ALIBABA CLOUD

阿里云

容器服务Kubernetes版
基因计算服务AGS用户指南

文档版本：20201109

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.AGS概览

阿里云基因计算AGS（Alibaba Cloud Genomics Service）是由阿里云推出极速、低成本、高精度的基因组测序二级分析的云服务，主要提供以容器平台为基础的生物信息工作流服务和无需搭建集群，开箱即用的加速API服务。本文介绍什么是AGS及其应用优势。

什么是AGS

AGS主要应用于基因组测序二级分析，通过AGS加速API只需要15分钟即可完成一个30X WGS的基因比对、排序、去重、变异检测全流程，相比经典流程可加速120倍，比目前全球最快的FPGA/GPU方案仍能提速2~4倍。

通过分析个体基因序列的突变机制，可为遗传病检测、肿瘤筛查等提供有力支撑，未来将在临床医学和基因诊断方面发挥巨大作用。人类全基因组有约30亿个碱基对，一个30X的WGS测序数据量大约在100GB。AGS在计算速度、精准度、成本、易用性、与上游测序仪的整合度上具有极大优势，同时适用于DNA的SNP/INDEL以及CNV结构变异检测，以及DNA/RNA病毒检测等场景。

更多信息请参见AGS产品详情页。

AGS的优势

- 极速、精准。

  经过实际测试，整套方案在15分钟内完成了8组30X WGS样本二级分析处理。在保证精度的前提下，实现15分钟对7200亿碱基拼装、排序、去重、变异检测，完成基因检测全流程120倍加速。且通过NA12878测试数据集与金标准VCF比较，二级分析的精度高于或等于BWA-0.7.17/GATK 4.1.3的数据产出，SNP精度到达99.80%。

数据集：30X NA12878			
SNP	**RECALL**	**PRECISION**	**F1**
GATK 4.1版	99.86%	99.79%	99.82%
AGS版	99.86%	99.80%	99.83%
INDEL	**RECALL**	**PRECISION**	**F1**
GATK 4.1版	99.28%	99.70%	99.49%
AGS版	99.27%	99.68%	99.47%

- 成本大幅优化。

  阿里云ACK/AGS提供云上PaaS加速能力，以混合云方式协助华大基因完成自主测序仪大批量下机数据二级分析。同时实现二级分析计算行业内低成本，缩短交付周期95%。

- 适用场景广。

  在保证分析通量的同时满足灵活性需求，可根据不同平台和数据定制分析流程。为各大测序服务商、研究机构等提供更简单更高效的存储、自动化分析、数据传输、项目协作以及生物信息工具开发等方面的解决方案。

AGS能够提供Kubernetes-native工作流机制，帮助用户在Kubernetes集群上运行支持DAG的工作流。在处理基因计算，数据计算等场景具有良好的通用性。

- 简单易用。

  AGS凭借云端的自动伸缩特性，实现大规模弹性调度计算。在使用上，该方案用户无需关心基因数据处理过程中的计算资源、处理逻辑、数据缓存等细节，只需将下机数据（FASTQ文件）上传至OSS，以及授权Bucket给AGS服务，即可高效、快速完成整个数据分析流程，并将结果数据上传到用户期望的存储空间。

AGS工作流

AGS工作流基于argo开发，可以为Kubernetes提供容器化的本地工作流程。工作流程中的每个步骤都定义为容器。

AGS工作流是作为Kubernetes CRD（自定义资源定义）实现的。 因此可以使用kubectl管理工作流，并与其他Kubernetes服务本地集成，例如Volumes、Secrets和RBAC。 工作流控制器提供完整的工作流程功能，包括参数替换、存储、循环和递归工作流程，详细介绍请参见创建工作流。

WDL工作流

WDL（Workflow Description Language）是由Broad Institute开发的一种流程开发语言，简单易用，能够有效的提高生物信息工作流的构建效率。详细介绍请参见创建WDL工作流。

在ACK上运行WDL的优势

- 兼容社区CronwellServer，完整兼容WDL的流程定义，对遗留流程无需修改，便可以通过AGS在ACK上运行WDL流程。WDL的详细介绍请参见WDL。
- 对Task资源申请优化，通过Pod Guarantee QoS方式，避免资源过度争取造成节点负载过高和效率下降。
- 与阿里云存储的无缝整合，目前支持直接访问OSS和NAS，并支持多数据源的挂载。

WDL局限性

- CronwellServer在以下方面仍然落后于云原生的AGS工作流。
  - 资源控制粒度（CPU、Mem min、Mem max）方面。
  - 调度优化，自动重试，资源上限的动态调整。
  - 监控、日志等方面。
- 在集群资源使用水位上低于AGS，在批量样本投递成功率上也低于AGS工作流。对于大批量重复性的工作流仍然建议改造成原生的AGS工作流来提升效率，详细介绍请参见创建工作流。
- 对于高CPU消耗的Mapping、HC、Mutecv2等流程可以使用AGS API来降低处理成本节省和加速，详细介绍请参见通过AGS处理全基因组测序WGS。

相关文档

除了上述所提及特点，AGS产品还成功解决了工作流程组装管理，海量数据存储、迁移与传输、安全合规等行业痛点问题。详情请参见以下文档。

- 通过AGS处理全基因组测序WGS
- 通过AGS排查病毒序列
- 通过AGS分析肿瘤样本
- 工作流在Kubernetes集群中的实践
- AGS命令行帮助

2.通过AGS进行群体多样本联合变异检测

对于一个家族或者相似的群体来说，变异往往具有一定的相似性。通过对一个家族或者群体进行联合变异的检测，可以有效提高变异检测的准确率。本文介绍如何通过命令行调用AGS服务API进行群体联合变异检测。

前提条件

开通服务。

背景信息

通过AGS全基因组的变异检测（WGS）生成具备更多未突变点位覆盖信息的GVCF，详情请参见GVCF。通过AGS的merge工具生成多样本的GVCF，详情请参见CombineGVCFs，为后续的变异矫正提供数据支撑。

AGS

准备工作

1. 下载和安装AGS，请参见AGS命令行帮助。
2. 配置AGS。

   ```
   ags config init
   ```

3. 确认Bucket的Owner。

   确保指定Bucket的Owner是您当前的账号，否则建议您新建一个Bucket，请参见创建存储空间。

   > **说明** 如果您使用的是子账号，需要为子账号授予AliyunOSSFullAccess权限，请参见为RAM用户授权。同时建议您重新创建OSS Bucket，以确保该账号是所使用Bucket的Owner。

   ```
   ossutil stat oss://<your new bucket name>
   ```

4. 准备Bucket，并授权AGS服务读写权限和GetBucketInfo权限。

   > **说明**
   >
   > - 请确保指定Bucket的Owner是您当前的账号，否则建议您新建一个Bucket后再进行GetBucketInfo的授权。
   > - 如果您使用的是子账号，需要为子账号授予AliyunOSSFullAccess权限，请参见为RAM用户授权。
   > - 如果您使用的是子账号，建议您重新创建OSS Bucket，以确保该账号是所使用Bucket的Owner。执行以下命令确认Bucket的Owner。
   >
   >   ```
   >   ossutil stat oss://<your new bucket name>
   >   ```

```
Usage:
ags config oss <your bucket name>

e.g.
ags config oss my-test-shenzhen
```

5. 通过ossutil上传FASTQ或者GVCF数据到OSS Bucket。

   关于ossutil的下载和安装，请参见下载和安装ossutil。

   > **说明** 目前只支持人类基因数据WGS、WES等，暂不支持甲基化数据，以及动植物数据的比对。

```
Usage:
ossutil cp -r <local dir of fastq> <path of  oss bucket >
e. g.
ossutil cp -r ./MGISEQ oss://my-test-shenzhen/MGISEQ
```

启动群体联合变异检测

1. 生成GVCF文件。

   关于参考基因组--reference，推荐和默认使用 `hg19` 基因组（hs37d5版本）。并且各个样本 `--reference-group` 中的SampleName（SM）须为不同，如以下所示，*MGISEQ_NA12878_hs37d5_6.gvcf*采用的SM为 *12878*，*MGISEQ_NA12878_hs37d5_5.gvcf*采用的SM为 *12878_1*。

   > **说明** 如果您已经有生成的GVCF文件，可以跳过该步骤，直接进行群体变异合并检测。

```
Usage:
ags remote run wgs \
--region cn-shenzhen # region of oss, e.g. cn-shenzhen, cn-beijing and etc\
--fastq1 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz # filename of fastq pair 2, fastq-path\filename \
--fastq2 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz  # filename of fastq pair 1\
--bucket my-test-shenzhen # Bucket name\
--output-bam bam/MGISEQ_NA12878_hs37d5.bam, # Output BAM to bucket,  By default empty, non output of BAM \
--output-gvcf gvcf/MGISEQ_NA12878_hs37d5_5.gvcf # Output filename \
--service "s" #SLA: [n:normal|s:silver|g:gold|p:platinum]\
--reference [hg19|hg38|<reference path on OSS>] # hg19: it is hs37d5 version, GRCh37/hg19 include decoy contig, no support for UCSC hg19. hg38: GRCh38/hg38 include decoy
--reference-group "\"@RG\\tID:TEST\\tSM:12878\\tPL:MGISEQ2000\"" # allow to specify reference groups for PL/SM/ID and etc

e.g.

ags remote run wgs \
--region cn-shenzhen \
--fastq1 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz  \
--fastq2 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz  \
--bucket my-test-shenzhen \
--output-gvcf gvcf/MGISEQ_NA12878_hs37d5_5.gvcf \
--output-bam bam/MGISEQ_NA12878_hs37d5_5.bam \
--service "s" \
--reference hg19 \
--reference-group "\"@RG\\tID:TEST\\tSM:12878_1\\tPL:MGISEQ2000\""

ags remote run wgs \
--region cn-shenzhen \
--fastq1 MGISAMPLE001 \
--fastq2 MGISAMPLE001  \
--bucket my-test-shenzhen \
--output-gvcf gvcf/MGISEQ_NA12878_hs37d5_6.gvcf \
--output-bam bam/MGISEQ_NA12878_hs37d5_6.bam \
--service "s" \
--reference hg19 \
--reference-group "\"@RG\\tID:TEST\\tSM:12878\\tPL:MGISEQ2000\""
```

2. 启动群体变异合并检测。

   通过merge调用生产多样本GVCF的合并GVCF *cohort.gvcf.gz*，相当于*GATK CombineGVCFs*结果。而联合基因型分析结果*output.vcf.gz*，相当于*GATK GenotypeGVCFs*的结果。

```
Usage:
ags remote merge \
--region cn-shenzhen # region of oss, e.g. cn-shenzhen, cn-beijing and etc\
--bucket my-test-shenzhen # Bucket name\
--gvcf gvcf/MGISEQ_NA12878_hs37d5_5.gvcf # filename of gvcf 1\
--gvcf gvcf/MGISEQ_NA12878_hs37d5_6.gvcf # filename of gvcf 2\
--gvcf gvcf/MGISEQ_NA12878_hs37d5_7.gvcf # filename of gvcf 3\
--output-vcf merge/output.vcf.gz # Output filename of vcf, it is simiar to result of GATK GenotypeGVCFs
\
--output-gvcf merge/cohort.gvcf.gz # Output filename of gvcf, it is similar to result of GATK CombineGV
CFs\
--service "s" #SLA: [s:silver|g:gold|p:platinum]\

e.g.
ags remote merge \
--region cn-shenzhen \
--bucket my-test-shenzhen \
--gvcf gvcf/MGISEQ_NA12878_hs37d5_5.gvcf \
--gvcf gvcf/MGISEQ_NA12878_hs37d5_6.gvcf \
--output-vcf merge/output.vcf.gz \
--output-gvcf merge/output.gvcf.gz \
--service "s"
INFO[0001] {"JobName":"merge-hck8x"}
INFO[0001] Job submit succeed
```

3. 列出远程任务。

```
Usage:
ags remote list
e.g.
ags remtoe list
```

```
+---------------+------------------------------+------------+
|  JOB NAME   |       CREATE TIME        | JOB STATUS |
+---------------+------------------------------+------------+
| merge-hck8x  | 2020-08-31 20:57:06 +0000 UTC | Running   |
| merge-5c4lt  | 2020-08-31 18:01:38 +0000 UTC | Succeeded  |
+---------------+------------------------------+------------+
```

2.1. 通过AGS处理全基因组测序WGS

通过AGS可以快速处理全基因组测序WGS（Whole Genome Sequencing）的全流程任务，包括基因比对、排序、去重和变异检测。本文介绍如何通过AGS命令行管理WGS工作流。

前提条件

开通服务。

准备工作

完成权限的配置和数据的准备。

1. 配置AGS。

   关于AGS的下载和安装，请参见AGS命令行帮助。

   ```
   ags config init
   ```

2. 准备Bucket，并授权AGS服务读写权限和GetBucketInfo权限。

   > **说明**
   > - 请确保指定Bucket的Owner是您当前的账号，否则建议您新建一个Bucket后再进行GetBucketInfo的授权。
   > - 如果您使用的是子账号，需要为子账号授予AliyunOSSFullAccess权限，请参见为RAM用户授权。
   > - 如果您使用的是子账号，建议您重新创建OSS Bucket，以确保该账号是所使用Bucket的Owner。执行以下命令确认Bucket的Owner。
   >
   >   ```
   >   ossutil stat oss://<your new bucket name>
   >   ```

   ```
   Usage:
   ags config oss <your bucket name>

   e.g.
   ags config oss my-test-shenzhen
   ```

3. 通过ossutil上传FASTQ数据到OSS Bucket。

   关于ossutil的下载和安装，请参见下载和安装。

   > **说明** 目前只支持人类基因数据 WGS、WES等，暂不支持甲基化数据，以及动植物数据的比对。

```
Usage:

ossutil cp -r <local dir of fastq> <path of  oss bucket >

e. g.
ossutil cp -r ./MGISEQ oss://my-test-shenzhen/MGISEQ
```

启动WGS流程

关于参考基因组--reference，推荐和默认使用 `hg19` 基因组（hs37d5版本）。

> **说明** `hg19` 基因组（hs37d5版本）主要有以下特点：
>
> - 不包含ALT contigs
> - Hard mask了chrY上的PARs区域
> - 包含decoy contig
>
> 虽然AGS是ALT-Aware，可以识别并处理ALT contigs，而 `UCSC hg19` 基因组包含了ALT contigs，但是由于 `UCSC hg19` 基因组不具备后面两个特点，仍会造成变异检测的质量下降。详情参见此处。

```
Usage:

ags remote run wgs \
--region cn-shenzhen # region of oss, e.g. cn-shenzhen, cn-beijing and etc\
--fastq1 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz # filename of fastq pair 2, fast
q-path\filename \
--fastq2 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz  # filename of fastq pair 1\
--bucket my-test-shenzhen # Bucket name\
--output-bam bam/MGISEQ_NA12878_hs37d5.bam, # Output BAM to bucket,  By default empty, non output
of BAM \
--output-vcf vcf/MGISEQ_NA12878_hs37d5_5.vcf # Output filename \
--service "g" #SLA: [n:normal|s:silver|g:gold|p:platinum]\
--reference [hg19|hg38|<reference path on OSS>] # hg19: it is hs37d5 version, GRCh37/hg19 include decoy co
ntig, no support for UCSC hg19. hg38: GRCh38/hg38 include decoy
--reference-group "\"@RG\\tID:TEST\\tSM:12878\\tPL:MGISEQ2000\"" # allow to specify reference groups for
PL/SM/ID and etc
e.g.
ags remote run wgs \
--region cn-shenzhen \
--fastq1 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz \
--fastq2 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz \
--bucket my-test-shenzhen \
```

```
--output-vcf vcf/MGISEQ_NA12878_hs37d5_5.vcf \
--output-bam bam/MGISEQ_NA12878_hs37d5_5.bam \
--service "s" \
--reference hg19

### 批量多Lane多样本的下机样本的处理
MGISAMPLE001是一组WGS多Lane测序样本，可以通过指定样本目录--fastq1 MGISAMPLE001，--fastq2 MGISAMP
LE001 实现对多Lane测序结果的合并和计算。
oss://my-test-shenzhen/MGISAMPLE001/L1/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz
oss://my-test-shenzhen/MGISAMPLE001/L2/MGISEQ2000_PCR-free_NA12878_1_V100003043_L02_1.fq.gz
oss://my-test-shenzhen/MGISAMPLE001/L1/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz
oss://my-test-shenzhen/MGISAMPLE001/L2/MGISEQ2000_PCR-free_NA12878_1_V100003043_L02_2.fq.gz

ags remote run wgs \
--region cn-shenzhen \
--fastq1 MGISAMPLE001 \
--fastq2 MGISAMPLE001  \
--bucket my-test-shenzhen \
--output-vcf vcf/MGISEQ_NA12878_hs37d5_6.vcf \
--output-bam bam/MGISEQ_NA12878_hs37d5_6.bam \
--service "g" \
--reference hg19

ags remote run wgs \
--region cn-shenzhen \
--fastq1 MGISAMPLE002 \
--fastq2 MGISAMPLE002  \
--bucket my-test-shenzhen \
--output-vcf vcf/MGISEQ_NA12878_hs37d5_7.vcf \
--output-bam bam/MGISEQ_NA12878_hs37d5_7.bam \
--service "g" \
--reference hg19
```

单击调用AGS WGS，将为您演示如何通过命令调用AGS WGS。

启动Mapping流程

通过--fastq1和--fastq2指定*fastq*，通过--output指定*bam*的输出路径。

```
Usage:

ags remote run mapping \
--region cn-shenzhen # region of oss, e.g. cn-shenzhen, cn-beijing and etc\
--fastq1 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz # filename of fastq pair 2, fast
q-path\filename \
--fastq2 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz  # filename of fastq pair 1\
--bucket my-test-shenzhen # Bucket name\
--output-bam bam/MGISEQ_NA12878_hs37d5.bam # Output filename of BAM \
--service "g" #SLA: [n:normal|s:silver|g:gold|p:platinum]\
--markdup [true|false|default true] #Mark Duplicated, by default true
--reference [hg19|hg38|<reference path on OSS>]

e.g.

ags remote run mapping \
--region cn-shenzhen \
--fastq1 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz  \
--fastq2 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz  \
--bucket my-test-shenzhen \
--output-bam bam/MGISEQ_NA12878_hs37d5.bam # Output filename of BAM \
--service "g" \
--markdup "true" \
--reference hg19
```

单击调用AGS Mapping，将为您演示如何通过命令调用AGS Mapping。

列出远程流程

```
Usage:
ags remote list

e.g.
ags remtoe list
+---------------+------------------------------+
|  JOB NAME   |       CREATE TIME       |
+---------------+------------------------------+
| wgs-gpu-ckw96 | 2020-01-07 19:08:32 +0000 UTC |
| wgs-gpu-djzws | 2020-01-07 18:31:22 +0000 UTC |
| wgs-gpu-pd659 | 2020-01-03 20:34:09 +0000 UTC |
+---------------+------------------------------+
```

获取流程的详细信息

```
Usage:
ags remote get <workflow id> --show
--show show detail of input parameters of workflow

e.g.
ags remote get wgs-gpu-sjtlw
+---------------+-----------------+-----------+----------------------------+----------+----------------------------+
|  JOB NAME   | JOB NAMESPACE  | STATUS  |      CREATE TIME       | DURATION |      FINISH TIME       |
+---------------+-----------------+-----------+----------------------------+----------+----------------------------+
| wgs-gpu-sjtlw | XXXXXXXXXXXXXXXX | Succeeded | 2020-01-07 21:38:05 +0800 CST | 12m25s   | 2020-01-07 21:50
:30 +0800 CST |
+---------------+-----------------+-----------+----------------------------+----------+----------------------------+

ags remote get wgs-gpu-97xfn --show

+---------------+-----------------+-----------+----------------------------+----------+----------------------------+
|  JOB NAME   | JOB NAMESPACE  | STATUS  |      CREATE TIME       | DURATION |      FINISH TIME       |
+---------------+-----------------+-----------+----------------------------+----------+----------------------------+
| wgs-gpu-sjtlw | XXXXXXXXXXXXXXXX | Succeeded | 2020-01-07 21:38:05 +0800 CST | 12m25s   | 2020-01-07 21:50
:30 +0800 CST |
+---------------+-----------------+-----------+----------------------------+----------+----------------------------+


+----------------------+-------------------------------+
|    JOB DETAIL     |                   |
+----------------------+-------------------------------+
| wgs_reference_file   | hg19                |
| wgs_service       | g                   |
| wgs_oss_region     | cn-shenzhen             |
| wgs_fastq_first_name | MGISAMPLE001            |
| wgs_fastq_second_name | MGISAMPLE001            |
| wgs_bucket_name     | my-test-shenzhen          |
| wgs_vcf_file_name   | vcf/MGISEQ_NA12878_hs37d5_6.vcf |
| wgs_bam_file_name   | bam/MGISEQ_NA12878_hs37d5_6.bam |
+----------------------+-------------------------------+
```

