基因容器

GCS 流程语法参考

文档版本 01

发布日期 2022-03-30

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 流程语法说明

1.1 语法说明

基因容器语法，是GCS引以为傲的特性之一。使用基因容器语法编写的流程描述文件是基因容器服务流程控制的唯一方式。如已有其他流程语法文件，需要先转换为基因容器语法，然后才能由基因容器执行。流程描述文件的生成方式有多种，其一基于基因容器语法手动编写；其二是通过基因容器命令行自动生成；其三是使用流程设计器辅助编写；最后一种是通过流程迁移工具转换生成。当前仅支持前三种。

基因容器的流程描述语言支持YAML，YAML语法规则请参见**YAML语法**。为确保写作正确的基因容器语法，请先了解YAML语法规则。

为了简化语法，基因容器为流程语法提供了内置函数，提升GCS语法描述流程的能力。当前已支持的流程函数包括：

- **range**：用于任务并发时，迭代变量数量过多时，简化迭代变量范围。
- **get_result**：用于获取“指定步骤”的标准输出，生成控制当前任务并发的迭代变量范围。
- **check_result**：用于判断“指定步骤”的结果，是否等于预期值，从而决定是否执行当前步骤。

模板结构

基因容器语法结构请参见**表1-1**。

表 1-1 模板组成元素

参数	是否必选	参数类型	参数描述
version	是	string	流程文件的版本。当前支持为**genecontainer_0_1**。
inputs	否	Map	整个流程的可变量。可以定义多个，在执行的时候设置这些变量的实际值。

参数	是否必选	参数类型	参数描述
workflow	是	Map	定义完整流程包含的各个步骤，以及步骤之间的依赖关系。
volumes	否	Map	定义任务挂载共享存储的挂载路径。
outputs	否	Map	定义流程最终生成的文件列表。用于界面展示使用。

模板示例

```
version: genecontainer_0_1
inputs: #设置流程变量
  sample: #变量名
    default: sample1
    type: string
    description: 变量描述，支持中文
    label: basic
workflow: #设置流程顺序
  test-job-a:
    tool: busybox:latest
    type: GCS.Job
    resources:
      memory: 4G
    commands_iter: # {item} 为GCS内置变量，表示并发任务的 index，需要的时候选用
      command: sleep 10; touch /obs/test-job/${sample}.${item}.${1}.txt
      vars_iter:
        - [0,1]
  test-job-b:
    tool: busybox:latest
    type: GCS.Job
    resources:
      memory: 4G
    commands:
      - sleep 10; touch /obs/test-job/${sample}.job-b.txt
    depends:
      - target: test-job-a #表示依赖另一个步骤，默认依赖类型: whole
volumes: #设置挂载共享存储
  genobs:
    mount_path: /obs
    mount_from:
      pvc: '${GCS_DATA_PVC}' #OBS数据桶
outputs: #设置最终结果
  result-txt: #结果文件
    paths_iter:
      path:  /obs/test-job/${sample}.${item}.${1}.txt
      vars_iter:
        - [0,1]
```

1.2 version

流程文件的版本。当前支持为 **genecontainer_0_1**。

version 格式

```
version: genecontainer_0_1
```


1.3 inputs

可选项，用于定义测序流程的可变部分。inputs由多个变量组成，最多可定义60个变量，且每个变量名唯一，若变量名重复，则后面定义的将会覆盖之前定义的。

变量可以在流程文件的其他参数中引用，引用方式采用shell脚本方式，即：**${var}**，其中var为变量名称。

inputs 格式

```
inputs:
  <输入参数名称>:
    type: <类型>
    default: <默认值>
    description: <描述，支持中文>
    label: <标签>
```

表 1-2 参数属性说明

属性	是否必选	参数类型	参数描述
输入参数名称	是	String	输入参数名称，由字母、数字和中划线“-”或下划线“_”组成，长度为[1, 20]。
type	是	String	参数类型，请根据参数类型选择，可选范围如下所示。 ● string，类型为字符串 ● number，类型为数字 ● bool，类型为布尔值 ● array，类型为数组
description	否	String	参数描述信息，支持中文，长度为[0, 255]。
default	否	String	参数默认值，按照type填写对应的值。默认值信息可在执行流程时由外部输入替换；若未填写默认值，外部输入将必须填写这个参数的值。 一个变量可以引用另一个变量的值，如： `default: /home/${path_var}` **须知** 默认值的类型和定义的参数类型必须统一。

属性	是否必选	参数类型	参数描述
label	否	String	参数的标签，用于参数分组，便于管理，以便其他使用者理解。 长度为[0, 64]，支持中文。 取值如下所示。 • basic，表示基础配置，显示在控制台“配置”>“通用配置”中。当此参数不设label时，**默认配置值为basic**。 • 其他值，表示分组名称，显示在控制台“配置”>“高级配置”中。 此处定义的标签可在控制台界面执行时进行分类展示，如下图中的“group_info”即为标签。 

inputs 配置样例

```
inputs:
  sample:
    type: string
    default: /home/zhangsan
  split:
    default: 3
    type: number
    label: gatk #为参数分组
```

