个 VPC 中；若同时需要挂载NAS，则 NAS 也必须在该 VPC 中，即批量计算、ECS、NAS 三者必须在同一个 VPC 中。



2.4.4 其他参数设置

- 其他参数根据需要进行设置；
- 所有参数设置完成提交创建；
- 以上步骤支持通过 API 形式创建集群，请参考实例文档。

2.5 批量计算节点访问

2.5.1 登录到跳板机

```
➜  ~ ssh root@101.132.100.191
The authenticity of host '101.132.100.191 (101.132.100.191)' can't be established.
ECDSA key fingerprint is SHA256:Vzgqq/OdXmqBIQYFBXKCeaXZWSE0kiEiqZPwZ25yZvY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '101.132.100.191' (ECDSA) to the list of known hosts.
root@101.132.100.191's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-105-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

Welcome to Alibaba Cloud Elastic Compute Service !

Last login: Sat May  5 16:16:44 2018 from 42.120.74.92
root@iZuf62qbgrfj20v34b8wbsZ:~#
root@iZuf62qbgrfj20v34b8wbsZ:~#
```

2.5.2 通过 步骤 1.1~1.4的方法获取批量计算实例的 IP 以及密码信息

2.5.3 登录到批量计算实例

```
root@iZuf62qbgrfj20v34b8wbsZ:~#
root@iZuf62qbgrfj20v34b8wbsZ:~#
root@iZuf62qbgrfj20v34b8wbsZ:~# ssh root@192.168.104.219
The authenticity of host '192.168.104.219 (192.168.104.219)' can't be established.
RSA key fingerprint is SHA256:zj5u3tk4loGFozmsimJnG0jqJoyO5B8zXwvZLKIa6CQ.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.104.219' (RSA) to the list of known hosts.
root@192.168.104.219's password:
Last login: Tue Oct 24 15:26:26 2017

Welcome to Alibaba Cloud Elastic Compute Service !

[root@iZuf6e0h1o563r63gttb37Z ~]#
```

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-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）
cn-hongkong	香港
cn-chengdu	西南1（成都）

国外部分

RegionId	地域
ap-southeast-1	新加坡
ap-southeast-2	澳大利亚（悉尼）
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）