取消运行中的工作流

```
Usage:

ags remote cancel  <workflow id>

e.g.

ags remote cancel wgs-gpu-zls6r
INFO[0000] Successed to cancel wgs-gpu-zls6r
```

删除结束的工作流

您可以删除成功和失败的工作流，但不能删除运行中的工作流。

```
Usage:

ags remote remove <workflow id>

e.g.

ags remote remove wgs-gpu-zls6r
INFO[0000] Successed to remove wgs-gpu-zls6r
```

2.2. 通过AGS进行群体多样本联合变异检测

对于一个家族或者相似的群体来说，变异往往具有一定的相似性。通过对一个家族或者群体进行联合变异的检测，可以有效提高变异检测的准确率。本文介绍如何通过命令行调用AGS服务API进行群体联合变异检测。

前提条件

开通服务。

背景信息

通过AGS全基因组的变异检测（WGS）生成具备更多未突变点位覆盖信息的GVCF，详情请参见GVCF。通过AGS的merge工具生成多样本的GVCF，详情请参见CombineGVCFs，为后续的变异矫正提供数据支撑。

AGS

准备工作

1. 下载和安装AGS，请参见AGS命令行帮助。
2. 配置AGS。

   ```
   ags config init
   ```

3. 确认Bucket的Owner。

   确保指定Bucket的Owner是您当前的账号，否则建议您新建一个Bucket，请参见创建存储空间。

> **说明** 如果您使用的是子账号，需要为子账号授予AliyunOSSFullAccess权限，请参见为RAM用户授权。同时建议您重新创建OSS Bucket，以确保该账号是所使用Bucket的Owner。

```
ossutil stat oss://<your new bucket name>
```

4. 准备Bucket，并授权AGS服务读写权限和GetBucketInfo权限。

> **说明**
> - 请确保指定Bucket的Owner是您当前的账号，否则建议您新建一个Bucket后再进行GetBucketInfo的授权。
> - 如果您使用的是子账号，需要为子账号授予AliyunOSSFullAccess权限，请参见为RAM用户授权。
> - 如果您使用的是子账号，建议您重新创建OSS Bucket，以确保该账号是所使用Bucket的Owner。执行以下命令确认Bucket的Owner。
>
> ```
> ossutil stat oss://<your new bucket name>
> ```

```
Usage:
ags config oss <your bucket name>

e.g.
ags config oss my-test-shenzhen
```

5. 通过ossutil上传FASTQ或者GVCF数据到OSS Bucket。

   关于ossutil的下载和安装，请参见下载和安装ossutil。

> **说明** 目前只支持人类基因数据WGS、WES等，暂不支持甲基化数据，以及动植物数据的比对。

```
Usage:
ossutil cp -r <local dir of fastq> <path of  oss bucket >
e. g.
ossutil cp -r ./MGISEQ oss://my-test-shenzhen/MGISEQ
```

启动群体联合变异检测

1. 生成GVCF文件。

   关于参考基因组--reference，推荐和默认使用 `hg19` 基因组（hs37d5版本）。并且各个样本 `--reference-group` 中的SampleName（SM）须为不同，如以下所示，*MGISEQ_NA12878_hs37d5_6.gvcf*采用的SM为*12878*，*MGISEQ_NA12878_hs37d5_5.gvcf*采用的SM为*12878_1*。

> **说明** 如果您已经有生成的GVCF文件，可以跳过该步骤，直接进行群体变异合并检测。

```
Usage:
ags remote run wgs \
--region cn-shenzhen # region of oss, e.g. cn-shenzhen, cn-beijing and etc\
--fastq1 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz # filename of fastq pair 2, fastq-path\filename \
--fastq2 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz  # filename of fastq pair 1\
--bucket my-test-shenzhen # Bucket name\
--output-bam bam/MGISEQ_NA12878_hs37d5.bam, # Output BAM to bucket,  By default empty, non output of BAM \
--output-gvcf gvcf/MGISEQ_NA12878_hs37d5_5.gvcf # Output filename \
--service "s" #SLA: [n:normal|s:silver|g:gold|p:platinum]\
--reference [hg19|hg38|<reference path on OSS>] # hg19: it is hs37d5 version, GRCh37/hg19 include decoy contig, no support for UCSC hg19. hg38: GRCh38/hg38 include decoy
--reference-group "\"@RG\\tID:TEST\\tSM:12878\\tPL:MGISEQ2000\"" # allow to specify reference groups for PL/SM/ID and etc

e.g.

ags remote run wgs \
--region cn-shenzhen \
--fastq1 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz  \
--fastq2 MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz  \
--bucket my-test-shenzhen \
--output-gvcf gvcf/MGISEQ_NA12878_hs37d5_5.gvcf \
--output-bam bam/MGISEQ_NA12878_hs37d5_5.bam \
--service "s" \
--reference hg19 \
--reference-group "\"@RG\\tID:TEST\\tSM:12878_1\\tPL:MGISEQ2000\""

ags remote run wgs \
--region cn-shenzhen \
--fastq1 MGISAMPLE001 \
--fastq2 MGISAMPLE001  \
--bucket my-test-shenzhen \
--output-gvcf gvcf/MGISEQ_NA12878_hs37d5_6.gvcf \
--output-bam bam/MGISEQ_NA12878_hs37d5_6.bam \
--service "s" \
--reference hg19 \
--reference-group "\"@RG\\tID:TEST\\tSM:12878\\tPL:MGISEQ2000\""
```

2. 启动群体变异合并检测。

   通过merge调用生产多样本GVCF的合并GVCF *cohort.gvcf.gz*，相当于*GATK CombineGVCFs*结果。而联合基因型分析结果*output.vcf.gz*，相当于*GATK GenotypeGVCFs*的结果。

```
Usage:
ags remote merge \
--region cn-shenzhen # region of oss, e.g. cn-shenzhen, cn-beijing and etc\
--bucket my-test-shenzhen # Bucket name\
--gvcf gvcf/MGISEQ_NA12878_hs37d5_5.gvcf # filename of gvcf 1\
--gvcf gvcf/MGISEQ_NA12878_hs37d5_6.gvcf # filename of gvcf 2\
--gvcf gvcf/MGISEQ_NA12878_hs37d5_7.gvcf # filename of gvcf 3\
--output-vcf merge/output.vcf.gz # Output filename of vcf, it is simiar to result of GATK GenotypeGVCFs
\
--output-gvcf merge/cohort.gvcf.gz # Output filename of gvcf, it is similar to result of GATK CombineGV
CFs\
--service "s" #SLA: [s:silver|g:gold|p:platinum]\

e.g.
ags remote merge \
--region cn-shenzhen \
--bucket my-test-shenzhen \
--gvcf gvcf/MGISEQ_NA12878_hs37d5_5.gvcf \
--gvcf gvcf/MGISEQ_NA12878_hs37d5_6.gvcf \
--output-vcf merge/output.vcf.gz \
--output-gvcf merge/output.gvcf.gz \
--service "s"
INFO[0001] {"JobName":"merge-hck8x"}
INFO[0001] Job submit succeed
```

3. 列出远程任务。

```
Usage:
ags remote list
e.g.
ags remtoe list
```

```
+---------------+-------------------------------+------------+
|  JOB NAME     |       CREATE TIME             | JOB STATUS |
+---------------+-------------------------------+------------+
| merge-hck8x   | 2020-08-31 20:57:06 +0000 UTC | Running    |
| merge-5c4lt   | 2020-08-31 18:01:38 +0000 UTC | Succeeded  |
+---------------+-------------------------------+------------+
```

2.3. 通过AGS排查病毒序列

在病原体鉴定过程中，通过对比待测样本与已知病原体基因组数据库，可以高效准确的对待测样本进行鉴定分析。AGS服务提供了针对宏基因组测序数据的快速比对能力。本文介绍如何通过AGS排查待测病原体。

前提条件

开通服务。

背景信息

新冠肺炎发病后数周内都可以在上呼吸道或下呼吸道中检测到新型冠状病毒SARS-CoV-2 RNA（以下简称“新冠病毒”）。目前，已经有众多新冠病毒RT-PCR试剂盒可供选择，虽然PCR操作成本低、实效性高。但由于受病毒浓度和试剂盒质量等因素的影响，容易出现假阴性，需要多次检测确认，而且一次检测只能对一种特定病毒进行排查。

泛病原体检测的基因组下一代测序（Metagenomics Next-generation Sequencing, mNGS）技术直接从临床样本中随机抽取一定比例的核酸片段（包括大量人源核酸和少量微生物核酸）进行测序、数据库比对和生物信息分析，进而对病原微生物进行无偏性鉴定。该技术为新冠病毒SARS-CoV-2 RNA的早期发现和准确测序做出很大贡献。

相较于RT-PCR试剂盒检测，虽然mNGS方式检测分析周期相对较长，但准确率高，检测全面，可以一次排查多种病毒，同时也可监测病毒在传播过程中可能发生的变异，增强对病毒的防控。临床上，针对荧光定量PCR无法确诊的疑似患者进行二次检测可进一步提升检测结果的准确性，有效防止病毒变异产生的漏检。基于核酸序列比对的分析方式，一旦病原体的基因组已知，通过更新数据库，就可以实现高效准确检测病原体。

阿里云基因计算服务AGS提供了针对mNGS宏基因组测序数据的快速比对能力，对一组肺泡采样测序的宏基因组数据3.2Gbase（22M reads），60秒内可以完成和已知的病原体基因组包括新型冠状病毒SARS-CoV-2，39种BetaCov RNA，以及9334种已知病毒的参考序列的比对，并且支持自定义的病毒库的上传和比对。对于疾控中心，医院，实验室只需要一个阿里云的对象存储Bucket，以及命令行ags就可以完成整个的比对过程，并拿到高质量的匹配reads的数据和初步质量报告，为多种病原体检测，进一步的新冠病毒的蛋白质研究和变异研究提供了快捷准确的数据支撑。

欢迎基因测序厂商，疾控中心，医院，学校，制药企业申请使用。

准备工作

1. 配置AGS。AGS的下载和安装，请参见AGS命令行帮助。命令示例：

   ```
   ags config init
   ```

2. 下载安装ossutil命令行，详情请参见下载和安装。
3. 注册阿里云账号，开通对象存储OSS服务并创建存储空间（Bucket）用于存放mNGS测序数据（例如：oss://my-test-shenzhen）。详情请参见开始使用阿里云OSS。

   > **说明** 如果您使用的是子账号，建议您重新创建OSS Bucket，以确保该账号是所使用Bucket的Owner。执行以下命令确认Bucket的Owner。
   >
   > ```
   > ossutil stat oss://<your new bucket name>
   > ```

4. 为服务AGS服务授予Bucket的GetBucketInfo权限。

> 说明
> - 请确保指定Bucket的Owner是您当前的账号，否则建议您新建一个Bucket后再进行GetBucketInfo的授权。
> - 如果您使用的是子账号，需要为子账号授予AliyunOSSFullAccess权限，请参见为RAM用户授权。

命令示例：

```
ags config oss <bucket_name>
```

rna-mapping使用示例

单击检测AGS rna-mapping病毒，将为您演示如何通过命令检测AGS rna-mapping病毒。

与新冠病毒的比对

1. 运行ossutil命令，将mNGS测序数据上传至存储空间（Bucket）。命令示例：

```
ossutil cp  ICU6G_S2_L001_R1_001.fastq.gz oss://my-test-shenzhen/cov2-samples/
ossutil cp  ICU6G_S2_L001_R2_001.fastq.gz oss://my-test-shenzhen/cov2-samples/
```

2. 运行比对任务，对比mNGS数据和已知RNA序列数据。

   本文以比对ICU6G_S2_L001测序样本和新冠病毒的相似度为例，运行任务如下。

   命令格式：

```
Usage:
ags remote run rna-mapping \ # <rna-mapping>: RNA 序列的比对任务
--region <region_id> \ #
<cn-shenzhen|cn-beijing|...>: 地域ID，目前支持深圳和北京。
--bucket <bucket_name> \ # <bucket_name> 对象存储bucket的名称
--fastq1 <path_fq1> \ # 双端测序数据fq1相对路径
--fastq2 <path_fq2>\ # 双端测序数据fq2的相对路径
--output-bam <path_of_output_bam> \ #产出比对结果bam的输出路径，报告也在同样位置，以.txt结尾
--reference [sars-cov-2 | betacov-ncbi-39 | viral-9334 | <path of RNA library reference in specified bucket
>] # 参考序列预置了新型冠状病毒sars-cov-2和目前已经知道的39种betacov的冠状病毒，可以指定自定义的病毒
序列库
```

   命令示例：

```
ags remote run rna-mapping \
--region cn-shenzhen \
--fastq1 cov2-samples/ICU6G_S2_L001_R1_001.fastq.gz \
--fastq2 cov2-samples/ICU6G_S2_L001_R2_001.fastq.gz  \
--bucket my-test-shenzhen \
--output-bam bam/ICU6G_S2.bam  \
--reference sars-cov-2



INFO[0002] {"JobName":"rna-mapping-gpu-2ms6w"}
INFO[0002] Job submit succeed
```

3. 检查比对任务和比对结果。

   在这个比对任务示例中，10M reads和新冠病毒序列MN908947.3比对产生了3629个高质量重合的reads，并在新冠病毒特征区间排列分（AS）超过120分的序列（reads）条数有404个。说明可以精确的从此样本的测序数据中检测出SARS-CoV-2 RNA的序列。

   比对结果示例：

```
ags remote get rna-mapping-gpu-2ms6w --show
+----------------------+-----------------+-----------+-----------------------------+----------+-----------------------------+-----
--------+-------------+
|    JOB NAME     | JOB NAMESPACE  | STATUS  |      CREATE TIME       | DURATION |       FINISH TIME
| TOTAL READS | TOTAL BASES |
+----------------------+-----------------+-----------+-----------------------------+----------+-----------------------------+-----
--------+-------------+
| rna-mapping-gpu-2ms6w | XXXXXXXXXXXX | Succeeded | 2020-03-04 16:40:30 +0800 CST | 43s     | 2020-03-
04 16:41:13 +0800 CST |   10369818 |  1456539874 |
+----------------------+-----------------+-----------+-----------------------------+----------+-----------------------------+-----
--------+-------------+



+--------------------------------+-------------------------------------------+
|       JOB DETAIL       |                           |
+--------------------------------+-------------------------------------------+
| rna_matached_reads         |                      480 |
| rna_is_sars_cov2          | True                      |
| rna_mapping_oss_region       | cn-shenzhen                 |
| rna_mapping_fastq_second_name  | cov2-samples/ICU6G_S2_L001_R2_001.fastq.gz |
| rna_mapping_no_unmapped      |                           |
| rna_mapping_service        | s                         |
| rna_matached_reads_alignment   |                      404 |
| rna_high_quality_mapped      |                     3629 |
| rna_mapping_fastq_first_name   | cov2-samples/ICU6G_S2_L001_R1_001.fastq.gz |
| rna_mapping_mark_dup        |                           |
| rna_mapping_reference_file_name | sars-cov-2                |
| rna_cov_detail_file         | bam/ICU6G_S2.bam.cov.txt          |
| rna_mapping_bam_file_name     | bam/ICU6G_S2.bam              |
| rna_mapping_bucket_name      | my-test-shenzhen              |
+--------------------------------+-------------------------------------------+
```

4. 执行ossutil命令，下载对比数据和简单报告。

   命令示例：

```
ossutil ls oss://my-test-shenzhen/bam/ICU6G_S2.bam
LastModifiedTime            Size(B)  StorageClass   ETAG                    ObjectName
2020-03-04 16:41:11 +0800 CST    356320    Standard  9596D012A30438A0073A2A0B38F5D578     oss://my-test-shenzhen/bam/ICU6G_S2.bam
2020-03-04 16:41:11 +0800 CST      2889    Standard  63175E7180D110BA9D3BAB34F4313C59     oss://my-test-shenzhen/bam/ICU6G_S2.bam.cov.txt
2020-03-04 16:41:11 +0800 CST       396    Standard  940D51FF7ECFF60B5E5A41D1F635180D     oss://my-test-shenzhen/bam/ICU6G_S2.bam.summary.json

ossutil cp oss://my-test-shenzhen/bam/HKU2_160660.summary.json .
ossutil cp -r oss://my-test-shenzhen/bam/ICU6G_S2.bam.cov.txt .
ossutil cp oss://my-test-shenzhen/bam/HKU2_160660.bam .
```

报告示例：

```
cat bam/ICU6G_S2.bam.cov.txt

Summary:
High Quality Mapped Reads is: 3629
Matched reads in orf1ab range is: 480
Matched reads in orf1ab range with alignment score (AS) is greater than 120: 404
/data/cov2-samples_ICU6G_S2_L001_R1_001.fastq.gz-output/ICU6G_S2.bam is similar to SARS-CoV-2 with very high mappQ and AS reads: True
     21571    21581    21591    21601    21611    21621    21631
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCCCTGC
ATGTT GTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCCCTGC
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGT                 CAATTACCCCCTGC
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGA     CCCCCTGC
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCCCTGC
ATGTTTGTTTTTCTTGTTTTATTGCCA agtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
atgtttgtttttcttgttttattgcca      AGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCCCTGC
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCCCTGC
atgtttgtttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
atgtttgtttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
atgtttgtttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
```

```
atgtttgttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCC
CTGC
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCC
CTGC
atgtttgttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
atgtttgttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
atgtttgttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
atgtttgttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
atgtttgttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
atgtttgttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
atgtttgttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCC
CTGC
atgtttgttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCC
CTGC
ATGTTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCC
CTGC
atgtttgttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
 TGTTTGTTTTTCTTGTTTT    CACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCCCTGC
 tgtttgttttttcttgtttt
  TTTGTTTTTCTTGTTTTATTGCCACTAGTCTCTAGTCAGTGTGTTAATCTTACAACCAGAACTCAATTACCCCCT
GC
    gttttttcttgttttattgccactagtctctagtcagtgtgttaatcttacaaccagaactcaattaccccctgc
```

5. 进一步分析比对数据。您可以通过samtools stats，plot-bamstat 等工具将比对数据进一步分析coverage, depth等的相似度，还可以进一步实现蛋白质组成分析，以及变异分析。
6. 重复上述步骤可实现不同样本之间的比对。

与已知的39个betaCov病毒的比对

1. 运行ossutil命令，将mNGS测序数据上传至存储空间（Bucket）。命令示例：

```
ossutil cp  ICU6G_S2_L001_R1_001.fastq.gz oss://my-test-shenzhen/cov2-samples/
ossutil cp  ICU6G_S2_L001_R2_001.fastq.gz oss://my-test-shenzhen/cov2-samples/
```

2. 运行对比任务。

   命令示例：

```
ags remote run rna-mapping \
--region cn-shenzhen \
--fastq1 cov2-samples/ICU6G_S2_L001_R1_001.fastq.gz \
--fastq2 cov2-samples/ICU6G_S2_L001_R2_001.fastq.gz  \
--bucket my-test-shenzhen \
--output-bam bam/ICU6G_S2_virus.bam  \
--reference betacov-ncbi-39

INFO[0011] {"JobName":"rna-mapping-gpu-6mpcc"}
INFO[0011] Job submit succeed
```

3. 检查比对任务和比对结果。

   比对结果示例：

```
ags remote get rna-mapping-gpu-6mpcc --show
+----------------------+-----------------+-----------+-----------------------------+----------+-----------------------------+-----
--------+-------------+
|    JOB NAME     | JOB NAMESPACE  | STATUS  |      CREATE TIME       | DURATION |       FINISH TIME
| TOTAL READS | TOTAL BASES |
+----------------------+-----------------+-----------+-----------------------------+----------+-----------------------------+-----
--------+-------------+
| rna-mapping-gpu-6mpcc | XXXXXXXXX | Succeeded | 2020-03-04 17:36:21 +0800 CST | 40s     | 2020-03-04 1
7:37:01 +0800 CST |   10369818 |  1456539874 |
+----------------------+-----------------+-----------+-----------------------------+----------+-----------------------------+-----
--------+-------------+
# 2014 mapped reads detected, but no mapped reads found in range
+---------------------------------+-------------------------------------------+
|       JOB DETAIL        |                           |
+---------------------------------+-------------------------------------------+
| rna_mapping_reference_file_name | betacov-ncbi-39                 |
| rna_matached_reads_alignment   |                          0 |
| rna_mapping_bam_file_name     | bam/ICU6G_S2_virus.bam              |
| rna_mapping_fastq_first_name   | cov2-samples/ICU6G_S2_L001_R1_001.fastq.gz |
| rna_mapping_oss_region       | cn-shenzhen                    |
| rna_cov_detail_file         | bam/ICU6G_S2_virus.bam.cov.txt         |
| rna_mapping_no_unmapped      |                           |
| rna_matached_reads          |                         0 |
| rna_mapping_mark_dup        |                           |
| rna_mapping_service         | s                         |
| rna_high_quality_mapped      |                        2014 |
| rna_mapping_bucket_name      | my-test-shenzhen                |
| rna_mapping_fastq_second_name  | cov2-samples/ICU6G_S2_L001_R2_001.fastq.gz |
| rna_is_sars_cov2           | False                         |
+---------------------------------+-------------------------------------------+
```