1.4 workflow

必选项，用于定义流程包含的各个步骤，以及步骤之间的依赖关系。一个workflow由多个步骤组成，每个步骤可以执行特定的任务，比如运行容器、执行FPGA程序，或是调用Spark服务加速等任意的华为云服务。当前支持的类型为运行容器，即：**GCS.Job**。未来会支持其他类型的任务。

workflow 格式

```
workflow:
  <任务名称>:
    description: <当前步骤的描述>
    type: <任务类型>
    tool: <当前任务运行的工具+版本>
    resources: <执行任务需要的资源>
```

```
commands: <容器中执行的命令，直接给出方式>
commands_iter: <容器中执行的命令，变量迭代方式>
condition: <当前步骤是否执行的条件>
depends: <依赖的任务>
```

表 1-3 GCS.Job 参数属性说明

属性	是否必选	参数类型	参数描述
任务名称	是	String	基因容器具体步骤的任务名称。 小写字母打头，由小写字母数字中划线组成，长度为[1, 40]。
description	否	String	当前步骤的描述信息，支持中文，长度为[0, 255]
type	是	String	GCS支持的任务类型，当前仅支持运行容器类，此处填写“GCS.Job”。
tool	是	String	当前任务运行的工具及版本，格式为“tool_name: tool_version”。例如，0.7.12版本的bwa设置为“tool: bwa:0.7.12”。 **须知** 指定的工具必须存在。
resources	否	Struct	执行任务需要的资源，参数说明参见**表2 resources属性说明**。

属性	是否必选	参数类型	参数描述

commands	否	array[string]	容器中执行的命令，数组长度表示并发数量。每个成员表示一个容器中执行的命令。
			提示：**可以是任意的Linux命令**。如：
			a ）直接写要执行的命令，如： echo helloworld; touch sample.txt
			b ）把要执行的命令放入待挂载卷中，xx.sh文件，这里直接调用，如：sh /path/to/xx.sh
			并发示例：
			如下示例中，commands有四行，则表示容器并发数量为4，每个容器分别执行不同的命令
			`commands:`
			`  - sh /obs/gcscli/run-xxx/run.sh 1 a`
			`  - sh /obs/gcscli/run-xxx/run.sh 2 a`
			`  - sh /obs/gcscli/run-xxx/run.sh 1 b`
			`  - sh /obs/gcscli/run-xxx/run.sh 2 b`
			提示：
			如果命令行是**由多行组成**，可以使用yaml语法中的“|”（保留换行符，整个字符串当做yaml中一个key的value）格式。这样就可以把大篇幅的命令行原封不动的拷贝过来，如：
			`commands:`
			`  - |`
			`    samtools merge -f -@ ${nthread} -b ${volume-path-sfs}/${sample}/mergelist.txt \`
			`    ${volume-path-sfs}/${sample}/${sample}.sort.bam && \`
			`    samtools flagstat ${volume-path-sfs}/${sample}/${sample}.sort.bam > ${volume-path-sfs}/${sample}/${sample}.sort.flagstat`
			详见： **yaml语法**。
			须知
			给定的command默认会使用/bin/sh解释器执行，如果需要使用其他解释器如/bin/bash执行脚本，可以通过在命令行前添加解释器来控制，例如：
			`commands:`
			`  - bash /obs/gcscli/run-xxx/run.sh 1 a #方式1`
			`  - bash -c "echo helloworld"  #方式2`
			说明
			此参数和commands_iter二选一。如果容器中执行的命令需要带变量，可以使用commands_iter，正常则使用commands。

commands_iter	否	Struct	容器中执行的命令，和commands区别为，commands_iter支持带变量的shell脚本。数组行数表示并发的数量，参数说明请参见**表3 commands_iter属性说明**。

属性	是否必选	参数类型	参数描述
condition	否	bool	某个步骤是否执行的条件，用于流程条件分支控制。可以是引用**inputs变量**，也可以根据上一步的执行结果来判断当前步骤是否执行。详见 **条件分支控制**。
depends	否	array[Struct]	指定当前任务所依赖的其他任务，数组格式，参数说明请参见**表4 depends属性说明**。

表 1-4 resources 属性说明

属性	是否必选	参数类型	参数描述
memory	否	String	所需内存资源大小，单位为G。此处填写“数字+单位”。 ● 数字支持小数。 ● 单位为G或是g。 例如，需要内存大小为4G，则此处可填写为“4G”或是“4g”。 resources: memory: 4G **须知** 请确保请求内存量，小于容器集群中最大节点剩余内存大小。
cpu	否	String	所需CPU核数，单位为C。此处填写“数字+单位”。 ● 数字支持小数。 ● 单位为C或是c。 例如，所需CPU核数为0.5核，则此处可填写为 "0.5c" 或者 "0.5C"。 resources: cpu: 0.5c **须知** 请确保请求CPU核数，小于容器集群中最大节点剩余核数大小。
gpu	否	Number	所需GPU卡数量，仅数字，无单位。此处填写 "数字"。 ● 数字支持小数。（注：一般GPU推荐整数） 例如，需要1个GPU显卡，则此处可填写为 "1"。 resources: gpu: 1 **须知** 请确保请求GPU卡数，小于容器集群中最大GPU节点剩余数大小。

属性	是否必选	参数类型	参数描述
options	否	Struct	gpu的配置参数，参数说明请参见表1-5。 例如： options: gpu-type: nvidia.com/gpu-tesla-v100-16GB gpu-driver: gpu-418.126 **须知** 该参数只有在CCI环境中才生效。