与自定义病毒库的比对

1. 从NCBI GeneBank下载reference序列合并成为一个多contig的参考序列。

   示例：

   搜索核酸包含“betacov”的所有参考系列， 并下载参考系列。

   

2. 把下载的序列文件 `sequence.fa` 改名为 `betacov-ncbi-test.fa` 。
3. 上传reference到存储空间。命令示例：

```
ossutil cp betacov-ncbi-test.fa oss://my-test-shenzhen/ref/
```

4. 提交比对任务，指定reference的路径。命令示例：

```
ags remote run rna-mapping \
--region cn-shenzhen \
--fastq1 cov2-samples/ICU6G_S2_L001_R1_001.fastq.gz \
--fastq2 cov2-samples/ICU6G_S2_L001_R2_001.fastq.gz \
--bucket my-test-shenzhen \
--output-bam bam/ICU6G_S2_virus.bam \
--reference ref/betacov-ncbi-test.fa

INFO[0002] {"JobName":"rna-mapping-gpu-69mwb"}
INFO[0002] Job submit succeed
```

5. 查看比对报告和获取匹配的比对数据。

   对比结果示例：

```
ags remote get rna-mapping-gpu-69mwb --show
+----------------------+-----------------+-----------+-----------------------------+----------+-----------------------------+-----
--------+-------------+
|    JOB NAME     | JOB NAMESPACE  | STATUS  |       CREATE TIME       | DURATION |       FINISH TIME
| TOTAL READS | TOTAL BASES |
+----------------------+-----------------+-----------+-----------------------------+----------+-----------------------------+-----
--------+-------------+
| rna-mapping-gpu-69mwb | 1365606736606053 | Succeeded | 2020-03-04 17:47:00 +0800 CST | 40s     | 202
0-03-04 17:47:40 +0800 CST |   10369818 |  1456539874 |
+----------------------+-----------------+-----------+-----------------------------+----------+-----------------------------+-----
--------+-------------+



+--------------------------------+-------------------------------------------+
|       JOB DETAIL        |                           |
+--------------------------------+-------------------------------------------+
| rna_mapping_fastq_first_name   | cov2-samples/ICU6G_S2_L001_R1_001.fastq.gz |
| rna_mapping_fastq_second_name  | cov2-samples/ICU6G_S2_L001_R2_001.fastq.gz |
| rna_mapping_mark_dup        |                           |
| rna_mapping_oss_region       | cn-shenzhen                   |
| rna_cov_detail_file         | bam/ICU6G_S2_virus.bam.cov.txt        |
| rna_is_sars_cov2          | False                         |
| rna_mapping_bam_file_name     | bam/ICU6G_S2_virus.bam             |
| rna_mapping_service         | s                           |
| rna_matached_reads_alignment   |                          0 |
| rna_high_quality_mapped      |                       2014 |
| rna_mapping_bucket_name      | my-test-shenzhen                |
| rna_mapping_no_unmapped      |                           |
| rna_mapping_reference_file_name | ref/betacov-ncbi-test.fa            |
| rna_matached_reads          |                          0 |
+--------------------------------+-------------------------------------------+
+--------------------------------+-------------------------------------------+
```

6. 下载比对数据做进一步分析。命令示例：

```
ossutil ls oss://my-test-shenzhen/bam/ICU6G_S2_virus.bam
LastModifiedTime           Size(B)  StorageClass  ETAG                    ObjectName
2020-03-04 17:47:38 +0800 CST    753458    Standard  DF7B1A6CA5AF5DE6BF4FFDBB6DEF71C3    oss://
my-test-shenzhen/bam/ICU6G_S2_virus.bam
2020-03-04 17:47:38 +0800 CST      1474    Standard  9D7968A779A0DE7C1993CC2A8D0E5A56    oss://my
-test-shenzhen/bam/ICU6G_S2_virus.bam.cov.txt
2020-03-04 17:47:38 +0800 CST       397    Standard  81170E30BAAFEB947A2238E015171A51    oss://my-t
est-shenzhen/bam/ICU6G_S2_virus.bam.summary.json
Object Number is: 3

ossutil cp oss://my-test-shenzhen/bam/ICU6G_S2_virus.bam.summary.json .

cat bam/ICU6G_S2_virus.bam.summary.json
{
  "total_reads":10369818,
  "total_bases":1456539874,
  "pass_vendor_filter_reads":10369818,
  "mapped_reads":6736,
  "pair_reads":6680,
  "properly_paired_reads":6520,
  "mapq_40_to_inf_reads":2030,
  "mapq_30_to_40_reads":0,
  "mapq_20_to_30_reads":1,
  "mapq_10_to_20_reads":3,
  "mapq_0_to_10_reads":23,
  "mapq_0_reads":10367761,
  "GC":"46.499%",
  "total_alignment":2057,
  "supplementary_alignment":0
}%

ossutil cp oss://my-test-shenzhen/bam/ICU6G_S2_virus.bam .
samtools view bam/ICU6G_S2_virus.bam
```

2.4. 通过AGS分析肿瘤样本

通过AGS调用 `mutect2` 任务来检测体细胞短突变，短突变包括单核苷酸（SNV）以及插入和缺失（InDel）的突变。本文介绍如何通过AGS分析肿瘤样本。

背景信息

AGS `mutect2` 支持两种模式的典型场景：

- 肿瘤加正常样本模式：肿瘤样本在分析过程跳过正常人的胚系变异。
- 肿瘤模式：对单个肿瘤样本的比对数据进行分析。

`mutect2` 的体系变异检测是保持了和GATK4.1.3一致的变异检测方式，但提供了30~80倍的加速。针对90Gbase的比对数据，10分钟内可以完成变异检测。

mutect2使用示例

单击检测AGS Mutect2肿瘤，将为您演示如何通过命令检测AGS Mutect2肿瘤。

在肿瘤加正常样本模式下分析样本

以给定匹配的正常样本作为基准，`mutect2` 仅检测体细胞变异。`mutect2` 会根据提供的证据（例如在匹配的正常人中），实现跳过在胚系中明显存在的变异的逻辑，以避免在胚系事件上花费计算资源。

- 用法

  执行命令：

  ```
  ags remote run mutect2 \
  ```

  返回结果：

  ```
  --region cn-shenzhen # region of oss, e.g. cn-shenzhen, cn-beijing and etc\
  --bucket my-test-shenzhen # Bucket name\
  --input-bam-tumor bam/HKU2_160660.bam #Tumor sample bam file\
  --input-bam-normal  bam/MGISEQ_NA12878_RG_HG38.bam  # Optional normal sample bam \
  --bed bed/performance.blocks.exp.bed # Optional target bed \
  --output-vcf vcf/HKU2_160660.vcf  # Output filename\
  --service "s" #SLA: [n:normal|s:silver|g:gold|p:platinum]\
  --reference [hg19|hg38|<reference path on OSS>] # hg19: it is hs37d5 version, GRCh37/hg19 include decoy contig, no support for UCSC hg19. hg38: GRCh38/hg38 include decoy
  ```

- 结果示例：

```
--region cn-shenzhen \
--bucket my-test-shenzhen \
--input-bam-tumor bam/HKU2_160660.bam \
--input-bam-normal  bam/MGISEQ_NA12878_RG.bam \
--output-vcf vcf/HKU2_160660.vcf \
--service "s"  \
--reference hg19
INFO[0001] {"JobName":"mutect2-gpu-vp7d9"}
INFO[0001] Job submit succeed

ags remote get mutect2-gpu-vp7d9 --show
+------------------+------------------+---------+-------------------------------+---------------+-------------+-------------+
|   JOB NAME   | JOB NAMESPACE  | STATUS |      CREATE TIME      |  DURATION   | TOTAL READS | TOTAL BASES |
+------------------+------------------+---------+-------------------------------+---------------+-------------+-------------+
| mutect2-gpu-vp7d9 | XXXXXXXXX | Running | 2020-04-10 16:02:39 +0800 CST | 36.311883677s |       0 |       0 |
+------------------+------------------+---------+-------------------------------+---------------+-------------+-------------+



+-------------------------+-------------------------+
|     JOB DETAIL      |                         |
+-------------------------+-------------------------+
| mutect2_reference_group |                         |
| mutect2_oss_region      | cn-shenzhen             |
| mutect2_bucket_name     | my-test-shenzhen        |
| mutect2_output_vcf_name | vcf/HKU2_160660.vcf     |
| mutect2_reference_file  | hg19                    |
| mutect2_input_bam_tumor | bam/HKU2_160660.bam     |
| mutect2_input_bam_normal | bam/MGISEQ_NA12878_RG.bam |
| mutect2_input_bed       |                         |
| mutect2_service         | s                       |
+-------------------------+-------------------------+
```

在单独肿瘤样本模式下分析样本

此模式对单一类型的样本（例如肿瘤或正常样本）进行分析。

- 用法

  执行命令：

```
ags remote run mutect2 \
```

返回结果：

```
--region cn-shenzhen # region of oss, e.g. cn-shenzhen, cn-beijing and etc\
--bucket my-test-shenzhen # Bucket name\
--input-bam-tumor bam/HKU2_160660.bam #Tumor／Normal sample bam file\
--output-vcf vcf/HKU2_160660.vcf  # Output filename\
--service "s" #SLA: [n:normal|s:silver|g:gold|p:platinum]\
--reference [hg19|hg38|<reference path on OSS>] # hg19: it is hs37d5 version, GRCh37/hg19 include decoy contig, no support for UCSC hg19. hg38: GRCh38/hg38 include decoy
```

- 结果示例：

```
--region cn-shenzhen \
--bucket my-test-shenzhen \
--input-bam-tumor bam/HKU2_160660.bam \
--output-vcf vcf/HKU2_160660.all.vcf \
--service "s" \
--reference hg19
INFO[0001] {"JobName":"mutect2-gpu-6tc8s"}
INFO[0001] Job submit succeed

ags remote get mutect2-gpu-6tc8s --show
+------------------+-----------------+-----------+----------------------------+----------+----------------------------+-----------
-+-------------+
|    JOB NAME    | JOB NAMESPACE  | STATUS  |       CREATE TIME       | DURATION |       FINISH TIME       | T
OTAL READS | TOTAL BASES |
+------------------+-----------------+-----------+----------------------------+----------+----------------------------+-----------
-+-------------+
| mutect2-gpu-6tc8s | XXXXXXXXXX | Succeeded | 2020-04-10 15:51:59 +0800 CST | 4m12s    | 2020-04-10 15:56:
11 +0800 CST |        0 |        0 |
+------------------+-----------------+-----------+----------------------------+----------+----------------------------+-----------
-+-------------+



+-------------------------+-------------------------+
|      JOB DETAIL      |                         |
+-------------------------+-------------------------+
| mutect2_oss_region     | cn-shenzhen          |
| mutect2_input_bam_tumor  | bam/HKU2_160660.bam    |
| mutect2_input_bam_normal |                      |
| mutect2_input_bed      |                      |
| mutect2_output_vcf_name  | vcf/HKU2_160660.all.vcf |
| mutect2_bucket_name     | my-test-shenzhen      |
| mutect2_reference_file  | hg19                 |
| mutect2_reference_group |                      |
| mutect2_service        | s                    |
+-------------------------+-------------------------+
```

3.AGS无服务器化API参考

3.1. 通过AGS查询所有工作流

调用DescribeWorkflows查询已创建的所有工作流。

请求信息

请求行RequestLine

```
GET /gs/workflows HTTP/1.1
```

特有请求头RequestHead

无，请参见公共请求和返回结果。

请求体RequestBody

无

返回信息

返回行ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头ResponseHead

无，请参见公共请求和返回结果。

返回体ResponseBody

```
{
 "jobs": [
  {
   "create_time": "2020-01-15T14:13:16Z",
   "cluster_id": "cb1a7214cfc0b41d9bb086affc2d8f51c",
   "job_name": "mapping-gpu-mhhgh"
  },
  {
   "create_time": "2020-01-15T13:19:26Z",
   "cluster_id": "cb1a7214cfc0b41d9bb086affc2d8f51c",
   "job_name": "mapping-gpu-98wt4"
  },
  {
   "create_time": "2020-01-15T13:18:52Z",
   "cluster_id": "cb1a7214cfc0b41d9bb086affc2d8f51c",
   "job_name": "wgs-gpu-qb4dk"
  }
 ]
}
```


返回体解释

名称	类型	描述
cluster_id	String	集群ID。
job_name	String	工作流名称。
create_time	String	工作流创建时间。

示例

请求示例（Python）

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

from aliyunsdkcore.client import AcsClient
from aliyunsdkcore.request import CommonRequest
import os
client = AcsClient(os.environ['accessKeyID'], os.environ['accessKeySecret'], 'cn-beijing')

request = CommonRequest()
request.set_accept_format('json')
request.set_method('GET')
request.set_protocol_type('https') # https | http
request.set_domain('cs.cn-beijing.aliyuncs.com')
request.set_version('2015-12-15')

request.add_query_param('RegionId', "cn-beijing")
request.add_header('Content-Type', 'application/json')
request.set_uri_pattern('/gs/workflows')

response = client.do_action_with_exception(request)

print(response)
```

3.2. 通过AGS查询单个工作流

调用DescribeWorkflow查询单个工作流的详细信息。

请求信息

请求行RequestLine

```
GET /gs/workflow/{workflowName} HTTP/1.1
```

特有请求头RequestHead

无，请参见公共请求和返回结果。

请求体RequestBody

无

返回信息

返回行ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头ResponseHead

无，请参见公共请求和返回结果。

返回体ResponseBody


```
{
 "create_time": "2020-01-15 16:30:25 +0800 CST",
 "duration": "1h15m33.529968361s",
 "finish_time": "0001-01-01 00:00:00 +0000 UTC",
 "input_data_size": "0",
 "job_name": "wgs-gpu-97xfn",
 "job_namespace": "1171330362041663",
 "output_data_size": "0",
 "status": "Running",
 "total_bases": "0",
 "total_reads": "0",
 "user_input_data": "{\"wgs_oss_region\":\"cn-shenzhen\",\"wgs_fastq_first_name\":\"fastq/huada/MGISEQ-200019SZ0002402\",\"wgs_fastq_second_name\":\"fastq/huada/MGISEQ-200019SZ0002402\",\"wgs_bucket_name\":\"gene-shenzhen\",\"wgs_vcf_file_name\":\"output/vcf/huada.vcf\",\"wgs_bam_file_name\":\"output/bam/huada.bam\",\"wgs_reference_file\":\"hg19\",\"wgs_service\":\"g\"}"
}
```


返回体解释

名称	类型	描述
create_time	String	工作流创建时间。
duration	String	工作流经过时长。
finish_time	String	任务结束时间。
input_data_size	String	输入数据大小。
job_name	String	工作流名称。
job_namespace	String	工作流所在命名空间。
output_data_size	String	输出数据大小。
status	String	工作流当前状态。
total_bases	String	碱基对个数。

名称	类型	描述
total_reads	String	Reads个数。
user_input_data	String	用户输入参数。

示例

请求示例（Python）

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

from aliyunsdkcore.client import AcsClient
from aliyunsdkcore.request import CommonRequest
import os
client = AcsClient(os.environ['accessKeyID'], os.environ['accessKeySecret'], 'cn-beijing')

request = CommonRequest()
request.set_accept_format('json')
request.set_method('GET')
request.set_protocol_type('https') # https | http
request.set_domain('cs.cn-beijing.aliyuncs.com')
request.set_version('2015-12-15')

request.add_query_param('RegionId', "cn-beijing")
request.add_header('Content-Type', 'application/json')
request.set_uri_pattern('/gs/workflow/wgs-gpu-97xfn')

response = client.do_action_with_exception(request)

print(response)
```

3.3. 通过AGS创建基因工作流

调用StartWorkflow创建一个新的基因工作流。

请求信息

请求行RequestLine

```
POST /gs/workflow  HTTP/1.1
```

特有请求头RequestHead

无，请参见公共请求和返回结果。

请求体RequestBody

```
这里以mapping为例
{
    "workflow_type": "mapping",
    "service": "s" (#SLA: [n: normal|s: silver|g: gold|p: platinum])
    "mapping_oss_region": "cn-shenzhen",
    "mapping_fastq_first_filename": "MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz",
    "mapping_fastq_second_filename": "MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz",
    "mapping_bucket_name": "gene-shenzhen",
    "mapping_fastq_path": "fastq/MGISEQ2000",
    "mapping_reference_path": "reference/hg19", [Optional]
    "mapping_is_mark_dup": "true",
    "mapping_bam_out_path": "output/bamDirName",
    "mapping_bam_out_filename": "abc.bam"
}


这里以WGS为例
{
    "workflow_type": "wgs",
    "service": "s" (#SLA: [n: normal|s: silver|g: gold|p: platinum]),
    "wgs_oss_region": "cn-shenzhen",
    "wgs_fastq_first_filename": "MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz",
    "wgs_fastq_second_filename": "MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz",
    "wgs_bucket_name": "gene-shenzhen",
    "wgs_fastq_path": "fastq/MGISEQ2000",
    "wgs_reference_path": "reference/hg19",
    "wgs_vcf_out_path": "output/vcf",
    "wgs_vcf_out_filename": "abc.vcf"
}
```

请求体解释

名称	类型	描述
workflow_type	String	工作流类型，可选值：wgs或mapping。
service	String	SLA类型，可选值：n、s、g、p。

名称	类型	描述
mapping_oss_region	String	mapping oss数据的存放region。
mapping_fastq_first_filename	String	mapping的第一个fastq文件名。
mapping_fastq_second_filename	String	mapping的第二个fastq文件名。
mapping_bucket_name	String	存放mapping的bucket名称。
mapping_fastq_path	String	mapping的fastq文件路径。
mapping_reference_path	String	mapping的reference文件位置。
mapping_is_mark_dup	String	是否进行dup。
mapping_bam_out_path	String	bam文件输出路径。
mapping_bam_out_filename	String	bam文件输出名称。
wgs_oss_region	String	wgs oss数据的存放region。
wgs_fastq_first_filename	String	wgs的第一个fastq文件名。
wgs_fastq_second_filename	String	wgs的第二个fastq文件名。
wgs_bucket_name	String	存放wgs的bucket名称。
wgs_fastq_path	String	wgs的fastq文件路径。
wgs_reference_path	String	wgs的reference文件路径。
wgs_vcf_out_path	String	wgs的vcf输出路径。
wgs_vcf_out_filename	String	wgs的vcf输出文件名称。

返回信息

返回行ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头ResponseHead

无，请参见公共请求和返回结果。

返回体ResponseBody

```
这里以mapping为例
{
    JobName: mapping-gpu-66xv7
  }

这里以wgs为例
  {
    JobName: wgs-gpu-tvltf
  }
```

返回体解释

名称	类型	描述
JobName	String	工作流名称

示例

请求示例（Python）

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

from aliyunsdkcore.client import AcsClient
from aliyunsdkcore.request import CommonRequest
import os

os.environ.setdefault('DEBUG', 'sdk')

client = AcsClient(os.environ['accessKeyID'], os.environ['accessKeySecret'], 'cn-beijing')

request = CommonRequest()
request.set_accept_format('json')
request.set_method('POST')
request.set_protocol_type('https') # https | http
request.set_domain('cs.cn-beijing.aliyuncs.com')
request.set_version('2015-12-15')

request.add_query_param('RegionId', "cn-shenzhen")
request.add_header('Content-Type', 'application/json')
request.set_uri_pattern('/gs/workflow')
body = '''{"cli_version":"v1.0.1-882299b","wgs_bucket_name":"my-test-shenzhen","wgs_fastq_first_name"
:"MGISEQ/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz","wgs_fastq_second_name":"MGISE
Q/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz","wgs_oss_region":"cn-shenzhen","wgs_ref
erence_file":"hs37d5","wgs_service":"g","wgs_vcf_File_name":"vcf/MGISEQ_NA12878_hs37d5_13.vcf","wo
rkflow_type":"WGS"}'''
request.set_content(body.encode('utf-8'))
response = client.do_action_with_exception(request)
```

3.4. 通过AGS取消运行中的工作流

调用CancelWorkflow取消正在运行中的工作流。

请求信息

请求行RequestLine

```
PUT /gs/workflow/{workflowName}     HTTP/1.1
```

特有请求头RequestHead

无，请参见公共请求和返回结果。

请求体RequestBody

```
{
  "action": "cancel"
}
```

请求体解释

名称	类型	描述
action	String	执行的操作，目前只支持cancel。

返回信息

返回行ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头ResponseHead

无，请参见公共请求和返回结果。

返回体ResponseBody

无

示例

请求示例（Python）

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

from aliyunsdkcore.client import AcsClient
from aliyunsdkcore.request import CommonRequest
import os
client = AcsClient(os.environ['accessKeyID'], os.environ['accessKeySecret'], 'cn-beijing')

request = CommonRequest()
request.set_accept_format('json')
request.set_method('PUT')
request.set_protocol_type('https') # https | http
request.set_domain('cs.cn-beijing.aliyuncs.com')
request.set_version('2015-12-15')

request.add_query_param('RegionId', "cn-beijing")
request.add_header('Content-Type', 'application/json')
request.set_uri_pattern('/gs/workflow/wgs-gpu-97xfn')
body = '''{"action": "cancel"}'''
request.set_content(body.encode('utf-8'))

response = client.do_action_with_exception(request)

print(response)
```

3.5. 通过AGS删除工作流

调用RemoveWorkflow删除某个指定工作流。

请求信息

请求行RequestLine

```
DELETE /gs/workflow/{workflowName}  HTTP/1.1
```

特有请求头RequestHead

无，请参见公共请求和返回结果。

请求体RequestBody

无

返回信息

返回行ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头ResponseHead

无，请参见公共请求和返回结果。

返回体ResponseBody

无

示例

请求示例（Python）

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

from aliyunsdkcore.client import AcsClient
from aliyunsdkcore.request import CommonRequest
import os
client = AcsClient(os.environ['accessKeyID'], os.environ['accessKeySecret'], 'cn-beijing')

request = CommonRequest()
request.set_accept_format('json')
request.set_method('DELETE')
request.set_protocol_type('https') # https | http
request.set_domain('cs.cn-beijing.aliyuncs.com')
request.set_version('2015-12-15')

request.add_query_param('RegionId', "cn-beijing")
request.add_header('Content-Type', 'application/json')
request.set_uri_pattern('/gs/workflow/wgs-gpu-97xfn')


response = client.do_action_with_exception(request)

print(response)
```

4.AGS计费说明

本文介绍基因计算AGS的计费规则和SLA承诺，以便您了解计价规则，并更好地按需选型。

基因计算 AGS计费

AGS计费规则

根据不同的服务等级，SLA承诺和价格如下表所示。

服务等级	SLA承诺	价格	说明
白银级（s）	60分钟内，<=90 Gbp	1元/Gbp	超过90 Gbp的部分，按1.5 Gbp/min计算增加的时间。
黄金级（g）	45分钟内，<=90 Gbp	1.5元/Gbp	超过90 Gbp的部分，按2 Gbp/min计算增加的时间。
铂金级（p）	30分钟内，<=90 Gbp	2元/Gbp	超过90 Gbp的部分，按3 Gbp/min计算增加的时间。

> **说明** 1 Gbp = 10亿碱基对

计费示例

AGS根据实际发生的数据量进行计费。例如，针对白银级服务，在60分钟以内完成10 Gbp数据的处理，费用如下所示：

10 Gbp x 1元/Gbp = 10元

SLA承诺时间计算示例

针对白银级服务，单次120Gbp数据的承诺时间，计算如下所示：

(120 - 90) / 1.5 + 60 = 80 min

因此，120Gbp的数据不超过80分钟的处理时间都是满足服务承诺的。

> **说明** 超过承诺时间，不收取当次处理的费用。

5.AGS工作流

5.1. 创建工作流

工作流是基于argo开发，可以为Kubernetes提供容器化的本地工作流程。工作流程中的每个步骤都定义为容器。本文主要介绍通过控制台或命令行的方式创建工作流。

前提条件

- 创建Kubernetes托管版集群
- 通过kubectl连接Kubernetes集群

背景信息

工作流基于argo开发，可以为Kubernetes提供容器化的本地工作流程。工作流程中的每个步骤都定义为容器。

工作流是作为Kubernetes CRD（自定义资源定义）实现的。 因此可以使用kubectl管理工作流，并与其他Kubernetes服务本地集成，例如Volumes、Secrets和RBAC。 工作流控制器提供完整的工作流程功能，包括参数替换、存储、循环和递归工作流程。

您可以通过控制台或命令行的方式创建工作流。

通过控制台创建Hello World工作流

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中，单击**发布**。
5. 在**发布**页面，单击**工作流**，然后单击页面右上角的**创建**。
6. 对模板进行相关配置，完成配置后单击**创建**。
   - **集群**：选择目标集群。工作流将部署在该集群内。
   - **命名空间**：选择工作流所属的命名空间，默认是default。
   - **示例模板**：阿里云容器服务提供了多种资源类型的Kubernetes YAML示例模板，让您快速部署工作流。您可以根据Kubernetes YAML编排的格式要求自主编写，来描述您想定义的工作流。

   下面是一个Hello World工作流的示例编排，基于容器服务自定义的编排模板。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow              # new type of k8s spec
metadata:
 generateName: hello-world-   # name of the workflow spec
spec:
 entrypoint: whalesay        # invoke the whalesay template
 templates:
 - name: whalesay           # name of the template
  container:
    image: docker/whalesay
    command: [cowsay]
    args: ["hello world"]
    resources:            # limit the resources
     limits:
       memory: 32Mi
       cpu: 100m
```

7. 创建完成后，返回**工作流**页面，您可以看到已经创建成功的工作流。

   您可以单击**详情**，了解该工作流基本信息和器组信息。

通过命令行创建Parameters工作流

1. 创建并拷贝内容到*arguments-parameters.yaml*文件中。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: hello-world-parameters-
spec:
 # invoke the whalesay template with
 # "hello world" as the argument
 # to the message parameter
 entrypoint: whalesay
 arguments:
  parameters:
  - name: message
    value: hello world

 templates:
 - name: whalesay
  inputs:
    parameters:
    - name: message     # parameter declaration
  container:
    # run cowsay with that message input parameter as args
    image: docker/whalesay
    command: [cowsay]
    args: ["{{inputs.parameters.message}}"]
```

2. 执行命令，创建Parameters工作流。

```
ags submit arguments-parameters.yaml -p message="goodbye world"
```

您也可以参见Workflow示例模板，通过更新YAML文件的方式创建其他工作流。

Ags CLI是兼容阿里云定制的兼容社区版argo的命令行工具，使用Ags CLI可以方便的提交、查看、修改、删除工作流。关于更多内容请参见AGS命令行帮助。

5.2. Workflow示例模板

本文给您提供Workflow示例模板，便于您创建需要的工作流。

Steps

本示例中，我们将了解如何创建多步骤工作流，如何在工作流规范中定义多个模板，以及如何创建嵌套工作流。请务必阅读注释，以增强代码的可读性。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
```

```
 generateName: steps-
spec:
 entrypoint: hello-hello-hello

 # This spec contains two templates: hello-hello-hello and whalesay
 templates:
 - name: hello-hello-hello
  # Instead of just running a container
  # This template has a sequence of steps
  steps:
  - - name: hello1         # hello1 is run before the following steps
     template: whalesay
     arguments:
      parameters:
      - name: message
        value: "hello1"
  - - name: hello2a        # double dash => run after previous step
     template: whalesay
     arguments:
      parameters:
      - name: message
        value: "hello2a"
   - name: hello2b        # single dash => run in parallel with previous step
     template: whalesay
     arguments:
      parameters:
      - name: message
        value: "hello2b"

 # This is the same template as from the previous example
 - name: whalesay
  inputs:
   parameters:
   - name: message
  container:
   image: docker/whalesay
   command: [cowsay]
   args: ["{{inputs.parameters.message}}"]
```

Steps工作流模板打印出三种不同风格的hello。hello-hello-hello模板由三个步骤组成。名称为*hello1*的第一步将按顺序运行，而后两个名称为*hello2a*和*hello2b*的步骤将彼此并行运行。使用ags CLI命令，我们可以以图形方式显示此工作流规范的执行历史记录，该规则显示名为*hello2a*和*hello2b*的步骤彼此并行运行。

结果如下所示。

```
STEP                    PODNAME
 ✔ arguments-parameters-rbm92
 ├---✔ hello1           steps-rbm92-2023062412
 └- • -✔ hello2a          steps-rbm92-685171357
   └-✔ hello2b          steps-rbm92-634838500
```

DAG

作为指定步骤序列的替代方法，您可以通过指定每个任务的依赖关系将工作流定义为有向非循环图（DAG）。对于复杂的工作流程，这可以更简单地维护，并且在运行任务时允许最大的并行性。

在以下工作流程中，步骤A首先运行，因为它没有依赖项。A完成后，步骤B和C并行运行。最后，一旦B和C完成，步骤D就可以运行了。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: dag-diamond-
spec:
 entrypoint: diamond
 templates:
 - name: echo
  inputs:
   parameters:
   - name: message
  container:
   image: alpine:3.7
   command: [echo, "{{inputs.parameters.message}}"]
 - name: diamond
  dag:
   tasks:
   - name: A
    template: echo
    arguments:
     parameters: [{name: message, value: A}]
   - name: B
    dependencies: [A]
    template: echo
    arguments:
     parameters: [{name: message, value: B}]
   - name: C
    dependencies: [A]
    template: echo
    arguments:
     parameters: [{name: message, value: C}]
   - name: D
    dependencies: [B, C]
    template: echo
    arguments:
     parameters: [{name: message, value: D}]
```

依赖图可以有多个根。从DAG或步骤模板调用的模板本身可以是DAG或步骤模板。这可以允许将复杂的工作流程拆分为可管理的部分。

Secrets

Template支持与Kubernetes Pod规范相同的Secret语法和机制，允许访问Secret作为环境变量或Volume mounts。

```
# To run this example, first create the secret by running:
# kubectl create secret generic my-secret --from-literal=mypassword=S00perS3cretPa55word
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: secret-example-
spec:
 entrypoint: whalesay
 # To access secrets as files, add a volume entry in spec.volumes[] and
 # then in the container template spec, add a mount using volumeMounts.
 volumes:
 - name: my-secret-vol
  secret:
   secretName: my-secret    # name of an existing k8s secret
 templates:
 - name: whalesay
  container:
   image: alpine:3.7
   command: [sh, -c]
   args: ['
    echo "secret from env: $MYSECRETPASSWORD";
    echo "secret from file: `cat /secret/mountpath/mypassword`"
   ']
   # To access secrets as environment variables, use the k8s valueFrom and
   # secretKeyRef constructs.
   env:
   - name: MYSECRETPASSWORD  # name of env var
    valueFrom:
     secretKeyRef:
      name: my-secret    # name of an existing k8s secret
      key: mypassword    # 'key' subcomponent of the secret
   volumeMounts:
   - name: my-secret-vol    # mount file containing secret at /secret/mountpath
    mountPath: "/secret/mountpath"
```

Scripts & Results

通常，我们只需要一个模板来执行工作流规范中指定的脚本。此示例显示了如何执行此操作。

```
apiVersion: argoproj.io/v1alpha1
```

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: scripts-bash-
spec:
 entrypoint: bash-script-example
 templates:
 - name: bash-script-example
  steps:
  - - name: generate
     template: gen-random-int-bash
  - - name: print
     template: print-message
     arguments:
      parameters:
      - name: message
       value: "{{steps.generate.outputs.result}}"  # The result of the here-script

 - name: gen-random-int-bash
  script:
   image: debian:9.4
   command: [bash]
   source: |                          # Contents of the here-script
    cat /dev/urandom | od -N2 -An -i | awk -v f=1 -v r=100 '{printf "%i\n", f + r * $1 / 65536}'

 - name: gen-random-int-python
  script:
   image: python:alpine3.6
   command: [python]
   source: |
    import random
    i = random.randint(1, 100)
    print(i)

 - name: gen-random-int-javascript
  script:
   image: node:9.1-alpine
   command: [node]
   source: |
    var rand = Math.floor(Math.random() * 100);
    console.log(rand);
```

```
- name: print-message
 inputs:
  parameters:
  - name: message
 container:
  image: alpine:latest
  command: [sh, -c]
  args: ["echo result was: {{inputs.parameters.message}}"]
```

Script关键字允许使用source标记指定脚本体。这将创建一个包含脚本主体的临时文件，然后将临时文件的名称作为命令的最终参数传递，该命令应该是执行脚本主体的解释器。

脚本功能的使用还将运行脚本的标准输出分配给名为Result的特殊输出参数。这允许您在其余的工作流规范中使用运行脚本本身的结果。在此示例中，结果仅由打印消息模板回显。

Output Parameters

Output Parameters提供了将步骤的结果用作参数而不是使用外部存储的一般机制。这允许您使用任何类型的步骤的结果，而不仅仅是脚本，用于条件测试，循环和参数。Output Parameters与脚本结果的工作方式类似，只是Output Parameters的值设置为生成文件的内容而不是stdout的内容。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: output-parameter-
spec:
 entrypoint: output-parameter
 templates:
 - name: output-parameter
  steps:
  - - name: generate-parameter
     template: whalesay
  - - name: consume-parameter
     template: print-message
     arguments:
      parameters:
      # Pass the hello-param output from the generate-parameter step as the message input to print-messag
e
      - name: message
       value: "{{steps.generate-parameter.outputs.parameters.hello-param}}"

 - name: whalesay
  container:
   image: docker/whalesay:latest
   command: [sh, -c]
   args: ["echo -n hello world > /tmp/hello_world.txt"]  # generate the content of hello_world.txt
  outputs:
   parameters:
   - name: hello-param   # name of output parameter
    valueFrom:
     path: /tmp/hello_world.txt  # set the value of hello-param to the contents of this hello-world.txt

 - name: print-message
  inputs:
   parameters:
   - name: message
  container:
   image: docker/whalesay:latest
   command: [cowsay]
   args: ["{{inputs.parameters.message}}"]
```

DAG模板使用任务前缀来引用另一个任务，例如，`{{tasks.generate-parameter.outputs.parameters.hello-param}}`。

Loops

在编写Loops工作流时：

- 迭代一组输入（最为常用），如下例所示。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: loops-
spec:
 entrypoint: loop-example
 templates:
 - name: loop-example
  steps:
  - - name: print-message
     template: whalesay
     arguments:
      parameters:
      - name: message
       value: "{{item}}"
     withItems:          # invoke whalesay once for each item in parallel
     - hello world       # item 1
     - goodbye world     # item 2

 - name: whalesay
  inputs:
   parameters:
   - name: message
  container:
   image: docker/whalesay:latest
   command: [cowsay]
   args: ["{{inputs.parameters.message}}"]
```

- 迭代多组项目，如下例所示。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: loops-maps-
spec:
 entrypoint: loop-map-example
 templates:
 - name: loop-map-example
  steps:
  - - name: test-linux
     template: cat-os-release
     arguments:
      parameters:
      - name: image
       value: "{{item.image}}"
      - name: tag
       value: "{{item.tag}}"
     withItems:
     - { image: 'debian', tag: '9.1' }     #item set 1
     - { image: 'debian', tag: '8.9' }     #item set 2
     - { image: 'alpine', tag: '3.6' }     #item set 3
     - { image: 'ubuntu', tag: '17.10' }    #item set 4

 - name: cat-os-release
  inputs:
   parameters:
   - name: image
   - name: tag
  container:
   image: "{{inputs.parameters.image}}:{{inputs.parameters.tag}}"
   command: [cat]
   args: [/etc/os-release]
```

- 将项目列表作为参数传递，如下例所示。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: loops-param-arg-
spec:
 entrypoint: loop-param-arg-example
 arguments:
```

```
arguments:
 parameters:
 - name: os-list                    # a list of items
   value: |
     [
       { "image": "debian", "tag": "9.1" },
       { "image": "debian", "tag": "8.9" },
       { "image": "alpine", "tag": "3.6" },
       { "image": "ubuntu", "tag": "17.10" }
     ]

templates:
- name: loop-param-arg-example
 inputs:
   parameters:
   - name: os-list
 steps:
 - - name: test-linux
     template: cat-os-release
     arguments:
       parameters:
       - name: image
         value: "{{item.image}}"
       - name: tag
         value: "{{item.tag}}"
     withParam: "{{inputs.parameters.os-list}}"    # parameter specifies the list to iterate over

# This template is the same as in the previous example
- name: cat-os-release
 inputs:
   parameters:
   - name: image
   - name: tag
 container:
   image: "{{inputs.parameters.image}}:{{inputs.parameters.tag}}"
   command: [cat]
   args: [/etc/os-release]
```

- 动态生成要迭代的项目列表，如下例所示。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: loops-param-result-
spec:
 entrypoint: loop-param-result-example
 templates:
 - name: loop-param-result-example
  steps:
  - - name: generate
     template: gen-number-list
  # Iterate over the list of numbers generated by the generate step above
  - - name: sleep
     template: sleep-n-sec
     arguments:
      parameters:
      - name: seconds
        value: "{{item}}"
     withParam: "{{steps.generate.outputs.result}}"

 # Generate a list of numbers in JSON format
 - name: gen-number-list
  script:
   image: python:alpine3.6
   command: [python]
   source: |
    import json
    import sys
    json.dump([i for i in range(20, 31)], sys.stdout)

 - name: sleep-n-sec
  inputs:
   parameters:
   - name: seconds
  container:
   image: alpine:latest
   command: [sh, -c]
   args: ["echo sleeping for {{inputs.parameters.seconds}} seconds; sleep {{inputs.parameters.seconds}};
echo done"]
```

Conditionals

我们还支持条件执行，如下例所示。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: coinflip-
spec:
 entrypoint: coinflip
 templates:
 - name: coinflip
  steps:
  # flip a coin
  - - name: flip-coin
     template: flip-coin
  # evaluate the result in parallel
  - - name: heads
     template: heads             # call heads template if "heads"
     when: "{{steps.flip-coin.outputs.result}} == heads"
   - name: tails
     template: tails             # call tails template if "tails"
     when: "{{steps.flip-coin.outputs.result}} == tails"

 # Return heads or tails based on a random number
 - name: flip-coin
  script:
   image: python:alpine3.6
   command: [python]
   source: |
    import random
    result = "heads" if random.randint(0,1) == 0 else "tails"
    print(result)

 - name: heads
  container:
   image: alpine:3.6
   command: [sh, -c]
   args: ["echo \"it was heads\""]

 - name: tails
  container:
```

```
    image: alpine:3.6
    command: [sh, -c]
    args: ["echo \"it was tails\""]
```

Recursion

模板可以递归地相互调用。在上述硬币翻转模板的这种变体中，我们继续翻转硬币直到它出现在头部。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: coinflip-recursive-
spec:
 entrypoint: coinflip
 templates:
 - name: coinflip
  steps:
  # flip a coin
  - - name: flip-coin
     template: flip-coin
  # evaluate the result in parallel
  - - name: heads
     template: heads            # call heads template if "heads"
     when: "{{steps.flip-coin.outputs.result}} == heads"
    - name: tails               # keep flipping coins if "tails"
     template: coinflip
     when: "{{steps.flip-coin.outputs.result}} == tails"

 - name: flip-coin
  script:
   image: python:alpine3.6
   command: [python]
   source: |
    import random
    result = "heads" if random.randint(0,1) == 0 else "tails"
    print(result)

 - name: heads
  container:
   image: alpine:3.6
   command: [sh, -c]
   args: ["echo \"it was heads\""]
```

这是几次硬币翻转的结果，用于比较。

```
ags get coinflip-recursive-tzcb5

STEP                PODNAME                  MESSAGE
 ✔ coinflip-recursive-vhph5
  ├---✔ flip-coin          coinflip-recursive-vhph5-2123890397
  └- • -✔ heads            coinflip-recursive-vhph5-128690560
    └-◯ tails