表 1-5 gpu 的配置参数

属性	是否必选	参数类型	取值约束
gpu-type	否	String	gpu的类型，当前支持3种类型。 • nvidia.com/gpu-tesla-v100-16GB：NVIDIA Tesla V100 16G显卡 • nvidia.com/gpu-tesla-v100-32GB：NVIDIA Tesla V100 32G显卡 • nvidia.com/gpu-tesla-t4：NVIDIA Tesla T4显卡 详细说明请参见使用GPU。
gpu-driver	否	String	gpu驱动的版本。当前支持如下两个版本。 • gpu-460.106：460.106版本，配套CUDA 11.2.2 Update 2 及以下CUDA Toolkit版本。 • gpu-418.126：418.126版本，配套CUDA 10.1 (10.1.105)及以下CUDA Toolkit版本。 详细说明请参见使用GPU。

表 1-6 commands_iter 属性说明

属性	是否必选	参数类型	取值约束
command	是	String	带变量的任意Linux脚本，具体可以参考**commands属性说明**，变量由如下两种获取方式： ● 从vars或者vars_iter传递过来的变量，引用方式与shell一致，如“${1}”表示取第一个参数，取第二个参数则用“${2}”，依次递增。其中vars和vars_iter参数，两个二选一填写。vars参数需要列出所有组合，vars_iter参数为自动遍历组合（详见该字段描述）。 ● 基因容器提供的**内置变量“${item}”**，表示容器并发执行时，当前执行到了第几个容器，该计数从零开始。例如并发4个容器时，command命令为“echo ${item}”，则将按容器执行顺序，分别打印0，1，2，3。

属性	是否必选	参数类型	取值约束

vars	否	array[array]	二维数组，做为command代入变量，表示手动列出所有组合。
			说明
			vars和vars_iter二选一。如果变量变化有固定规则，推荐使用vars_iter，可自动迭代变量，不用手动列出该变量的所有组合。
			● 二维数组中，每一行的成员代表command命令中的变量${1}，${2}，${3}等。${1}即代表各行的第一个成员，${2}代表各行第二个成员，${3}代表各行第三个成员。
			● 二维数组中，有几行表示command命令迭代执行几次，同时也代表容器并发数量。在command命令执行过程中，先代入数组第一行，再按次序代入第二行，第三行等，直到数组代入完成。数组的行数即容器并发数量。
			例如，如下示例中，vars有四行，则表示容器并发数量为4。
			`command: echo ${1} ${2} ${item}`
			`vars:`
			`- [0, 0] # 每行代表一个并发。成员按顺序分别传入${1}和${2}...`
			`- [0, 1] # 这个并发中，0 -> ${1}; 1 -> ${2}`
			`- [1, 0] # 这个并发中，1 -> ${1}; 0 -> ${2}`
			`- [1, 1] # 这个并发中，1 -> ${1}; 1 -> ${2}`
			执行后，
			启动4个容器，
			第一个容器代入数组第一行，执行：
			`echo 0 0 0`
			第二个容器代入数组第二行，执行：
			`echo 0 1 1`
			第三个容器代入数组第三行，执行：
			`echo 1 0 2`
			第四个容器代入数组第四行，执行：
			`echo 1 1 3`

属性	是否必选	参数类型	取值约束

vars_iter	否	array[array]	二维数组，做为command代入变量，表示自动遍历组合。
			数组的第一行成员遍历代入command命令中的变量${1}，第二行成员遍历代入命令中的变量${2}，以此类推。
			示例：如下示例表示4个容器并发，执行效果与vars示例相同。
			`command: echo ${1} ${2}`
			`vars_iter:`
			`  - [0, 1]  # 第一行表示${1} 的迭代范围`
			`  - [0, 1]  # 第二行表示${2} 的迭代范围`
			如果某一行的数组成员很多，可以使用**range函数**。range(1,4)表示数组[1, 2, 3]。前闭区间，后开区间，例如：
			`vars_iter:`
			`  - range(0, 4)`
			等同于：
			`vars_iter:`
			`  - [0, 1, 2, 3]`
			提示：迭代范围可以使用输入变量，见**inputs**，如：
			`inputs:`
			`  samples:  # <=== 定义数组类型的变量`
			`    type: array`
			`    default: [a, b, c, d]`
			`workflow:`
			`  job-a:`
			`    command: echo ${1}`
			`    vars_iter:`
			`      - ${samples}  # <=== 使用此数组作为迭代范围`

表 1-7 depends 属性说明

属性	是否必选	参数类型	取值约束
target	是	String	所依赖的任务名，请确保指定的任务名必需存在。

属性	是否必选	参数类型	取值约束
type	否	String	依赖方式，可取值为： ● whole，整体依赖，**此为默认值**。 ● iterate，迭代式依赖。 例如，如果步骤一和步骤二都并发执行100个容器。 ● 设置whole则表示步骤一全部执行结束后，才能开始执行步骤二。 ● 设置iterate则表示，步骤一的1号执行完成，就可以开始执行步骤二的1号，并行执行，可以提高整体的并发效率。