STEP                PODNAME                  MESSAGE
 ✔ coinflip-recursive-tzcb5
  ├---✔ flip-coin          coinflip-recursive-tzcb5-322836820
  └- • -◯ heads
    └-✔ tails
      ├---✔ flip-coin      coinflip-recursive-tzcb5-1863890320
      └- • -◯ heads
        └-✔ tails
          ├---✔ flip-coin  coinflip-recursive-tzcb5-1768147140
          └- • -◯ heads
            └-✔ tails
              ├---✔ flip-coin  coinflip-recursive-tzcb5-4080411136
              └- • -✔ heads    coinflip-recursive-tzcb5-4080323273
                └-◯ tails
```

在第一次运行中，硬币立即出现在头部，我们停下来；在第二次运行中，硬币上升了三次，然后它终于出现了，我们就停了下来。

Exit handlers

Exit handlers是在工作流结束时始终执行的模板，无论成功或失败。

Exit handlers的一些常见用例。

- 工作流程运行后清理
- 发送工作流程状态通知（例如，电子邮件或Slack）
- 将通过或失败状态发布到webhook结果（例如GitHub构建结果）
- 重新提交或提交其他工作流程

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: exit-handlers-
spec:
 entrypoint: intentional-fail
 onExit: exit-handler              # invoke exit-hander template at end of the workflow
```

```
templates:
# primary workflow template
- name: intentional-fail
 container:
  image: alpine:latest
  command: [sh, -c]
  args: ["echo intentional failure; exit 1"]

# Exit handler templates
# After the completion of the entrypoint template, the status of the
# workflow is made available in the global variable {{workflow.status}}.
# {{workflow.status}} will be one of: Succeeded, Failed, Error
- name: exit-handler
 steps:
 - - name: notify
    template: send-email
  - name: celebrate
    template: celebrate
    when: "{{workflow.status}} == Succeeded"
  - name: cry
    template: cry
    when: "{{workflow.status}} != Succeeded"
- name: send-email
 container:
  image: alpine:latest
  command: [sh, -c]
  args: ["echo send e-mail: {{workflow.name}} {{workflow.status}}"]
- name: celebrate
 container:
  image: alpine:latest
  command: [sh, -c]
  args: ["echo hooray!"]
- name: cry
 container:
  image: alpine:latest
  command: [sh, -c]
  args: ["echo boohoo!"]
```

Timeouts

要限制工作流的已用时间，可以设置变量*activeDeadlineSeconds*。

```
# To enforce a timeout for a container template, specify a value for activeDeadlineSeconds.
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: timeouts-
spec:
 entrypoint: sleep
 templates:
 - name: sleep
  container:
   image: alpine:latest
   command: [sh, -c]
   args: ["echo sleeping for 1m; sleep 60; echo done"]
  activeDeadlineSeconds: 10        # terminate container template after 10 seconds
```

Volumes

以下示例动态创建卷，然后在两步工作流中使用该卷。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: volumes-pvc-
spec:
 entrypoint: volumes-pvc-example
 volumeClaimTemplates:          # define volume, same syntax as k8s Pod spec
 - metadata:
    name: workdir              # name of volume claim
  spec:
    accessModes: [ "ReadWriteOnce" ]
    resources:
     requests:
      storage: 1Gi             # Gi => 1024 * 1024 * 1024

 templates:
 - name: volumes-pvc-example
  steps:
  - - name: generate
     template: whalesay
  - - name: print
     template: print-message
```

```
- name: whalesay
  container:
    image: docker/whalesay:latest
    command: [sh, -c]
    args: ["echo generating message in volume; cowsay hello world | tee /mnt/vol/hello_world.txt"]
    # Mount workdir volume at /mnt/vol before invoking docker/whalesay
    volumeMounts:               # same syntax as k8s Pod spec
    - name: workdir
      mountPath: /mnt/vol

- name: print-message
  container:
    image: alpine:latest
    command: [sh, -c]
    args: ["echo getting message from volume; find /mnt/vol; cat /mnt/vol/hello_world.txt"]
    # Mount workdir volume at /mnt/vol before invoking docker/whalesay
    volumeMounts:               # same syntax as k8s Pod spec
    - name: workdir
      mountPath: /mnt/vol
```

卷是将大量数据从工作流中的一个步骤移动到另一个步骤的非常有用的方法。根据系统的不同，可以从多个步骤同时访问某些卷。

在某些情况下，您希望访问现有卷，而不是动态创建或销毁一个卷。

```
# Define Kubernetes PVC
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-existing-volume
spec:
  accessModes: [ "ReadWriteOnce" ]
  resources:
    requests:
      storage: 1Gi

---
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: volumes-existing-
```

```
spec:
 entrypoint: volumes-existing-example
 volumes:
 # Pass my-existing-volume as an argument to the volumes-existing-example template
 # Same syntax as k8s Pod spec
 - name: workdir
  persistentVolumeClaim:
   claimName: my-existing-volume

 templates:
 - name: volumes-existing-example
  steps:
  - - name: generate
     template: whalesay
  - - name: print
     template: print-message

 - name: whalesay
  container:
   image: docker/whalesay:latest
   command: [sh, -c]
   args: ["echo generating message in volume; cowsay hello world | tee /mnt/vol/hello_world.txt"]
   volumeMounts:
   - name: workdir
    mountPath: /mnt/vol

 - name: print-message
  container:
   image: alpine:latest
   command: [sh, -c]
   args: ["echo getting message from volume; find /mnt/vol; cat /mnt/vol/hello_world.txt"]
   volumeMounts:
   - name: workdir
    mountPath: /mnt/vol
```

Daemon Containers

工作流可以启动在后台运行的容器（也称为守护程序容器），同时工作流本身继续执行。请注意，当工作流退出调用守护程序的模板范围时，将自动销毁守护程序。守护进程容器可用于启动要测试的服务或用于测试（例如，固定装置）。我们还发现，在运行大型模拟以将数据库作为用于收集和组织结果的守护进程时，它非常实用。守护进程与sidecars相比的最大优势在于它们的存在可以持续跨越多个步骤甚至整个工作流程。

```
apiVersion: argoproj.io/v1alpha1
```

```
kind: Workflow
metadata:
 generateName: daemon-step-
spec:
 entrypoint: daemon-example
 templates:
 - name: daemon-example
  steps:
  - - name: influx
     template: influxdb          # start an influxdb as a daemon (see the influxdb template spec below)

  - - name: init-database          # initialize influxdb
     template: influxdb-client
     arguments:
      parameters:
      - name: cmd
       value: curl -XPOST 'http://{{steps.influx.ip}}:8086/query' --data-urlencode "q=CREATE DATABASE mydb"

  - - name: producer-1           # add entries to influxdb
     template: influxdb-client
     arguments:
      parameters:
      - name: cmd
       value: for i in $(seq 1 20); do curl -XPOST 'http://{{steps.influx.ip}}:8086/write?db=mydb' -d "cpu,host=s
erver01,region=uswest load=$i" ; sleep .5 ; done
    - name: producer-2           # add entries to influxdb
     template: influxdb-client
     arguments:
      parameters:
      - name: cmd
       value: for i in $(seq 1 20); do curl -XPOST 'http://{{steps.influx.ip}}:8086/write?db=mydb' -d "cpu,host=s
erver02,region=uswest load=$((RANDOM % 100))" ; sleep .5 ; done
    - name: producer-3           # add entries to influxdb
     template: influxdb-client
     arguments:
      parameters:
      - name: cmd
       value: curl -XPOST 'http://{{steps.influx.ip}}:8086/write?db=mydb' -d 'cpu,host=server03,region=useast
load=15.4'
```

```
  - - name: consumer          # consume intries from influxdb
      template: influxdb-client
      arguments:
       parameters:
       - name: cmd
         value: curl --silent -G http://{{steps.influx.ip}}:8086/query?pretty=true --data-urlencode "db=mydb" --d
ata-urlencode "q=SELECT * FROM cpu"

 - name: influxdb
   daemon: true               # start influxdb as a daemon
   container:
    image: influxdb:1.2
    restartPolicy: Always       # restart container if it fails
    readinessProbe:           # wait for readinessProbe to succeed
     httpGet:
      path: /ping
      port: 8086

 - name: influxdb-client
   inputs:
    parameters:
    - name: cmd
   container:
    image: appropriate/curl:latest
    command: ["/bin/sh", "-c"]
    args: ["{{inputs.parameters.cmd}}"]
    resources:
     requests:
      memory: 32Mi
      cpu: 100m
```

DAG模板使用任务前缀来引用另一个任务，例如 `{{tasks.influx.ip}}` 。

Sidecars

Sidecar是另一个容器，它与主容器在同一个容器中同时执行，在创建多容器时很实用。

在本示例中，我们创建了一个Sidecar容器，它将Nginx作为简单的Web服务器运行。容器出现的顺序是随机的，因此在此示例中，主容器轮询Nginx容器，直到它准备好为请求提供服务。在设计多容器系统时，这是一个很好的设计模式：在运行主代码之前，请始终等待所需的任何服务。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: sidecar-nginx-
spec:
 entrypoint: sidecar-nginx-example
 templates:
 - name: sidecar-nginx-example
  container:
   image: appropriate/curl
   command: [sh, -c]
   # Try to read from nginx web server until it comes up
   args: ["until `curl -G 'http://127.0.0.1/' >& /tmp/out`; do echo sleep && sleep 1; done && cat /tmp/out"]
  # Create a simple nginx web server
  sidecars:
  - name: nginx
   image: nginx:1.13
```

Kubernetes Resources

在多数情况下，您需要从工作流程管理Kubernetes资源。资源模板允许您创建，删除或更新任何类型的Kubernetes资源。

```
# in a workflow. The resource template type accepts any k8s manifest
# (including CRDs) and can perform any kubectl action against it (e.g. create,
# apply, delete, patch).
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: k8s-jobs-
spec:
 entrypoint: pi-tmpl
 templates:
 - name: pi-tmpl
  resource:              # indicates that this is a resource template
   action: create        # can be any kubectl action (e.g. create, delete, apply, patch)
   # The successCondition and failureCondition are optional expressions.
   # If failureCondition is true, the step is considered failed.
   # If successCondition is true, the step is considered successful.
   # They use kubernetes label selection syntax and can be applied against any field
   # of the resource (not just labels). Multiple AND conditions can be represented by comma
   # delimited expressions.
   # For more details: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/
   successCondition: status.succeeded > 0
   failureCondition: status.failed > 3
   manifest: |           #put your kubernetes spec here
    apiVersion: batch/v1
    kind: Job
    metadata:
     generateName: pi-job-
    spec:
     template:
      metadata:
       name: pi
      spec:
       containers:
       - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
       restartPolicy: Never
     backoffLimit: 4
```

以这种方式创建的资源独立于工作流程。如果希望在删除工作流时删除资源，则可以将Kubernetes垃圾回收与工作流资源一起为所有者引用。

> **说明**
>
> 更新Kubernetes资源时，资源将接受mergeStrategy属性，该属性的取值为*strategy*、*merge*或*json*。如果未提供此属性，则默认为strategy属性。需要注意的是，不能使用策略修补自定义资源，因此必须选择不同的策略。例如，假设您已定义CronTab CustomResourceDefinition，并且以下是CronTab的实例：

```
apiVersion: "stable.example.com/v1"
kind: CronTab
spec:
 cronSpec: "* * * * */5"
 image: my-awesome-cron-image
```

可以使用以下工作流修改此 Crontab。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: k8s-patch-
spec:
 entrypoint: cront-tmpl
 templates:
 - name: cront-tmpl
  resource:
   action: patch
   mergeStrategy: merge            # Must be one of [strategic merge json]
   manifest: |
    apiVersion: "stable.example.com/v1"
    kind: CronTab
    spec:
     cronSpec: "* * * * */10"
     image: my-awesome-cron-image
```

更多资源

- 更多资源展示，请参见argoproj/argo。
- 所有资料示例模板，请参见argoproj/argo/tree/master/examples。

5.3. 开启Workflow UI

Workflow提供了一套UI来展示目前工作流的状态，方便查看每个步骤的容器日志，下面为您介绍如何使用Ingress暴露UI访问端点。

前提条件

- 您已成功创建一个Kubernetes集群，参见创建Kubernetes托管版集群。
- 您已连接到Kubernetes集群的Master节点，参见通过kubectl连接Kubernetes集群。

操作步骤

1. 执行htpasswd命令生成*auth*文件，用于存放用户名密码。

   执行命令：

   ```
   htpasswd -c auth workflow
   ```

   返回结果如下：

   ```
   New password: <workflow>
   New password:
   Re-type new password:
   Adding password for user workflow
   ```

2. 执行如下命令，创建secret来在Kubernetes集群中存放此加密文件。

   ```
   kubectl create secret generic workflow-basic-auth --from-file=auth -n argo
   ```

3. 创建并拷贝内容到*ingress yaml*文件中，并执行 `kubectl apply -f ingress.yaml` 命令，创建workflow-ingress路由。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: workflow-ingress
 namespace: argo
 annotations:
  # type of authentication
  nginx.ingress.kubernetes.io/auth-type: basic
  # name of the secret that contains the user/password definitions
  nginx.ingress.kubernetes.io/auth-secret: workflow-basic-auth
  # message to display with an appropriate context why the authentication is required
  nginx.ingress.kubernetes.io/auth-realm: 'Authentication Required - workflow'
spec:
 rules:
 - host: workflow.<yourTestHost>
  http:
   paths:
   - path: /
    backend:
     serviceName: argo-ui
     servicePort: 80
```

> **说明** 此处的 host 需要替换成您对应的集群地址（即为**集群信息**中的测试域名的值，例如：workflow.cfb131.cn-zhangjiakou.alicontainer.com）。

4. 在浏览器输入*workflow.<yourTestHost>*，按照提示输入密码就能看到如下界面。

   

   您可以根据需要查看工作流的状态。

   

5.4. AGS命令行帮助

AGS 是阿里云基因服务的通用命令行工具， 目前主要集成 argo 功能并且适配阿里云各个产品的 addon功能命令。

AGS 下载和安装

AGS 默认使用阿里云ak调用各个服务，目前集成了阿里云日志服务用于收集Pod日志。使用此功能请在集群创建的时候选择开启日志服务。

执行如下命令，下载 AGS 工具并配置运行权限。

```
wget http://ags-hub.oss-cn-hangzhou.aliyuncs.com/ags-linux && chmod +x ags-linux && mv ags-linux /usr/local/bin/ags
```

> **说明**
>
> `ags config init` 根据交互式命令行输入CLI需要的信息，初始化完成后，配置文件会被默认存储到*~/.ags/config*文件中。可以通过 `ags config show` 展示配置好的信息。其中AccessKeySecret会被加密存储。
>
> 如果您需要使用日志采集功能，则需要配置ags config。另外为了安全考虑，您可以给CLI单独创建一个ak，赋予日志服务权限即可。

如果您使用的是Kubernetes 托管版集群，您可以通过kubectl连接Kubernetes集群，并执行如下命令，实现通过 CloudShell 使用 ags 命令行工具。

```
wget http://ags-hub.oss-cn-hangzhou.aliyuncs.com/ags-linux && chmod +x ags-linux && mv ags-linux /usr/local/bin/ags
```

AGS 功能特性

- 完全兼容argo 命令
- Pod被删除后从阿里云日志服务拉取日志查看功能
- 集成kubectl 命令，用户可以直接使用kubectl命令操作集群
- 提供install uninstall 命令，一键安装或者卸载所需资源
- get workflow 命令提供资源使用量查看功能
- Yaml 模板允许下划线等特殊字符
- securityContext 安全支持
- Pod pending/fails 状态与workflow 状态同步
- Yaml定义自动retry功能
- 基于最近失败断点retry整个workflow
- 支持 ECI Serverless Kubernetes 架构

AGS 基本概览

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags
ags is the command line interface to Alibaba Cloud Genomics Compute Service

Usage:
 ags [flags]
 ags [command]

Available Commands:
 completion  output shell completion code for the specified shell (bash or zsh)
 config      setup ags client necessary info
```

```
  delete     delete a workflow and its associated pods
  get        display details about a workflow
  help       Help about any command
  install    install ags
  kubectl    kubectl command
  lint       validate a file or directory of workflow manifests
  list       list workflows
  logs       view logs of a workflow
  resubmit   resubmit a workflow
  resume     resume a workflow
  retry      retry a workflow
  submit     submit a workflow
  suspend    suspend a workflow
  terminate  terminate a workflow
  uninstall  uninstall ags
  version    Print version information
  wait       waits for a workflow to complete
  watch      watch a workflow until it completes

Flags:
    --as string                Username to impersonate for the operation
    --as-group stringArray     Group to impersonate for the operation, this flag can be repeated to specify m
ultiple groups.
    --certificate-authority string  Path to a cert file for the certificate authority
    --client-certificate string     Path to a client certificate file for TLS
    --client-key string        Path to a client key file for TLS
    --cluster string           The name of the kubeconfig cluster to use
    --context string           The name of the kubeconfig context to use
-h, --help                     help for ags
    --insecure-skip-tls-verify     If true, the server's certificate will not be checked for validity. This will make y
our HTTPS connections insecure
    --kubeconfig string        Path to a kube config. Only required if out-of-cluster
-n, --namespace string         If present, the namespace scope for this CLI request
    --password string          Password for basic authentication to the API server
    --request-timeout string   The length of time to wait before giving up on a single server request. Non-ze
ro values should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero means don't timeout req
uests. (default "0")
    --server string            The address and port of the Kubernetes API server
    --token string             Bearer token for authentication to the API server
    --user string              The name of the kubeconfig user to use
```

```
  --username string          Username for basic authentication to the API server

Use "ags [command] --help" for more information about a command.
```

Install/Uninstall

您可以通过ags install 命令，一键安装AGS所需资源，无需手动安装。

也可以通过ags uninstall命令， 一键卸载AGS所有资源。

5.5. AGS帮助示例

本文主要为您提供AGS帮助示例。

前提条件

- 您已成功创建一个Kubernetes集群。参见创建Kubernetes托管版集群。
- 您已连接到Kubernetes集群，参见通过kubectl连接Kubernetes集群。

Log

1. 执行 ags config sls 命令，在AGS上配置并安装日志服务。原生Argo查看Pod日志只能从本地拉取，当Pod或者所在节点被删除后，日志也会随之丢失，这对查看错误分析以及原因等带来很大问题。 如果将日志上传到阿里云日志服务，即使节点消失也会从日志服务重新拉取，将日志持久化。
2. 执行 ags logs 命令，查看工作流的日志。本例中执行 ags logs POD/WORKFLOW ，查看Pod/Workflow的日志。

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags logs
view logs of a workflow

Usage:
 ags logs POD/WORKFLOW [flags]

Flags:
 -c, --container string   Print the logs of this container (default "main")
 -f, --follow             Specify if the logs should be streamed.
 -h, --help               help for logs
 -l, --recent-line int    how many lines to show in one call (default 100)
     --since string       Only return logs newer than a relative duration like 5s, 2m, or 3h. Defaults to all logs. Only one of since-time / since may be used.
     --since-time string  Only return logs after a specific date (RFC3339). Defaults to all logs. Only one of since-time / since may be used.
     --tail int           Lines of recent log file to display. Defaults to -1 with no selector, showing all log lines otherwise 10, if a selector is provided. (default -1)
     --timestamps         Include timestamps on each line in the log output
 -w, --workflow           Specify that whole workflow logs should be printed
```

> 说明
>
> - 如果Pod存在本机，AGS会从本地将Pod日志查询出来，所有flag兼容原Argo命令。
> - 如果Pod已经被删除，AGS会从阿里云日志服务来查询日志，默认返回最近100条日志，可以通过-l flag来指定到底返回多少条日志。

List

您可以通过--limit参数选择查看的Workflow条目数。

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags remote list --limit 8
+----------------------+------------------------------+------------+
|    JOB NAME      |       CREATE TIME       | JOB STATUS |
+----------------------+------------------------------+------------+
| merge-6qk46        | 2020-09-02 16:52:34 +0000 UTC | Pending    |
| rna-mapping-gpu-ck4cl | 2020-09-02 14:47:57 +0000 UTC | Succeeded  |
| wgs-gpu-n5f5s      | 2020-09-02 13:14:14 +0000 UTC | Running    |
| merge-5zjhv        | 2020-09-02 12:03:11 +0000 UTC | Succeeded  |
| merge-jjcw4        | 2020-09-02 10:44:51 +0000 UTC | Succeeded  |
| wgs-gpu-nvxr2      | 2020-09-01 22:18:44 +0000 UTC | Succeeded  |
| merge-4vg42        | 2020-09-01 20:52:13 +0000 UTC | Succeeded  |
| rna-mapping-gpu-2ss6n | 2020-09-01 20:34:45 +0000 UTC | Succeeded  |
```

集成kubectl命令

您可以执行如下命令，查看Pod状态以及其他需要使用kubectl命令的情况。

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags get test-v2
Name:          test-v2
Namespace:     default
ServiceAccount:  default
Status:        Running
Created:       Thu Nov 22 11:06:52 +0800 (2 minutes ago)
Started:       Thu Nov 22 11:06:52 +0800 (2 minutes ago)
Duration:      2 minutes 46 seconds

STEP      PODNAME       DURATION  MESSAGE
● test-v2
└---● bcl2fq  test-v2-2716811808  2m

[root@iZwz92q9h36kv8posr0i6uZ ~]# ags kubectl describe pod test-v2-2716811808
Name:          test-v2-2716811808
Namespace:     default
Priority:      0
PriorityClassName: <none>
Node:          cn-shenzhen.i-wz9gwobtqrbjgfnqxl1k/192.168.0.94
Start Time:    Thu, 22 Nov 2018 11:06:52 +0800
Labels:        workflows.argoproj.io/completed=false
               workflows.argoproj.io/workflow=test-v2
Annotations:   workflows.argoproj.io/node-name=test-v2[0].bcl2fq
               workflows.argoproj.io/template={"name":"bcl2fq","inputs":{},"outputs":{},"metadata":{},"containe
r":{"name":"main","image":"registry.cn-hangzhou.aliyuncs.com/dahu/curl-jp:1.2","command":["sh","-c"],"
ar...
Status:        Running
IP:            172.16.*.***
Controlled By:  Workflow/test-v2
```