workflow 配置样例

```
workflow:
  job-a:
    tool: nginx:latest
    type: GCS.Job
    resources:
      memory: 2G
      cpu: 1c
    commands:
      - sleep `expr 3 \* ${wait-base}`; echo ${output-prefix}job-a | tee -a ${obs}/${output}/${result};
  job-b:
    tool: nginx:latest
    type: GCS.Job
    commands_iter:
      command: sleep `expr ${1} \* ${wait-base}`; echo ${output-prefix}job-b-${item} | tee -a ${obs}/$
{output}/${result};
      vars_iter:
        - range(0, 4) #内置函数range，实际表示 [0, 1, 2, 3]。最终并发执行4个容器，每个使用数组成员替代
    depends:
      - target: job-a
        type: whole #可不写，whole为默认值
  job-c:
    tool: nginx:latest
    type: GCS.Job
    resources:
      memory: 8G
      cpu: 2c
    commands_iter:
      command: sleep `expr ${1} \* ${wait-base}`; echo ${output-prefix}job-c-${item} | tee -a ${obs}/$
{output}/${result};
      vars_iter:
        - [3, 20] #最终并发2个容器，每个使用数组成员替代
    depends:
      - target: job-a
        type: iterate
      - target: job-b
```

1.5 volumes

可选项，用于定义各个任务挂载哪些共享存储，以及挂载的路径。任务可以挂载多块共享存储。

volumes 格式

```
volumes:
  <共享卷名称>:
    mount_path: <挂载到容器中的路径>
    mount_from: <共享存储的详细信息>
```

表 1-8 参数属性说明

属性	是否必选	参数类型	取值约束
mount_path	是	String	将共享存储挂载到容器中的路径，可填写任意路径，如果指定路径已存在则会覆盖。 **说明** 请确保此路径为合法路径。
mount_from	是	Struct	共享存储的详细信息，参数描述请参见**表1-9**。
only_to	否	array[string]	指定需要挂载卷的步骤，为空则默认所有步骤均需要挂该卷。

表 1-9 mount_from 属性说明

属性	是否必选	参数类型	取值约束
pvc	是	字符串	CCE集群、CCI中共享存储（PVC）名字。可以在CCE、CCI界面上查询需要的PVC名称，GCS支持云硬盘卷、文件存储卷、对象存储卷和极速文件存储卷，具体请参见**图1-1**和**图1-2**。 **说明** 指定的PVC必须存在。 基因容器服务提供了3个内置的共享存储可供挂载，分别为： • ${GCS_REF_PVC}，基因容器提供的参考组OBS桶。 • ${GCS_DATA_PVC}，关联OBS桶，表示基因容器环境关联的OBS桶。 • ${GCS_SFS_PVC}，加速SFS卷，表示基因容器环境关联的SFS卷。
sub_path	否	字符串	只挂载共享存储的某个子目录。如：abc 或者 abc/def 可以使用 **inputs变量** 注意：挂载子目录abc，则写abc可以，不要写 /abc

图 1-1 查询 CCE PVC 名称

图 1-2 查询 CCI PVC 名称

volumes 配置样例

以下示例，使得每个任务容器会挂载5个共享存储，也就是所有的容器可以看到一样的共享目录。

```
volumes:
  genref:
    mount_path: '${volume-path-ref}'
    mount_from:
      pvc: '${GCS_REF_PVC}' #内置变量，基因容器提供的参考组OBS桶
  genobs:
    mount_path: '${volume-path-obs}'
    mount_from:
      pvc: '${GCS_DATA_PVC}' #内置变量，环境关联的OBS桶
  gensfs:
    mount_path: '${volume-path-sfs}'
    mount_from:
      pvc: '${GCS_SFS_PVC}' #内置变量，加速存储盘
  other:
    mount_path: /home/mydata
    mount_from:
      pvc: '${my_k8s_pvc}' # 关联的Kubernetes集群中的任意共享存储（PVC）名字
  subdir:
    mount_path: /home/subdir
    mount_from:
      pvc: '${my_k8s_pvc}'
      sub_path: '${path_var}'  #仅挂载共享存储的某个子目录，可以使用inputs变量，如: path_var = abc/def
```

1.6 ouputs

可选项，用于定义数据处理流程完成后，最终生成的结果文件列表。结果文件列表在基因容器服务的控制台界面展示，此外，为了让您更直观的查看结果文件，基因容器集成 IGV（IGV的详细描述请参见http://www.igv.org/）软件，您通过控制台即可查看到通过IGV软件转换后的可视化结果文件。

ouputs中可以同时定义多种结果文件。

须知

如果使用该字段，需要确保最终文件是在基因容器关联的OBS桶中。

outputs 格式

```
<结果文件名称>:
  paths: <结果文件列表>
  paths_iter: <结果文件列表，变量迭代方式>
```

表 1-10 结果文件属性说明

属性	是否必选	参数类型	取值约束
paths	否	array[String]	流程生成的文件在OBS桶中的路径列表。与paths_iter二选一 **说明** 此参数与path_iter必须二选一，其中path_iter支持变量代入。
paths_iter	否	参见**表2 paths_iter属性说明**	流程生成的文件在OBS桶中的路径列表，支持变量代入。

表 1-11 paths_iter 属性说明

属性	是否必选	参数类型	取值约束
path	是	String	在基因容器关联的OBS桶中，带变量的文件路径。
vars_iter	是	array[array]	二维数组，根据变量迭代，得到最终的文件列表，详见**表1-6**中的vars_iter参数。

outputs 配置样例

```
outputs:
  gene-report-vcf:
```

```
  paths:
  - ${sample}/merge.HaplotypeCaller.raw.vcf.gz
  - ${sample}/merge.HaplotypeCaller.raw.g.vcf.gz
```

1.7 YAML 语法

YAML 是一种简洁强大的语言，它的设计目标是便于设计和使用人员阅读。

基本语法规则

- 大小写敏感。
- 使用缩进表示层级关系。
- 缩进时不允许使用Tab键，只允许使用空格。
- 缩进的空格数目不重要，要求相同层级的元素左侧对齐。
- 使用#表示注释。

YAML 支持三种数据结构

- 对象：键值对的集合，又称为映射（mapping）/ 哈希（hashes）/ 字典（dictionary）。
- 数组：一组按次序排列的值，又称为序列（sequence） / 列表（list）。
- 纯量（scalars）：数据最小的单位， 单个的、不可再分的值。

对象

对象是一组键值对（key: value，冒号后面必须有一个空格或换行），合法的表示方法如下：

```
animal: pets
plant:
  tree
```

也可以将多个键值对写成一个行内对象：

```
hash: {name: Steve, foo: bar}
```

下面这种情况会出错

```
foo: somebody said I should put a colon here: so I did
windows_drive: c:
```

用引号括起来就没有问题，如下所示

```
foo: 'somebody said I should put a colon here: so I did'
windows_drive: 'c:'
```

数组

数组使用连字符和空格“- ”表示，合法的表示方法如下：

```
animal:
- Cat
- Dog
- Goldfish
```

也可使用行内表示法：

```
animal: [Cat, Dog, Goldfish]
```


对象和数组可以嵌套使用，形成复合结构：

```
languages:
 - Ruby
 - Perl
 - Python
websites:
 YAML: yaml.org
 Ruby: ruby-lang.org
 Python: python.org
 Perl: use.perl.org
```

纯量

纯量的数据类型有字符串、布尔值、整数、浮点数、Null、时间、日期。

- 字符串表示：

  字符串默认不使用引号表示：

  ```
  str: This_is_a_line
  ```

  如果字符串之中包含空格或特殊字符，需要放在引号之中：

  ```
  str: 'content: a string’
  ```

  单引号和双引号都可以使用，两者区别是单引号可以识别转义字符：双引号不会对特殊字符转义:↵

  ```
  s1: 'content:\n a string'
  s2: "content:\n a string"
  ```

  单引号之中如果还有单引号，必须连续使用两个单引号转义。

  ```
  str: 'labor''s day'
  ```

  字符串可以写成多行，从第二行开始，必须有一个单空格缩进。换行符会被转为空格。

  ```
  str: This_is
   a_multi_line
  ```

- 整数表示：

  ```
  int_value: 314
  ```

- 浮点型表示：

  ```
  float_value: 3.14
  ```

- Null表示：

  ```
  parent: ~
  ```

- 时间表示：

  时间采用ISO8601格式。

  ```
  iso8601: 2018-12-14t21:59:43.10-05:00
  ```

- 日期表示：

  日期采用复合ISO8601格式的年、月、日表示。

  ```
  date: 1976-07-31
  ```

一些特殊符号

- “---”表示一个Yaml文件的开始，“...”表示一个Yaml文件的结束。

```
---
# 一个美味水果的列表
- Apple
- Orange
- Strawberry
- Mango
...
```


- 对于整数型、浮点型、布尔型数据用两个感叹号“!!”进行强制转换：

```
strbool: !!str true
strint: !!str 10
```

- 多行字符串可以使用“|”保留换行符，也可以使用“>”折叠换行。这两个符号是Yaml中字符串经常使用的符号。

```
this: |
 Foo
 Bar
that: >
 Foo
 Bar
```

对应的对象为：

```
{ this: 'Foo\nBar\n', that: 'Foo Bar\n' }
```

一般推荐使用“|”就能够满足大多数场景了。

注释

YAML支持注释，这是YAML相比JSON的一个优点。

YAML的注释使用“#”开头，如下所示。

```
languages:
 - Ruby            # 这是Ruby语言
 - Go              # 这是Go语言
 - Python          # 这是Python语言
```

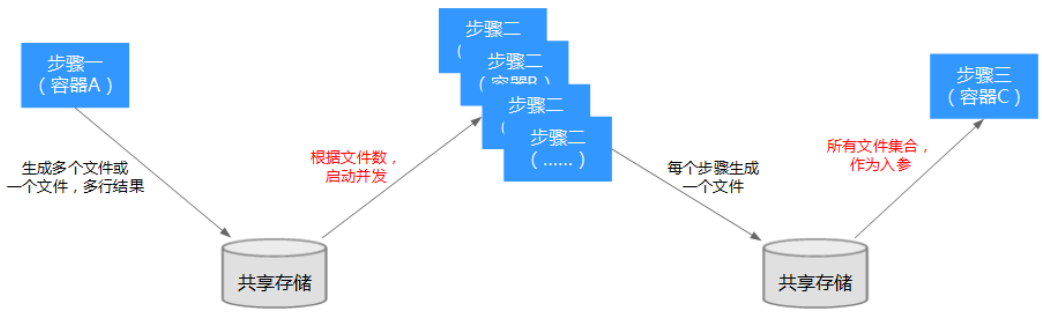
参考文档

- YAML 1.2 规格
- Ansible YAML Syntax
- YAML 语言教程

1.8 读取文件控制并发

在基因数据处理流程中，经常需要读取某个文件的内容来控制并发任务，或者获取另一个步骤的“输出结果”来控制并发任务。如，把样本文件按照固定大小进行拆分之后，需要得到所有的拆分文件名集合。或者上一步是分布式处理的，需要得到结果的总和。