通过使用ags kubectl命令，可以查看到describe pod的状态信息，所有kubectl原生命令AGS均支持。

集成ossutil命令

AGS初始化完毕后，您可以使用如下命令进行文件的上传和查看。

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags oss cp test.fq.gz oss://my-test-shenzhen/fasq/
Succeed: Total num: 1, size: 690. OK num: 1(upload 1 files).

average speed 3000(byte/s)

0.210685(s) elapsed
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags oss ls oss://my-test-shenzhen/fasq/
LastModifiedTime             Size(B)  StorageClass   ETAG                             ObjectName
2020-09-02 17:20:34 +0800 CST      690    Standard  9FDB86F70C6211B2EAF95A9B06B14F7E    oss://my-test-
shenzhen/fasq/test.fq.gz
Object Number is: 1

0.117591(s) elapsed
```

通过使用ags oss命令，可以进行文件的上传下载等，所有的ossutil原生命令AGS均支持。

查看Workflow资源使用量

1. 创建并拷贝内容到*arguments-workflow-resource.yaml*文件中，并执行 `ags submit arguments-workflow-resource.yaml` 命令，指定resource requests。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 name: test-resource
spec:
 arguments: {}
 entrypoint: test-resource-
 templates:
 - inputs: {}
  metadata: {}
  name: test-resource-
  outputs: {}
  parallelism: 1
  steps:
  - - arguments: {}
     name: bcl2fq
     template: bcl2fq
 - container:
    args:
    - id > /tmp/yyy;echo `date` > /tmp/aaa;ps -e -o comm,euid,fuid,ruid,suid,egid,fgid,gid,rgid,sgid,supgi
d
     > /tmp/ppp;ls -l /tmp/aaa;sleep 100;pwd
    command:
    - sh
    - -c
    image: registry.cn-hangzhou.aliyuncs.com/dahu/curl-jp:1.2
    name: main
    resources:          #don't use too much resources
     requests:
      memory: 320Mi
      cpu: 1000m
  inputs: {}
  metadata: {}
  name: bcl2fq
  outputs: {}
```

2. 执行 `ags get test456 --show` 命令，查看Workflow资源使用。本例中，结果显示的是Pod和test456使用的核/时。

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags get test456 --show
Name:          test456
Namespace:     default
ServiceAccount:  default
Status:        Succeeded
Created:       Thu Nov 22 14:41:49 +0800 (2 minutes ago)
Started:       Thu Nov 22 14:41:49 +0800 (2 minutes ago)
Finished:      Thu Nov 22 14:43:30 +0800 (27 seconds ago)
Duration:      1 minute 41 seconds
Total CPU:     0.02806  (core*hour)
Total Memory:  0.00877  (GB*hour)

STEP      PODNAME       DURATION  MESSAGE  CPU(core*hour)  MEMORY(GB*hour)
✔ test456                       0        0
└---✔ bcl2fq  test456-4221301428  1m           0.02806       0.00877
```

securityContext安全支持

1. 创建并拷贝内容到*arguments-security-context.yaml*文件中，并执行 `ags submit arguments-security-context.yaml` 命令，绑定对应的psp来进行权限控制。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 name: test
spec:
 arguments: {}
 entrypoint: test-security-
 templates:
 - inputs: {}
  metadata: {}
  name: test-security-
  outputs: {}
  parallelism: 1
  steps:
  - - arguments: {}
     name: bcl2fq
     template: bcl2fq
 - container:
    args:
    - id > /tmp/yyy;echo `date` > /tmp/aaa;ps -e -o comm,euid,fuid,ruid,suid,egid,fgid,gid,rgid,sgid,supgi
d
     > /tmp/ppp;ls -l /tmp/aaa;sleep 100;pwd
    command:
    - sh
    - -c
    image: registry.cn-hangzhou.aliyuncs.com/dahu/curl-jp:1.2
    name: main
    resources:          #don't use too much resources
     requests:
      memory: 320Mi
      cpu: 1000m
  inputs: {}
  metadata: {}
  name: bcl2fq
  outputs: {}
  securityContext:
   runAsUser: 800
```

YAML定义自动重试功能

bash命令会由于不明原因失败，重试就可以解决，AGS提供一种基于YAML配置的自动重启机制，当Pod内命令运行失败后，会自动拉起重试，并且可以设置重试次数。

1. 创建并拷贝内容到*arguments-auto-retry.yaml*文件中，并执行 `ags submit arguments-auto-retry.yaml` 命令，配置Workflow的自动重启机制。

```
# This example demonstrates the use of retries for a single container.
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: retry-container-
spec:
 entrypoint: retry-container
 templates:
 - name: retry-container
  retryStrategy:
   limit: 10
  container:
   image: python:alpine3.6
   command: ["python", -c]
   # fail with a 66% probability
   args: ["import random; import sys; exit_code = random.choice([0, 1, 1]); sys.exit(exit_code)"]
```

基于最近失败断点重试整个Workflow

在整个Workflow运行中，有时候任务中的某个步骤会失败，这时候希望从某个失败的节点重试Workflow，类似断点续传的断点重试功能。

1. 执行 `ags get test456 --show` 命令，查看workflow test456从哪个步骤断点。

```
[root@iZwz92q9h36kv8posr0i6uZ ~]# ags get test456 --show
Name:          test456
Namespace:     default
ServiceAccount:  default
Status:        Succeeded
Created:       Thu Nov 22 14:41:49 +0800 (2 minutes ago)
Started:       Thu Nov 22 14:41:49 +0800 (2 minutes ago)
Finished:      Thu Nov 22 14:43:30 +0800 (27 seconds ago)
Duration:      1 minute 41 seconds
Total CPU:     0.0572  (core*hour)
Total Memory:    0.01754  (GB*hour)

STEP      PODNAME       DURATION  MESSAGE  CPU(core*hour)  MEMORY(GB*hour)
 ✔ test456                        0         0
  └---✔ bcl2fq  test456-4221301428  1m          0.02806      0.00877
  └---X bcl2fq  test456-4221301238  1m          0.02806      0.00877
```

2. 执行 `ags retry test456` 命令，从最近失败断点处继续重试workflow test456。

使用ECI运行workflow

ECI操作请参见弹性容器实例ECI。

配置使用ECI前，请先安装AGS，请参见AGS 下载和安装。

1. 执行 `kubectl get cm -n argo` 命令，获取Workflow对应的YAML文件名称。

```
[root@iZwz9f4ofes6kbo01ipuf1Z ~]# kubectl get cm -n argo
NAME                       DATA    AGE
workflow-controller-configmap  1      4d
```

2. 执行 `kubectl get cm -n argo workflow-controller-configmap -o yaml` 命令，打开*workflow-controller-configmap.yaml*文件，并使用如下内容覆盖当前YAML文件的内容。

```
apiVersion: v1
data:
 config: |
  containerRuntimeExecutor: k8sapi
kind: ConfigMap
```

3. 执行 `kubectl delete pod <podName>` 命令，重启argo controller。

> **说明** 这里的*podName*为workflow所在的Pod的名称。

4. 创建并拷贝内容到*arguments-workflow-eci.yaml*文件中，并执行 `ags submit arguments-workflow-eci.y`

aml 命令，在ECI上运行的容器添加nodeSelector和Tolerations这两个标识。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: hello-world-
spec:
 entrypoint: whalesay
 templates:
 - name: whalesay
  container:
   image: docker/whalesay
   command: [env]
   #args: ["hello world"]
   resources:
    limits:
     memory: 32Mi
     cpu: 100m
  nodeSelector:          # 添加nodeSelector
   type: virtual-kubelet
  tolerations:           # 添加tolerations
  - key: virtual-kubelet.io/provider
   operator: Exists
  - key: alibabacloud.com
   effect: NoSchedule
```

查看Workflow实际资源使用量以及峰值

ags workflow controller会通过metrics-server自动获取Pod每分钟的实际资源使用量，并且统计出来总量和各个Pod的峰值使用量。

执行 ags get steps-jr6tw --metrics 命令，查看Workflow实际资源使用量以及峰值。

```
➜ ags get steps-jr6tw --metrics
Name:          steps-jr6tw
Namespace:     default
ServiceAccount:  default
Status:        Succeeded
Created:       Tue Apr 16 16:52:36 +0800 (21 hours ago)
Started:       Tue Apr 16 16:52:36 +0800 (21 hours ago)
Finished:      Tue Apr 16 19:39:18 +0800 (18 hours ago)
Duration:      2 hours 46 minutes
Total CPU:     0.00275  (core*hour)
Total Memory:    0.04528  (GB*hour)

STEP       PODNAME          DURATION  MESSAGE  CPU(core*hour)  MEMORY(GB*hour)  MaxCpu(core)  MaxM
emory(GB)
 ✔ steps-jr6tw                       0         0         0         0
  └---✔ hello1  steps-jr6tw-2987978173  2h          0.00275     0.04528     0.000005    0.00028
```

设置Workflow优先级

当前面有一些任务正在运行时，有一个紧急任务急需运行，此时，您可以给Workflow设置高、中、低的优先级，高优先级抢占低优先级任务的资源。

- 您可以给某个Pod设置高优先级，示例如下：

  创建并拷贝内容到*arguments-high-priority-taskA.yaml*文件中，并执行 `ags submit arguments-high-priority-taskA.yaml` 命令，给任务A设置高优先级。

  ```
  apiVersion: scheduling.k8s.io/v1beta1
  kind: PriorityClass
  metadata:
    name: high-priority
  value: 1000000
  globalDefault: false
  description: "This priority class should be used for XYZ service pods only."
  ```

- 您可以给某个Pod设置中优先级，示例如下：

  创建并拷贝内容到*arguments-high-priority-taskB.yaml*文件中，并执行 `ags submit arguments-high-priority-taskB.yaml` 命令，给任务B设置中优先级。

```
apiVersion: scheduling.k8s.io/v1beta1
kind: PriorityClass
metadata:
 name: medium-priority
value: 100
globalDefault: false
description: "This priority class should be used for XYZ service pods only."
```

- 您也可以一个Workflow设置高优先级，示例如下：

  创建并拷贝内容到*arguments-high-priority-Workflow.yaml*文件中，并执行 `ags submit arguments-high-priority-Workflow.yaml` 命令，给Workflow中所有的Pod设置高优先级。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow            # new type of k8s spec
metadata:
 generateName: high-proty-  # name of the workflow spec
spec:
 entrypoint: whalesay       # invoke the whalesay template
 podPriorityClassName: high-priority # workflow level priority
 templates:
 - name: whalesay          # name of the template
  container:
   image: ubuntu
   command: ["/bin/bash", "-c", "sleep 1000"]
   resources:
    requests:
     cpu: 3
```

下面以一个Workflow里面含有两个Pod，分别给一个Pod设置中优先级，另一个Pod设置高优先级，此时，高优先级的Pod就能抢占低优先级Pod的资源。

1. 创建并拷贝内容到*arguments-high-priority-steps.yaml*文件中，并执行 `ags submit arguments-high-priority-steps.yaml` 命令，给Pod设置优先级。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
 generateName: steps-
spec:
 entrypoint: hello-hello-hello

 templates:
 - name: hello-hello-hello
  steps:
  - - name: low
     template: low
  - - name: low-2
     template: low
   - name: high
     template: high

 - name: low
  container:
   image: ubuntu
   command: ["/bin/bash", "-c", "sleep 30"]
   resources:
    requests:
     cpu: 3

 - name: high
  priorityClassName: high-priority # step level priority
  container:
   image: ubuntu
   command: ["/bin/bash", "-c", "sleep 30"]
   resources:
    requests:
     cpu: 3
```

2. 运行结果为高优先级Pod会抢占并删除旧的Pod，执行结果如下所示。

```
Name:          steps-sxvrv
Namespace:     default
ServiceAccount: default
Status:        Failed
Message:       child 'steps-sxvrv-1724235106' failed
Created:       Wed Apr 17 15:06:16 +0800 (1 minute ago)
Started:       Wed Apr 17 15:06:16 +0800 (1 minute ago)
Finished:      Wed Apr 17 15:07:34 +0800 (now)
Duration:      1 minute 18 seconds

STEP       PODNAME          DURATION  MESSAGE
 ✖ steps-sxvrv                       child 'steps-sxvrv-1724235106' failed
 ├---✔ low    steps-sxvrv-3117418100  33s
 └-・-✔ high   steps-sxvrv-603461277  45s
   └-⚠ low-2  steps-sxvrv-1724235106  45s     pod deleted
```

> **说明** 这里高优先级任务会自动抢占低优先Pod所占用资源，会停止低优先级任务，中断正在运行的进程，所以使用时要非常谨慎。

Workflow Filter

在ags get workflow中，针对较大的Workflow可以使用filter列出指定状态的Pod。

1. 执行 `ags get <pod名称> --status Running` 命令，列出指定状态的Pod。

```
[root@iZ8vb19036cvgu1tpxkzl1Z workflow-prioty]# ags get pod-limits-n262v --status Running
Name:          pod-limits-n262v
Namespace:     default
ServiceAccount:  default
Status:        Running
Created:       Wed Apr 17 15:59:08 +0800 (1 minute ago)
Started:       Wed Apr 17 15:59:08 +0800 (1 minute ago)
Duration:      1 minute 17 seconds
Parameters:
 limit:       300

STEP          PODNAME            DURATION  MESSAGE
● pod-limits-n262v
  ├-● run-pod(13:13)  pod-limits-n262v-3643890604  1m
  ├-● run-pod(14:14)  pod-limits-n262v-4115394302  1m
  ├-● run-pod(16:16)  pod-limits-n262v-3924248206  1m
  ├-● run-pod(17:17)  pod-limits-n262v-3426515460  1m
  ├-● run-pod(18:18)  pod-limits-n262v-824163662   1m
  ├-● run-pod(20:20)  pod-limits-n262v-4224161940  1m
  ├-● run-pod(22:22)  pod-limits-n262v-1343920348  1m
  ├-● run-pod(2:2)    pod-limits-n262v-3426502220  1m
  ├-● run-pod(32:32)  pod-limits-n262v-2723363986  1m
  ├-● run-pod(34:34)  pod-limits-n262v-2453142434  1m
  ├-● run-pod(37:37)  pod-limits-n262v-3225742176  1m
  ├-● run-pod(3:3)    pod-limits-n262v-2455811176  1m
  ├-● run-pod(40:40)  pod-limits-n262v-2302085188  1m
  ├-● run-pod(6:6)    pod-limits-n262v-1370561340  1m
```

2. 执行 `ags get <pod名称> --sum-info` 命令，统计当前Pod状态信息。

```
[root@iZ8vb19036cvgu1tpxkzl1Z workflow-prioty]# ags get pod-limits-n262v --sum-info --status Error
Name:          pod-limits-n262v
Namespace:     default
ServiceAccount:  default
Status:        Running
Created:       Wed Apr 17 15:59:08 +0800 (2 minutes ago)
Started:       Wed Apr 17 15:59:08 +0800 (2 minutes ago)
Duration:      2 minutes 6 seconds
Pending:       198
Running:       47
Succeeded:     55
Parameters:
 limit:       300

STEP          PODNAME  DURATION  MESSAGE
 ● pod-limits-n262v
```

敏捷版Autoscaler使用流程

在敏捷版使用autoscaler需要用户提前创建或者已经具备以下资源：

- 您已经有一个VPC。
- 您已经有一个vSwitch。
- 您已经设置好一个安全组。
- 您已经获取到敏捷版的APIServer内网地址。
- 您明确扩容节点的规格。
- 您已创建好一个ECS实例且拥有公网访问能力。

您可以在AGS命令行，按照界面提示进行如下操作。

```
$ags config autoscaler根据提示输入对应的值Please input vswitchs with comma separated
vsw-hp3cq3fnv47bpz7x58wfe
Please input security group id
sg-hp30vp05x6tlx13my0qu
Please input the instanceTypes with comma separated
ecs.c5.xlarge
Please input the new ecs ssh password
xxxxxxxx
Please input k8s cluster APIServer address like(192.168.1.100)
172.24.61.156
Please input the autoscaling mode (current: release. Type enter to skip.)
Please input the min size of group (current: 0. Type enter to skip.)
Please input the max size of group (current: 1000. Type enter to skip.)
Create scaling group successfully.
Create scaling group config successfully.
Enable scaling group successfully.
Succeed
```

配置完成后，登录弹性伸缩控制台， 可以看到创建好的自动伸缩组。

配置使用ags configmap

本例中，默认使用hostNetwork。

1. 执行 `kubectl get cm -n argo` 命令，获取Workflow对应的YAML文件名称。

   ```
   [root@iZ8vb19036cvgu1tpxkzl1Z ~]# kubectl get cm -n argo
   ```

   ```
   NAME                       DATA  AGE
   workflow-controller-configmap  1    6d23h
   ```

2. 执行 `kubectl edit cm workflow-controller-configmap -n argo` 命令，打开*workflow-controller-configmap.yaml*文件，将如下内容填入当前YAML文件中。

   ```
   data:
    config: |
     extraConfig:
       enableHostNetwork: true
       defaultDnsPolicy: Default
   ```

   填入完成后，*workflow-controller-configmap.yaml*全文如下所示。

```
apiVersion: v1
data:
 config: |
  extraConfig:
   enableHostNetwork: true
   defaultDnsPolicy: Default
kind: ConfigMap
metadata:
 name: workflow-controller-configmap
 namespace: argo
```

3. 配置完成后，新部署的Workflow均会默认使用*hostNetwork*，且dnsPolicy为*Default*。
4. （可选）如果配置了psp，需要在psp中对应的YAML文件增加如下内容。

```
hostNetwork: true
```

> **说明** 如果该yaml文件中已有hostNetwork参数，需要将值改为*ture*。

完整YAML示例模板如下：

展开查看完整示例YAML文件

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
 name: restricted
 annotations:
  seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default,runtime/default'
  apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
  seccomp.security.alpha.kubernetes.io/defaultProfileName:  'runtime/default'
  apparmor.security.beta.kubernetes.io/defaultProfileName:  'runtime/default'
spec:
 privileged: false
 # Required to prevent escalations to root.
 allowPrivilegeEscalation: false
 # This is redundant with non-root + disallow privilege escalation,
 # but we can provide it for defense in depth.
 requiredDropCapabilities:
  - ALL
 # Allow core volume types.
 volumes:
  - 'configMap'
  - 'emptyDir'
```

```
  - 'projected'
  - 'secret'
  - 'downwardAPI'
  # Assume that persistentVolumes set up by the cluster admin are safe to use.
  - 'persistentVolumeClaim'
hostNetwork: false
hostIPC: false
hostPID: false
runAsUser:
  # Require the container to run without root privileges.
  rule: 'MustRunAsNonRoot'
seLinux:
  # This policy assumes the nodes are using AppArmor rather than SELinux.
  rule: 'RunAsAny'
supplementalGroups:
  rule: 'MustRunAs'
  ranges:
    # Forbid adding the root group.
    - min: 1
      max: 65535
fsGroup:
  rule: 'MustRunAs'
  ranges:
    # Forbid adding the root group.
    - min: 1
      max: 65535
readOnlyRootFilesystem: false
```

5.6. 通过AGS服务调用加速工作流

通过本地工作流和AGS远程工作流组成的混合工作流，可以将耗时长、计算量大、标准流程的任务放在AGS服务端执行。将耗时短、所需资源有限、需要自行定制流程的任务放在本地工作流中执行，数据通过OSS进行中间流转，从而有效的提高任务执行效率、节省资源成本。本文将通过RemoteAPI实现Mapping过程为例，演示如何编写并运行一个AGS混合工作流。

前提条件

开通服务。

操作步骤

1. 配置AGS。

   关于AGS的下载和安装，请参见AGS命令行帮助。

```
ags config init
```

配置AGS完成后，会自动生成AGS配置文件*config*。

2. 创建ConfigMap。

```
kubectl create configmap config --from-file=~/.ags/config -n argo
```

3. 创建*hybridStorage.yaml*文件。

根据实际情况修改以下YAML文件中的NAS的IP、目录，OSS的AKID、AKSecret、Bucket等参数。

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: gene-shenzhen-cache-nas
 namespace: argo
 labels:
  alicloud-pvname: gene-shenzhen-cache-nas
spec:
 capacity:
  storage: 20Gi
 storageClassName: alicloud-nas
 accessModes:
  - ReadWriteMany
 csi:
  driver: nasplugin.csi.alibabacloud.com
  volumeHandle: gene-shenzhen-cache-nas-pvc
  volumeAttributes:
   server: "xxxxxxxx-fbi71.cn-beijing.nas.aliyuncs.com"
   path: "/tarTest"
 mountOptions:
 - nolock,tcp,noresvport
 - vers=3
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: gene-shenzhen-cache-nas-pvc
 namespace: argo
spec:
 accessModes:
  - ReadWriteMany
 resources:
```

```
    requests:
      storage: 20Gi
  storageClassName: alicloud-nas
  selector:
    matchLabels:
      alicloud-pvname: gene-shenzhen-cache-nas
---
apiVersion: v1
kind: Secret
metadata:
  name: oss-secret
  namespace: argo
stringData:
  akId: xxxxxxxxxx
  akSecret: xxxxxxxxxxx
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: gene-shenzhen-cache-oss-pvc
  namespace: argo
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 200Gi
  selector:
    matchLabels:
      alicloud-pvname: gene-shenzhen-cache-oss
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: gene-shenzhen-cache-oss
  namespace: argo
  labels:
    alicloud-pvname: gene-shenzhen-cache-oss
spec:
  capacity:
    storage: 200Gi
```

```
  accessModes:
   - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  csi:
   driver: ossplugin.csi.alibabacloud.com
   volumeHandle: gene-shenzhen-cache-oss // 需要和PV名字一致。
   nodePublishSecretRef:
    name: oss-secret
    namespace: argo
   volumeAttributes:
    bucket: "oss-test-tsk"
    url: "oss-cn-beijing.aliyuncs.com"
    otherOpts: "-o max_stat_cache_size=0 -o allow_other"
```

4. 创建存储数据的PV和PVC。

```
kubectl create -f hybridStorage.yaml
```

5. 创建*hybrid.yam*l文件。

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow              #new type of k8s spec
metadata:
 generateName: mpileup-   #name of workflow spec
 namespace: argo
spec:
 entrypoint: mpileup      #invoke the whalesay template
 arguments:
  parameters:

  # fastq define
  - name: bucket
    value: "my-test-shenzhen"
  - name: fastqDir
    value: "sample"
  - name: fastq1
    value: "MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz"
  - name: fastq2
    value: "MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz"
  - name: reference
    value: "hg19"

  # gene define
```

```
  # gene demo
  - name: bamDir
    value: "outbam/bam"
  - name: vcfDir
    value: "output/vcf"
  - name: bamFileName
    value: "gene.bam"
  - name: cpuNumber
    value: "32"
  - name: vcfFileName
    value: "gene.vcf"
  - name: service
    value: "s"

 volumes:
 - name: workdir
   persistentVolumeClaim:
     claimName: gene-shenzhen-cache-nas-pvc
 - name: outputdir
   persistentVolumeClaim:
     claimName: gene-shenzhen-cache-oss-pvc
 - name: agsconfig
   configMap:
     name: config

 templates:
 #It is executed remotely and accelerated
 - name: agsmapping
   container:
     image: registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1
     imagePullPolicy: Always
     command: [sh,-c]
     args:
     - ags remote run mapping --region cn-shenzhen --fastq1 sample/{{workflow.parameters.fastq1}} --fas
tq2 sample/{{workflow.parameters.fastq2}} --bucket {{workflow.parameters.bucket}} --output-bam {{w
orkflow.parameters.bamDir}}/{{workflow.parameters.bamFileName}} --reference {{workflow.paramete
rs.reference}} --service s --watch;
     volumeMounts:
     - name: agsconfig
       mountPath: /root/.ags/config
       subPath: config
```

```
#Download BAM file from OSS to NAS
- name: mpileupprepare
  container:
    image: registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1
    imagePullPolicy: Always
    command: [sh,-c]
    args:
    - cp /output/outbam/bam/gene.bam /data/sample/.
    resources:
      requests:
        memory: 8Gi
        cpu: 4
    volumeMounts:
    - name: workdir
      mountPath: /data
    - name: outputdir
      mountPath: /output
#It is executed locally
- name: samtoolsindex
  retryStrategy:
    limit: 3
  container:
    image: registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1
    command: [sh,-c]
    args:
    - samtools index -@ {{workflow.parameters.cpuNumber}} /data/sample/gene.bam
    volumeMounts:
    - name: workdir
      mountPath: /data
    resources:
      requests:
        memory: 20Gi
        cpu: 4

##Upload index file from NAS to OSS
- name: upload
  retryStrategy:
    limit: 3
  container:
    image: registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1
```

```
    command: [sh,-c]
    args:
    - cp /data/sample/gene.bam.bai /output/outbam/bam/.
    volumeMounts:
    - name: workdir
      mountPath: /data
    - name: outputdir
      mountPath: /output
    resources:
      requests:
        memory: 20Gi
        cpu: 4


  - name: mpileup
    dag:
      tasks:
      - name: agsmappingtask
        template: agsmapping

      - name: download-data
        dependencies: [agsmappingtask]
        template: mpileupprepare

      - name: index
        dependencies: [download-data]
        template: samtoolsindex

      - name: upload-data
        dependencies: [index]
        template: upload
```

6. 创建混合工作流。

```
kubectl create -f hybrid.yaml
```

创建混合工作流成功后，会自动执行混合工作流。

找到gene-shenzhen-cache-oss-pvc对应的Bucket，进入Bucket下的*/outbam/bam/*，可以看到*gene.bam.bai*，说明混合工作流创建并执行成功。

6.WDL工作流

6.1. 创建WDL工作流

WDL（Workflow Description Language）是由Broad Institute开发的一种流程开发语言，简单易用，能够有效的提高生物信息工作流的构建效率。本文介绍如何通过AGS在ACK集群上编写并执行WDL工作流。

前提条件

- 已创建ACK集群，详细介绍请参见创建Kubernetes托管版集群。
- 已拥有一个或多个存储服务（NAS存储卷或者符合NFS协议的文件存储、OSS存储），用于存储输入数据和输出结果，详细介绍请参见使用NAS动态存储卷。

在ACK上运行WDL的优势

- 兼容社区CronwellServer，完整兼容WDL的流程定义，对遗留流程无需修改，便可以通过AGS在ACK上运行WDL流程。WDL的详细介绍请参见WDL。
- 对Task资源申请优化，通过Pod Guarantee QoS方式，避免资源过度争取造成节点负载过高和效率下降。
- 与阿里云存储的无缝整合，目前支持直接访问OSS和NAS，并支持多数据源的挂载。

与AGS工作流的区别

- CronwellServer在以下方面仍然落后于云原生的AGS工作流。
  - 资源控制粒度（CPU、Mem min、Mem max）方面。
  - 调度优化，自动重试，资源上限的动态调整。
  - 监控、日志等方面。
- 在集群资源使用水位上低于AGS，在批量样本投递成功率上也低于AGS工作流。对于大批量重复性的工作流仍然建议改造成原生的AGS工作流来提升效率，详细介绍请参见创建工作流。
- 对于高CPU消耗的Mapping、HC、Mutecv2等流程可以使用AGS API来降低处理成本节省和加速，详细介绍请参见通过AGS处理全基因组测序WGS。

步骤一：部署应用

构建WDL工作流所需的组件被打包成为一个Helm Chart，作为一个应用放在应用市场中，避免了复杂的环境配置，方便进行一键部署。

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**市场 > 应用目录**。
3. 在**应用目录**页面右侧搜索框中搜索ack-ags-wdl，找到ack-ags-wdl，然后单击ack-ags-wdl。
4. 在**应用目录 - ack-ags-wdl**页面单击**参数**页签。
5. 配置应用参数。

```
# PVCs to be mounted
naspvcs:
 - naspvc1
 - naspvc2
osspvcs:
 - osspvc1
```

```
 - osspvc1
 - osspvc2

# config in transfer-pvc.yaml、storageclass.yaml
naspvc1:
 # modify it to actual url of NAS/NFS server .
 server : "XXXXXX-fbi71.cn-beijing.nas.aliyuncs.com"
 # absolute path of your data on NAS/NFS, pls modify it to your actual path of data root
 path: "/tarTest" #eg "/tarTest"
 # strorage driver. flexVolume or csi,default is csi, if your kubernetes use flexVolume,chang it to flexVolu
me
 driver: "csi" # by default is csi, you could change to flexVolume for early version of K8s (<= 1.14.x)
 nasbasepath: "/ags-wdl-nas"
 # mountoptions. if you use local NFS, modify it to your mount options.
 # config in storageclass.yaml 、transfer-pvc.yaml
 mountVers: "3" #mount version-svc:
 mountOptions: "nolock,tcp,noresvport"  # mount options, for local NFS server, you could remove nor
esvport option

naspvc2:
 # modify it to actual url of NAS/NFS server .
 server : "XXXXXXX-fbi71.cn-beijing.nas.aliyuncs.com"
 # absolute path of your data on NAS/NFS, pls modify it to your actual path of data root
 path: "/tarTest/bwatest" #eg "/tarTest"
 # strorage driver. flexVolume or csi,default is csi, if your kubernetes use flexVolume,chang it to flexVolu
me
 driver: "csi" # by default is csi, you could change to flexVolume for early version of K8s (<= 1.14.x)
 nasbasepath: "/ags-wdl-nas2"
 # mountoptions. if you use local NFS, modify it to your mount options.
 # config in storageclass.yaml 、transfer-pvc.yaml
 mountVers: "3" #mount version-svc:
 mountOptions: "nolock,tcp,noresvport"  # mount options, for local NFS server, you could remove nor
esvport option

osspvc1:
 # modify it to actual bucket name
 bucket: "oss-test-tsk"
 # modify it to actual bucket url
 url: "oss-cn-beijing.aliyuncs.com"
 # mount options. It is not changed by default.
 options: "-o max_stat_cache_size=0 -o allow_other"
```

```
 akid: "XXXXXXX"
 aksecret: "XXXXXXX"
 # absolute path of your data on OSS
 path: "/"
 ossbasepath: "/ags-wdl-oss"

osspvc2:
 bucket: "oss-test-tsk"
 url: "oss-cn-beijing.aliyuncs.com"
 options: "-o max_stat_cache_size=0 -o allow_other"
 akid: "XXXXXXX"
 aksecret: "XXXXXXX"
 path: "/input"
 ossbasepath: "/ags-wdl-oss-input"

# Relative root path of input data and output data in your wdl.json, your path should add basepath bef
ore the relative path of the mount.
# e.g.
# {
# "wf.bwa_mem_tool.reference": "/ags-wdl/reference/subset_assembly.fa.gz",#reference filename。
# "wf.bwa_mem_tool.reads_fq1": "/ags-wdl/fastq_sample/SRR1976948_1.fastq.gz",#fastq1 filename
# "wf.bwa_mem_tool.reads_fq2": "/ags-wdl/fastq_sample/SRR1976948_2.fastq.gz",#fastq2 filename
# "wf.bwa_mem_tool.outputdir": "/ags-wdl/bwatest/output", #output path,the result will output in xx
xxxxxx.cn-beijing.nas.aliyuncs.com:/mydata_root/bwatest/output,you should make sure the path exist
s.
# "wf.bwa_mem_tool.fastqFolder": "fastqfolder" #任务执行的工作目录
# }

# workdir, you can choose nasbasepath or ossbasepath as your workdir
workdir : "/ags-wdl-oss"
# provisiondir,you can choose nasbasepath or ossbasepath as your provisiondir. task will create a volu
me dynamic to input/output date.
provisiondir: "/ags-wdl-oss-input"
#Scheduling the task to the specified node. e.g. nodeselector: "node-type=wdl"
nodeselector: "node-type=wdl"
#config in cromwellserver-svc.yaml
cromwellserversvc:
 nodeport : "32567" # cromwellserver nodeport, for internal access, you could use LB instead for extern
al access to cromwell server.

#config in config.yaml
```

```
config:
 # project namespace. default is wdl，Generally, no change is required
 namespace: "wdl" # namespace
 #tesk backand domain name. default is tesk-api，Generally, no change is required
 teskserver: "tesk-api"
 #cromwellserver domain name,default is cromwellserver，Generally, no change is required
 cromwellserver: "cromwellserver"
 #cromwellserver port,default is 8000，Generally, no change is required
 cromwellport: "8000"
```

- 配置应用存储卷。

  应用存储卷包括NAS和OSS。

参数	说明	是否必须配置	配置方法
naspvcs	需要挂载的NAS存储卷	是	当需要挂载的NAS存储卷PVC为 naspvc1 、naspvc2 时，配置如下。 naspvcs: - naspvc1 - naspvc2
osspvcs	需要挂载的OSS存储卷	是	当需要挂载的OSS存储卷PVC为 osspvc1 、osspvc2 时，配置如下。 osspvcs: - osspvc1 - osspvc2

- 配置NAS存储。

  每一个NAS存储卷对应执行任务容器中的一个路径 nasbasepath 。配置 naspvcs 需要为每一个NAS存储卷进行详细的参数配置。

  > **说明** 其他参数配置保持默认即可。

参数	说明	是否必须配置	配置方法
server	NAS IP	是	输入和输出数据。需要修改为您自己NAS的url，和path。

参数	说明	是否必须配置	配置方法
path	挂载的子目录	是	
driver	Flexvolume or CSI	是	集群的存储驱动，默认为CSI。当集群使用Flexvolume时，需将该参数修改为Flexvolume。
nasbasepath	NAS目录在应用中的映射目录	是	NAS目录在应用中的映射目录，在后续输入数据和输出数据时，需要将绝对目录中的 `server:path` 替换为 `basepath` 。例如， `server` 为192.168.0.1， `path` 为*/wdl*， `basepath` 为*/ags-wdl*，输入数据test.fq.gz在NAS中的存储目录为*192.168.0.1:/wdl/test/test.fq.gz* 。当需要填写输入路径时，应填写为*/ags-wdl/test/test.fq.gz*，此时应用能够访问到输入数据。本文后面均以 `server` 为192.168.0.1， `path` 为*/wdl*， `basepath` 为*/ags-wdl*作为示例。
mountOptions	挂载选项	是	挂载参数。当使用您自己的NFS时间时，可以修改该参数进行适配。
mountVers	挂载版本	是	

- 配置OSS存储。

  每一个OSS存储卷对应执行任务容器中的一个路径 `ossbasepath` 。配置 `osspvcs` 需要为每一个NAS存储卷进行详细的参数配置。

参数	说明	是否必须配置	配置方法
bucket	oss bucket	是	需要修改为您自己OSS的bucket name和url。
url	oss url	是	

参数	说明	是否必须配置	配置方法
options	挂载参数	是	默认为 `-o max_stat_cache_size=0 -o allow_other` ，不用修改。
akid	akid	是	您的ak信息，将会以Secret的方式部署在集群中。
aksecret	aksecret	是	
path	挂载的子目录	是	需要挂载的bucket的子目录。
ossbasepath	OSS目录在应用中的映射目录	是	OSS目录在应用中的映射目录，在后续输入数据和输出数据时，需要将绝对目录中的 `bucket:path` 替换为 `ossbasepath` 。例如， `bucket` 为shenzhen-test， `path` 为*/wdl*， `ossbasepath` 为*/ags-wdl*，输入数据test.fq.gz在oss中的存储目录为*shenzhen-test:/wdl/test/test.fq.gz*。当需要填写输入路径时，应填写为*/ags-wdl/test/test.fq.gz*，此时应用能够正确的访问到输入数据。

- 配置其他参数。

参数	说明	是否必须配置	配置方法
workdir	工作目录	是	需要在配置的 `nasbasepath` 、 `ossbasepath` 中选择一个作为工作目录，任务运行过程中的中间文件（script、error等）均会生成在该目录下。

参数	说明	是否必须配置	配置方法
provisiondir	动态创建pv目录	是	需要在配置的 nasbasepath 、 ossbasepath 中选择一个作为动态创建PV的目录，任务运行过程中的输入输出数据均会生成在该目录下创建的PV中。
nodeselector	任务调度支持label	否	如果需要将任务调度到特定的label机器时，可以填写该字段。配置 node-type=wdl 会调度所有的任务到 node-type=wdl 的节点上。
nodeport	cromwellserver的服务暴露端口	是	cromwellserver的服务暴露端口。当您需要用自己的Widdler客户端提交任务时，需要用到该参数，默认为32567。
namespace	项目命名空间	是	应用部署的命名空间，默认为wdl。
teskserver	tesk服务域名	是	tesk的服务域名，默认无需更改。
cromwellserver	cromwellserver	是	cromwellserver的服务域名，默认无需更改。
cromwellport	cromwellserver端口	是	cromwellserver的服务端口，默认无需更改。

6. 部署应用。

   在**应用目录 - ack-ags-wdl**页面右侧选择集群和命名空间，设置发布名称，单击**创建**。

   > **说明**

   执行以下命令，返回的结果显示cromwellcli、cromwellserver、tesk-api三个组件正常运行，说明部署成功。

   ```
   kubectl get pods -n wdl
   ```

   系统输出类似如下结果。

```
NAME                READY  STATUS    RESTARTS  AGE
cromwellcli-85cb66b98c-bv4kt     1/1    Running   0      5d5h
cromwellserver-858cc5cc8-np2mc   1/1    Running   0      5d5h
tesk-api-5d8676d597-wtmhc        1/1    Running   0      5d5h
```

步骤二：提交任务

在集群外部可以通过AGS和命令行提交任务，推荐使用AGS提交任务。

通过AGS提交任务

新版本AGS-CLI提供了向集群提交WDL任务的功能，您只需下载AGS，配置crowellserver地址即可。下载AGS请参见AGS命令行帮助。

1. 创建*bwa.wdl*文件和*bwa.json*文件。
   - b*wa.wdl*文件示例

```
task bwa_mem_tool {
 Int threads
 Int min_seed_length
 Int min_std_max_min
 String reference
 String reads_fq1
 String reads_fq2
 String outputdir
 String fastqFolder
 command {
     mkdir -p /bwa/${fastqFolder}
     cd /bwa/${fastqFolder}
     rm -rf SRR1976948*
     wget https://ags-public.oss-cn-beijing.aliyuncs.com/alignment/subset_assembly.fa.gz
     wget https://ags-public.oss-cn-beijing.aliyuncs.com/alignment/SRR1976948_1.fastq.gz
     wget  https://ags-public.oss-cn-beijing.aliyuncs.com/alignment/SRR1976948_2.fastq.gz
     gzip -c ${reference} > subset_assembly.fa
     gunzip -c ${reads_fq1} | head -800000 > SRR1976948.1
     gunzip -c ${reads_fq2} | head -800000 > SRR1976948.2
     bwa index subset_assembly.fa
     bwa aln subset_assembly.fa SRR1976948.1 > ${outputdir}/SRR1976948.1.untrimmed.sai
     bwa aln subset_assembly.fa SRR1976948.2 > ${outputdir}/SRR1976948.2.untrimmed.sai
 }
 output {
  File sam1 = "${outputdir}/SRR1976948.1.untrimmed.sai"
  File sam2 = "${outputdir}/SRR1976948.2.untrimmed.sai"
 }
 runtime {
  docker: "registry.cn-hangzhou.aliyuncs.com/plugins/wes-tools:v3"
  memory: "2GB"
  cpu: 1
 }
}
workflow wf {
 call bwa_mem_tool
}
```

- *bwa.json*文件示例

```
{
"wf.bwa_mem_tool.reference": "subset_assembly.fa.gz",#reference filename。
"wf.bwa_mem_tool.reads_fq1": "SRR1976948_1.fastq.gz",#fastq1 filename
"wf.bwa_mem_tool.reads_fq2": "SRR1976948_2.fastq.gz",#fastq2 filename
"wf.bwa_mem_tool.outputdir": "/ags-wdl/bwatest/output", #output path,the result will output in 1
92.168.0.1:/wdl/bwatest/output,you should make sure the path exists.
"wf.bwa_mem_tool.fastqFolder": "fastqfolder" #任务执行的工作目录
}
```

2. 将访问集群中的32567端口的流量转发到service cromwellserver上。

```
kubectl port-forward svc/cromwellserver 32567:32567
```