图 1-3 读取文件控制并发



这种情况下，我们需要使用GCS语法中特定的功能，**get_result函数**来获得。

确定性的并发任务，GCS语法如下：

```
job-2:
  commands_iter:
    command: echo ${1} ${2}
    vars_iter:
      - [A, B, C]   # <==== 注意这里，表示${1} 的并发数量（范围）
      - [0, 1]
```

上述GCS语法中，[A, B, C] 表示并发数量为3数量。而动态的并发，就是数组[]中的值是上一步结果 。如下：

```
job-2:
  commands_iter:
    command: echo ${1} ${2}
    vars_iter:
      - get_result(job-1)   # <==== 注意这里，数组结果是根据指定任务的stdout动态“计算出来的”
      - [0, 1]
```

真正get到的result是指定步骤的标准输出，即“stdout”。

例如 job-1 的stdout为“1 2 3 4”，那么上一步的结果就是

```
    vars_iter:
      - ["1 2 3 4"]
```

注意，这里数组里面只有一个成员。也就是只有一个并发。

如果希望，每一个作为并发，可以**加入“分割符”进行切分**。具体语法见**get_result函数**。

```
    vars_iter:
      - get_result( job-1, " ")
```

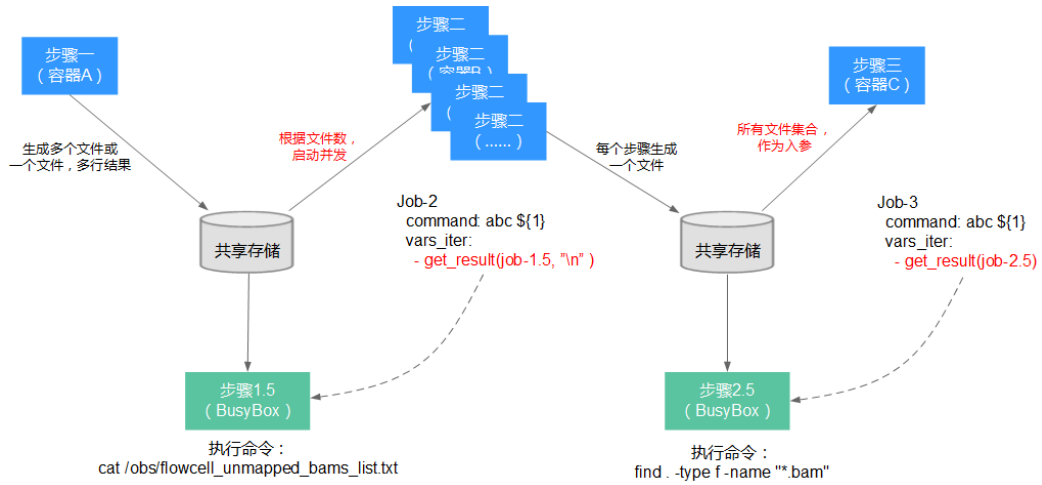
会得到：

```
    vars_iter:
      - ["1", "2", "3", "4"]
```

这样就得到了4个并发，每个变量分别为"1"， "2"， "3"， "4"。其中，分隔符是是任意的字符串。

在拥有get_result功能之后，如果需要实现根据上一步的结果，动态并发执行任务，则可以通过如下方式完成：

图 1-4 动态并发执行任务



这样可以通过执行自己熟悉的shell命令，读取某个文件内容，或者列出目录中特定的文件，来得到想要的并发行为。

示例

```
job-list:
  type: GCS.Job
  tool: nginx:new-latest
  commands: # <== (1) 列出目录中文件
    - |
      for i in `ls ${sfs}/${output-folder}`; do
        echo ${i}
      done
job-a:
  type: GCS.Job
  tool: nginx:new-latest
  commands_iter:
    command: echo ${output-prefix}job-c-${item} >> ${sfs}/${output-folder}/${1}; # <== (3) 迭代式并发，替换变量 ${1}
    vars_iter:
      - get_result(job-list, '\n') # <== (2) 把job-list输出，按一行一个，切分为并发数组
  depends:
    - target: job-list
      type: whole
```

1.9 条件分支

GCS支持条件分支功能，基础原理是 **根据每个步骤的condition字段来判断当前步骤是否执行**。根据业务使用场景分为简单条件分支和复杂条件分支。

- 简单条件分支，即在流程执行前，已经确定某个值是true还是false，从而可以确定对应的步骤是否需要执行。一般情况下，简单条件分支适用于condition字段，后跟布尔类型的值，做为判断。
- 复杂条件分支，即在流程执行过程中，根据流程执行的结果才能判断对应的步骤是否需要执行。一般情况下，复杂条件分支可以使用**check_result**函数来判断执行结果，并配合condition字段确定对应的步骤是否需要执行。

简单条件分支示例

所谓简单，就是在执行流程时，提前可以决定哪些步骤要不要执行，和软件中的if/else语法类似。例如完整流程为 A->B，运行流程时，输入给B步骤的condition条件为false，那么B步骤不需要执行。

如下示例中，如果job-a的condition值为false，则commands命令不会执行。

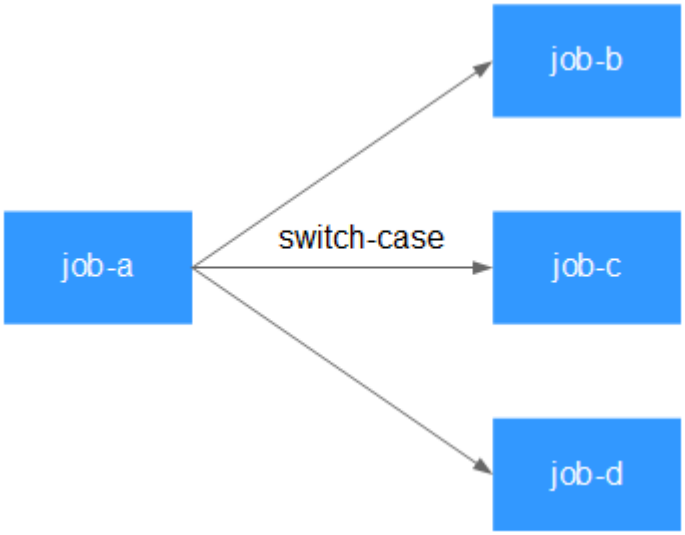
```
inputs:
  bool-var:
    default: true
    description: 布尔值输入变量
    type: bool
workflow:
  job-a:
    tool: nginx:latest
    condition: ${bool-var}  # <==通过变量，决定步骤是否执行。相当于 if ( bool-var )
    commands:
      - echo run-job-a
```


复杂条件分支示例

所谓复杂，就是流程运行过程中，基于运行结果才决定下一步骤要不要执行，和软件中if/else或是switch/case语法类似。例如: A步骤检测样本是否合格，如果合格则执行B步骤；如果不合格则执行C步骤。

因为需要根据上一个步骤的“执行结果”来判断下一步骤的condition值是true还是false，所以GCS使用了**check_result函数**来动态判断。

图 1-5 复杂条件分支示例



如下示例中，job-a的结果为pass，则job-b将会执行，job-c和job-d不会执行。

```
workflow:
  job-a:
    tool: nginx:latest
    commands:
      - echo pass  # <== 执行结果以标准std_out体现
  job-b:
    tool: nginx:latest
    condition: check_result(job-a, "pass")  # <==判断job-a的结果，等于pass才执行。
    commands:
      - echo run-job-b
  job-c:
    tool: nginx:latest
    condition: check_result(job-a, "npass")  # <==判断job-a的结果，等于npass才执行。
    commands:
      - echo run-job-c
  job-d:
    tool: nginx:latest
    condition: check_result(job-a, "failed")  # <==判断job-a的结果，等于failed才执行。
    commands:
      - echo run-job-d
```

说明

check_result的第二个参数可以使用 inputs变量，详细请参见**check_result**函数。示例如下:

```
condition: check_result(job-a, ${my_var})
```

依赖传递

如果流程为 A->B->C，当步骤B由于条件不满足而不需要执行时，由于依赖原因，那么步骤C不管condition是否为true也不需要执行。

即：

A -> B（**condition为false时**）-> C（**自动不需要执行**）

所以不需要为每个Job重复写condition条件，GCS自动会判断依赖传递。

2 内置函数

2.1 range

range用于迭代变量数量过多时，简化迭代变量范围。

须知

range仅在vars_iter字段内可用。

语法

```
job-a:
  commands_iter:
    command: echo ${1} ${2}
    vars_iter:
      - [A, B, C]
      - range(begin, end)
```

参数说明

表 2-1 参数说明

参数	是否必填	类型	描述
begin	是	Int	迭代范围的起始数值，包含该数值。
end	是	Int	迭代范围的结束值，不包含该数值。

返回值

对应迭代范围的数组。

示例

通过range函数，快速迭代执行命令。

```
job-a:
  commands_iter:
    command: echo ${1} ${2}
    vars_iter:
      - [A, B, C]
      - range(0, 10)   # <==== 注意这里，表示${2} 的并发数量(范围): [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.2 get_result

get_result用于获取“指定步骤”的标准输出，生成控制迭代的变量范围。

须知

- 目前支持获取的最大输出长度为 1M 字节，如果目标步骤的stdout超出，则execution执行时会报错终止（当然，您可以在失败后调整命令，然后触发继续执行）。
- get_result仅在vars_iter字段内可用。
- 获取结果的目标Job中，不允许使用GCS_JOB_START和GCS_JOB_END字段，这些字段为预留字段。

语法

```
job-a:
  commands_iter:
    command: echo ${1} ${2}
    vars_iter:
      - [A, B, C]
      - get_result( job-target, "split")   # <==== 注意这里，根据实际结果生成并发数量(范围)
```

参数说明

表 2-2 参数说明

参数	是否必填	类型	描述
job-target	是	String	指定获取结果的目标Job名。
split	否	String	用于分割字符串的“分割符”： ● 可以是单个字符，也可以是字符串； ● 使用双引号来表示，如 "\n"； ● 可以使用变量，如 ${input}。

返回值

字符串。

示例

假设某目标步骤“job-1”的标准输出为：

```
list-1.txt
list-2.txt
list-3.txt
list-4.txt
```

通过get_result函数，得到迭代执行命令。

```
job-a:
  commands_iter:
    command: echo ${1} ${2}
    vars_iter:
      - [A, B, C]
      - get_result( job-1, "\n")   # <==== 注意这里，表示${2}的并发数量(范围)
```

那么在执行job-a这个步骤的时候，会实际得到如下结果：