3. 配置cromwellserver地址。

   输入以下命令。

```
ags config init
```

   系统输出类似如下结果。

```
Please input your AccessKeyID
xxxxx
Please input your AccessKeySecret
xxxxx
Please input your cromwellserver url
192.168.0.1:32567
```

4. 在本地提交WDL任务。

   输入以下命令。

```
ags wdl run resource/bwa.wdl resource/bwa.json --watch #watch参数可以让该命令进行一个同步等待，直到该任务成功或失败。
```

   系统输出类似如下结果。

```
INFO[0000] bd747360-f82c-4cd2-94e0-b549d775f1c7 Submitted
```

5. （可选）您还可以在本地进行查询、删除WDL任务。
   - 查询WDL任务。

- 通过explain命令查询WDL任务。

  输入以下命令。

  ```
  ags wdl explain bd747360-f82c-4cd2-94e0-b549d775f1c7
  ```

  系统输出如下结果。

  ```
  INFO[0000] bd747360-f82c-4cd2-94e0-b549d775f1c7 Running
  ```

- 通过query命令查询WDL任务。

  输入以下命令。

  ```
  ags wdl query bd747360-f82c-4cd2-94e0-b549d775f1c7
  ```

  系统输出如下结果。

  ```
  INFO[0000] bd747360-f82c-4cd2-94e0-b549d775f1c7 {"calls":{"end":"0001-01-01T00:00:00.000Z","executionStatus":null,"inputs":null,"start":"0001-01-01T00:00:00.000Z"},"end":"0001-01-01T00:00:00.000Z","id":"b3aa1563-6278-4b2e-b525-a2ccddcbb785","inputs":{"wf_WGS.Reads":"/ags-wdl-nas/c.tar.gz"},"outputs":{},"start":"2020-10-10T09:34:56.022Z","status":"Running","submission":"2020-10-10T09:34:49.989Z"}
  ```

- 删除WDL任务。

  输入以下命令。

  ```
  ags wdl abort bd747360-f82c-4cd2-94e0-b549d775f1c7
  ```

  系统输出如下结果。

  ```
  INFO[0000] bd747360-f82c-4cd2-94e0-b549d775f1c7 Aborting
  ```

通过命令行提交任务

您需要编写WDL文件和输入的JSON文件，通过提供的镜像提交任务，其中cromwellserver的地址为集群IP:nodeport。

1. 创建*wa.wdl*文件和*bwa.json*文件。
   - *bwa.wdl*文件示例

```
task bwa_mem_tool {
 Int threads
 Int min_seed_length
 Int min_std_max_min
 String reference
 String reads_fq1
 String reads_fq2
 String outputdir
 String fastqFolder
 command {
     mkdir -p /bwa/${fastqFolder}
     cd /bwa/${fastqFolder}
     rm -rf SRR1976948*
     wget https://ags-public.oss-cn-beijing.aliyuncs.com/alignment/subset_assembly.fa.gz
     wget https://ags-public.oss-cn-beijing.aliyuncs.com/alignment/SRR1976948_1.fastq.gz
     wget  https://ags-public.oss-cn-beijing.aliyuncs.com/alignment/SRR1976948_2.fastq.gz
     gzip -c ${reference} > subset_assembly.fa
     gunzip -c ${reads_fq1} | head -800000 > SRR1976948.1
     gunzip -c ${reads_fq2} | head -800000 > SRR1976948.2
     bwa index subset_assembly.fa
     bwa aln subset_assembly.fa SRR1976948.1 > ${outputdir}/SRR1976948.1.untrimmed.sai
     bwa aln subset_assembly.fa SRR1976948.2 > ${outputdir}/SRR1976948.2.untrimmed.sai
 }
 output {
  File sam1 = "${outputdir}/SRR1976948.1.untrimmed.sai"
  File sam2 = "${outputdir}/SRR1976948.2.untrimmed.sai"
 }
 runtime {
  docker: "registry.cn-hangzhou.aliyuncs.com/plugins/wes-tools:v3"
  memory: "2GB"
  cpu: 1
 }
}
workflow wf {
 call bwa_mem_tool
}
```

- *bwa.json*文件示例

```
{
"wf.bwa_mem_tool.reference": "subset_assembly.fa.gz",#reference filename。
"wf.bwa_mem_tool.reads_fq1": "SRR1976948_1.fastq.gz",#fastq1 filename
"wf.bwa_mem_tool.reads_fq2": "SRR1976948_2.fastq.gz",#fastq2 filename
"wf.bwa_mem_tool.outputdir": "/ags-wdl/bwatest/output", #output path,the result will output in 1
92.168.0.1:/wdl/bwatest/output,you should make sure the path exists.
"wf.bwa_mem_tool.fastqFolder": "fastqfolder" #任务执行的工作目录
}
```

2. 提交WDL任务。

   输入以下命令。

   ```
   docker run -e CROMWELL_SERVER=192.16*.*.** -e CROMWELL_PORT=30384 registry.cn-beijing.aliyuncs.
   com/tes-wes/cromwellcli:v1 run resources/bwa.wdl resources/bwa.json
   ```

   系统输出类似如下结果。

   ```
   ----------Cromwell Links----------
   http://192.16*.*.**:30384/api/workflows/v1/5d7ffc57-6883-4658-adab-3f508826322a/metadata
   http://192.16*.*.**:30384/api/workflows/v1/5d7ffc57-6883-4658-adab-3f508826322a/timing
   {
     "status": "Submitted",
     "id": "5d7ffc57-6883-4658-adab-3f508826322a"
   }
   ```

6.2. WDL帮助示例

本文介绍WDL工作流的帮助示例。

查询任务列表

输入以下命令，查询WDL工作流的任务列表。

```
ags wdl list
```

预期输出：

```
+----------+--------------------------+----------------------------------+------------+
| JOB NAME |     CREATE TIME      |          JOB ID          | JOB STATUS |
+----------+--------------------------+----------------------------------+------------+
| wf     | 2020-10-21T06:34:10.859Z | 87546541-f41d-4745-9d1e-4b9b0f27b033 | Running   |
| wf     | 2020-10-21T05:52:28.360Z | cc195aee-41c7-44b0-89e5-4e94ba47f56b | Succeeded |
| wf     | 2020-10-21T05:52:08.340Z | 7209675e-4277-4d68-8848-725abca6b143 | Succeeded |
| wf     | 2020-10-21T03:50:01.010Z | 05ed1979-39d4-41db-8535-446919088a2b | Succeeded |
| wf     | 2020-10-21T03:45:40.783Z | b2394c34-9086-441f-af49-02735b5710ed | Succeeded |
+----------+--------------------------+----------------------------------+------------+
```

获取任务详细信息

输入以下命令，获取任务详细信息。

> **说明** 您可以在任务详细信息中获取JOB ID、错误日志保存路径等信息。

```
ags wdl query cc195aee-41c7-44b0-89e5-4e94ba47f56b
```

预期输出：

```
{
  "workflowName": "wf",
  "workflowProcessingEvents": [
    {
      "description": "Finished",
      "timestamp": "2020-10-21T05:53:05.109Z",
      "cromwellId": "cromid-89e8e7d",
      "cromwellVersion": "54-36be57c-SNAP"
    },
    {
      "timestamp": "2020-10-21T05:52:28.359Z",
      "cromwellVersion": "54-36be57c-SNAP",
      "cromwellId": "cromid-89e8e7d",
      "description": "PickedUp"
    }
  ],
  "metadataSource": "Unarchived",
  "actualWorkflowLanguageVersion": "draft-2",
  "submittedFiles": {
    "workflow": "task bwa_mem_tool {\n  String outputdir\n  String fastqFolder\n  String cpunum\n  Stri
ng bamfilename\n  command {\n    cd ${fastqFolder}\n    cp gene.bam.bai gene.bam.bai.test\n  }\n  runti
me {\n    docker: \"registry.cn-beijing.aliyuncs.com/shuangkun/genetools:v1.1\"\n    memory: \"2GB\"\n    cpu:
```

me {\n docker: \"registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1\"\n memory: \"2GB\"\n cpu: 1\n }\n }\ntask agstask {\n String config\n command {\n mkdir /root/.ags\n cp ${config} /root/.ags/\n ags remote list > /ags-wdl-nas/remote.txt\n }\n runtime {\n docker: \"registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1\"\n memory: \"2GB\"\n cpu: 1\n }\n }\nworkflow wf {\n call agstask\n call bwa_mem_tool\n} \n",
"root": "",
"options": "{\n\n}",
"inputs": "{\"wf.agstask.config\":\"/ags-wdl-nas/bwatest/ags/config\",\"wf.bwa_mem_tool.bamfilename\":\"gene.bam\",\"wf.bwa_mem_tool.cpunum\":\"6\",\"wf.bwa_mem_tool.fastqFolder\":\"/ags-wdl-nas/sample/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz\",\"wf.bwa_mem_tool.outputdir\":\"/ags-wdl-nas/bwatest/output\"}",
"workflowUrl": "",
"labels": "{}"
},
"calls": {
"wf.bwa_mem_tool": [
{
"executionStatus": "Done",
"stdout": "/ags-wdl-nas/wf/cc195aee-41c7-44b0-89e5-4e94ba47f56b/call-bwa_mem_tool/execution/stdout",
"backendStatus": "Complete",
"commandLine": "cd /ags-wdl-nas/sample/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz\ncp gene.bam.bai gene.bam.bai.test",
"shardIndex": -1,
"outputs": {},
"runtimeAttributes": {
"preemptible": "false",
"failOnStderr": "false",
"continueOnReturnCode": "0",
"docker": "registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1",
"maxRetries": "0",
"cpu": "1",
"memory": "2 GB"
},
"callCaching": {
"allowResultReuse": false,
"effectiveCallCachingMode": "CallCachingOff"
},
"inputs": {
"fastqFolder": "/ags-wdl-nas/sample/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz",
"outputdir": "/ags-wdl-nas/bwatest/output",

```
        "cpunum": "6",
        "bamfilename": "gene.bam"
      },
      "returnCode": 0,
      "jobId": "task-6dbeff7e",
      "backend": "TESK",
      "end": "2020-10-21T05:53:04.317Z",
      "dockerImageUsed": "registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1",
      "stderr": "/ags-wdl-nas/wf/cc195aee-41c7-44b0-89e5-4e94ba47f56b/call-bwa_mem_tool/execution/stderr",
      "callRoot": "/ags-wdl-nas/wf/cc195aee-41c7-44b0-89e5-4e94ba47f56b/call-bwa_mem_tool",
      "attempt": 1,
      "executionEvents": [
        {
          "startTime": "2020-10-21T05:53:03.351Z",
          "description": "UpdatingJobStore",
          "endTime": "2020-10-21T05:53:04.317Z"
        },
        {
          "startTime": "2020-10-21T05:52:29.408Z",
          "endTime": "2020-10-21T05:52:29.408Z",
          "description": "Pending"
        },
        {
          "startTime": "2020-10-21T05:52:29.970Z",
          "endTime": "2020-10-21T05:53:03.351Z",
          "description": "RunningJob"
        },
        {
          "endTime": "2020-10-21T05:52:29.957Z",
          "startTime": "2020-10-21T05:52:29.408Z",
          "description": "RequestingExecutionToken"
        },
        {
          "description": "WaitingForValueStore",
          "startTime": "2020-10-21T05:52:29.957Z",
          "endTime": "2020-10-21T05:52:29.957Z"
        },
        {
          "startTime": "2020-10-21T05:52:29.957Z",
          "endTime": "2020-10-21T05:52:29.970Z",
```

```
            "description": "PreparingJob"
          }
        ],
        "start": "2020-10-21T05:52:29.408Z"
      }
    ],
    "wf.agstask": [
      {
        "executionStatus": "Done",
        "stdout": "/ags-wdl-nas/wf/cc195aee-41c7-44b0-89e5-4e94ba47f56b/call-agstask/execution/stdout",
        "backendStatus": "Complete",
        "commandLine": "mkdir /root/.ags\ncp /ags-wdl-nas/bwatest/ags/config /root/.ags/\nags remote list
> /ags-wdl-nas/remote.txt",
        "shardIndex": -1,
        "outputs": {},
        "runtimeAttributes": {
          "preemptible": "false",
          "failOnStderr": "false",
          "continueOnReturnCode": "0",
          "docker": "registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1",
          "maxRetries": "0",
          "cpu": "1",
          "memory": "2 GB"
        },
        "callCaching": {
          "allowResultReuse": false,
          "effectiveCallCachingMode": "CallCachingOff"
        },
        "inputs": {
          "config": "/ags-wdl-nas/bwatest/ags/config"
        },
        "returnCode": 0,
        "jobId": "task-1aa59269",
        "backend": "TESK",
        "end": "2020-10-21T05:53:03.297Z",
        "dockerImageUsed": "registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1",
        "stderr": "/ags-wdl-nas/wf/cc195aee-41c7-44b0-89e5-4e94ba47f56b/call-agstask/execution/stderr",
        "callRoot": "/ags-wdl-nas/wf/cc195aee-41c7-44b0-89e5-4e94ba47f56b/call-agstask",
        "attempt": 1,
        "executionEvents": [
```

```
        {
          "startTime": "2020-10-21T05:52:29.965Z",
          "description": "RunningJob",
          "endTime": "2020-10-21T05:53:02.773Z"
        },
        {
          "description": "PreparingJob",
          "startTime": "2020-10-21T05:52:29.957Z",
          "endTime": "2020-10-21T05:52:29.965Z"
        },
        {
          "description": "WaitingForValueStore",
          "startTime": "2020-10-21T05:52:29.957Z",
          "endTime": "2020-10-21T05:52:29.957Z"
        },
        {
          "startTime": "2020-10-21T05:52:29.408Z",
          "endTime": "2020-10-21T05:52:29.957Z",
          "description": "RequestingExecutionToken"
        },
        {
          "startTime": "2020-10-21T05:52:29.408Z",
          "description": "Pending",
          "endTime": "2020-10-21T05:52:29.408Z"
        },
        {
          "endTime": "2020-10-21T05:53:03.297Z",
          "startTime": "2020-10-21T05:53:02.773Z",
          "description": "UpdatingJobStore"
        }
      ],
      "start": "2020-10-21T05:52:29.408Z"
    }
  ]
},
"outputs": {},
"workflowRoot": "/ags-wdl-nas/wf/cc195aee-41c7-44b0-89e5-4e94ba47f56b",
"actualWorkflowLanguage": "WDL",
"id": "cc195aee-41c7-44b0-89e5-4e94ba47f56b",
"inputs": {
  "wf.agstask.config": "/ags-wdl-nas/bwatest/ags/config",
```

```
    "wf.bwa_mem_tool.fastqFolder": "/ags-wdl-nas/sample/MGISEQ2000_PCR-free_NA12878_1_V100003043
_L01_1.fq.gz",
    "wf.bwa_mem_tool.outputdir": "/ags-wdl-nas/bwatest/output",
    "wf.bwa_mem_tool.cpunum": "6",
    "wf.bwa_mem_tool.bamfilename": "gene.bam"
  },
  "labels": {
    "cromwell-workflow-id": "cromwell-cc195aee-41c7-44b0-89e5-4e94ba47f56b"
  },
  "submission": "2020-10-21T05:52:13.812Z",
  "status": "Succeeded",
  "end": "2020-10-21T05:53:05.109Z",
  "start": "2020-10-21T05:52:28.360Z"
}
```


挂载PV

1. 登录容器服务管理控制台。
2. 在控制台左侧导航栏中，单击**集群**。
3. 在**集群列表**页面中，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
4. 在集群管理页左侧导航栏中单击**发布**。
5. 在**发布**页面单击**Helm**页签，单击ack-ags-wdl**操作**列的**更新**。
6. 在**更新发布**对话框的文本框中挂载PV，以下以naspvc3为例。
   i. 添加PV名称。在**更新发布**对话框的文本框中找到 `naspvcs` 字段，在 `naspvcs` 字段下增加所要添加的PV名称。

   ```
   naspvcs:
   - naspvc1
   - naspvc2
   - naspvc3
   ```

   ii. 添加PV配置内容。在**更新发布**对话框的文本框添加PV配置内容，然后单击**更新**。

   ```
   naspvc3:
    driver: csi
    mountOptions: nolock,tcp,noresvport
    mountVers: "3"
    nasbasepath: /ags-wdl-nas3
    path: /tarTest/sample
    server: xxxxxxxx.cn-beijing.nas.aliyuncs.com
   ```

   完成挂载PV后，后续创建的任务可以读取或写入新挂载的PV。

6.3. 通过AGS服务调用加速工作流

通过本地的WDL工作流和远程的AGS工作流组成的混合工作流，可以将耗时长、计算量大、标准流程的任务放在AGS服务端中执行。将耗时短、所需资源有限、需要自行定制流程的任务放在本地的WDL工作流中执行，数据通过OSS进行中间流转，从而有效的提高任务执行效率、节省资源成本。本文将通过RemoteApi实现Mapping过程为例，演示如何编写并运行一个AGS混合工作流。

前提条件

- 开通服务
- 已在应用目录中安装ack-ags-wdl，详细介绍请参见创建WDL工作流的步骤一：部署应用。

操作步骤

1. 配置AGS。
   i. 下载和安装AGS，详细介绍请参见AGS命令行帮助。

   ```
   ags config init
   ```

   配置AGS完成后，会自动生成AGS配置文件*config*。
   ii. 将AGS配置文件*config*保存在*/ags-wdl-nas/bwatest/ags/config*路径下。
2. 创建*agsHybrid.wd*文件。

```
task agstask {
  String method
  String region
  String file1
  String file2
  String bucket
  String outputbam
  String ref
  String service
  String config
  command {
    mkdir /root/.ags
    cp ${config} /root/.ags/
    ags remote run ${method} --region ${region} --fastq1 ${file1} --fastq2 ${file2} --bucket ${bucket} --out
put-bam ${outputbam} --reference ${ref} --service ${service} --watch
  }
  runtime {
  docker: "registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1"
  memory: "20GB"
  cpu: 6
  }
 }
```

```
task downloaddata {
 String osspath
 String naspath
 command {
   cp ${osspath}/gene.bam ${naspath}
 }
 runtime {
 docker: "ubuntu"
 memory: "2GB"
 cpu: 1
 }
}

task bwa_mem_tool {
 String outputdir
 String fastqFolder
 String cpunum
 String bamfilename
 command {
   cd ${fastqFolder}
   samtools index -@ ${cpunum} ${bamfilename}
 }
 runtime {
 docker: "registry.cn-beijing.aliyuncs.com/shuangkun/genetool:v1.1"
 memory: "20GB"
 cpu: 6
 }
}

task uploaddata {
 String osspath
 String naspath
 command {
   cp ${naspath}/gene.bam.bai ${osspath}
 }
 runtime {
 docker: "ubuntu"
 memory: "2GB"
 cpu: 1
 }
}
```

```
workflow wf {
  call agstask
  call downloaddata
  call bwa_mem_tool
  call uploaddata
}
```

3. 创建*agsHybrid.json*文件。

   以下文件中的*/ags-wdl-oss/*、*/ags-wdl-nas/*需要设置为部署ack-ags-wdl的Bucket和NAS的路径。

```
{
  "wf.agstask.config": "/ags-wdl-nas/bwatest/ags/config",
  "wf.agstask.method": "mapping",
  "wf.agstask.region": "shenzhen",
  "wf.agstask.file1": "sample/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz",
  "wf.agstask.file2": "sample/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_2.fq.gz",
  "wf.agstask.bucket": "my-test-shenzhen",
  "wf.agstask.outputbam": "output/bam/gene.bam",
  "wf.agstask.ref": "hg19",
  "wf.agstask.service": "s",
  "wf.downloaddata.osspath": "/ags-wdl-oss/output/bam",
  "wf.downloaddata.naspath": "/ags-wdl-nas/sample",
  "wf.bwa_mem_tool.cpunum": "6",#cpu num
  "wf.bwa_mem_tool.bamfilename": "gene.bam",
  "wf.bwa_mem_tool.outputdir": "/ags-wdl-nas/bwatest/output", #output path,the result will output in 192.168.0.1:/wdl/bwatest/output,you should make sure the path exists.
  "wf.bwa_mem_tool.fastqFolder": "/ags-wdl-nas/sample/MGISEQ2000_PCR-free_NA12878_1_V100003043_L01_1.fq.gz", #任务执行的工作目录
  "wf.uploaddata.naspath": "/ags-wdl-nas/sample",
  "wf.uploaddata.osspath": "/ags-wdl-oss/output/bam",
  }
```

4. 创建混合工作流。

```
ags wdl run agsHybrid.wdl agsHybrid.json
```

   创建混合工作流成功后，会自动执行混合工作流。

   找到ags-wdl-oss对应Bucket，进入Bucket下的*output/bam*，可以看到*gene.bam.bai*文件。说明混合工作流创建并执行成功。