```
job-a:
  commands_iter:
    command: echo ${1} ${2}
    vars_iter:
      - [A, B, C]
      - "list-1.txt,list-2.txt,list-3.txt,list-4.txt"   # <==== 注意这里，表示${2}的并发数量(范围)
```

2.3 check_result

check_result用于获取“指定步骤”的标准输出，并判断是否与指定的字符串相等。主要用于控制流程的条件分支执行。

须知

- 目前支持获取的最大输出长度为1M字节，如果目标步骤的stdout超出，则execution执行时会报错终止（当然，您可以在失败后调整命令，然后触发继续执行）。
- check_result仅在condition字段内可用。

语法

```
job-a:
  tool: nginx:latest
  commands:
    - echo 111  # <== 某步骤的输出
job-b:
  condition: check_result(job-target, expect)  # <==== 注意这里，判断输出是否相等
```

参数说明

表 2-3 参数说明

参数	是否必填	类型	描述
job-target	是	String	指定获取结果的目标Job名。

参数	是否必填	类型	描述
expect	是	String	期待的值，用于判断条件是否满足： ● 使用双引号来表示，如"OK"； ● 可以使用变量，如 ${input}。

返回值

ture 或者 false。

- true：job-target的标准输出与expect参数值相等时，返回true；
- false：job-target的标准输出与expect参数值不相等时，返回false。

示例

假设某目标步骤“job-a”的功能为判断样本数据是否合格，其标准输出为：

```
ok
```

通过check_result函数，决定后续步骤执行分支。

```
job-a:
  tool: nginx:latest
  commands:
    - echo ok  # <== 步骤的输出
job-b:
  condition: check_result(job-a, "ok")      # <==== 这里运行结果为，conditon: true
job-c
  condition: check_result(job-a, "not_ok")  # <==== 这里运行结果为，conditon: false
```


3 内置变量

3.1 item

变量item用于在任务并发时，显示当前容器是第几个并发，该计数**从零开始**。

须知

item变量在workflow语法中，并发任务的command字段内可用。

语法

```
job-a:
  commands_iter:
    command: echo ${1} ${item}
    vars_iter:
      - [A, B, C]
```

解释

上述语法会产生3个并发任务，

第一个容器执行的命令${item} => 替换为0，最终执行为：echo A **0**

第二个容器执行的命令${item} => 替换为1，最终执行为：echo B **1**

第三个容器执行的命令${item} => 替换为2，最终执行为：echo C **2**

使用时，把item当做普通的**inputs变量**就可以。

3.2 GCS_REF_PVC

变量GCS_REF_PVC用于代表GCS提供的共享存储PVC，便于挂载存储。

须知

GCS_REF_PVC变量在workflow语法中，挂卷的volumes字段内可用。

语法

```
volumes:
  genref:
    mount_path: /ref  #挂载到容器内的路径
    mount_from:
      pvc: '${GCS_REF_PVC}' #内置变量，代表GCS提供的参考组OBS桶
```

解释

容器挂载共享存储是可以使用任意的PVC的，提供这个内置变量是为了省去用户定义PVC的动作。

使用时，把GCS_REF_PVC当做普通的**inputs变量**就可以。

3.3 GCS_DATA_PVC

变量GCS_DATA_PVC用于代表每个环境关联的OBS桶，便于挂载常用的对象存储桶。

须知

GCS_DATA_PVC变量在workflow语法中，挂卷的volumes字段内可用。

语法

```
volumes:
  genobs:
    mount_path: /obs  #挂载到容器内的路径
    mount_from:
      pvc: '${GCS_DATA_PVC}' #内置变量，代表执行流程的环境所关联的OBS桶
```

解释

容器挂载共享存储是可以使用任意的PVC的，提供这个内置变量是为了省去用户定义PVC的动作。你也可以自己定义PVC来完成这个挂载行为。

使用时，把GCS_DATA_PVC当做普通的**inputs变量**就可以。

3.4 GCS_SFS_PVC

变量GCS_SFS_PVC用于代表GCS环境的SFS文件存储卷，便于挂载常用的SFS文件存储卷。

须知

GCS_SFS_PVC变量在workflow语法中，挂卷的volumes字段内可用。

语法

```
volumes:
  gensfs:
    mount_path: /sfs  #挂载到容器内的路径
```



```
mount_from:
  pvc: '${GCS_SFS_PVC}' #内置变量，代表流程中使用的SFS文件存储卷
```

解释

容器挂载SFS文件存储卷是可以使用任意的PVC的，提供这个内置变量是为了省去用户定义PVC的动作。您也可以自己定义PVC来完成这个挂载行为。

使用时，把GCS_SFS_PVC当做普通的**inputs变量**就可以。

4 流程语法示例

4.1 单一步骤（一个并发）

命令示例

首先以执行单一命令为例。以 gcs sub job 为例，我们使用：

```
gcs sub job --cpu 0.5 --memory 1g --tool bwa:0.7.12 --shell sh bwa_help.sh
```

生成 GCS 流程示例

该命令将会生成以下模板，并提交基因容器语法解析器：

```
version: genecontainer_0_1
inputs:
  memory:
    default: 1g
    type: string
  cpu:
    default: 0.5c
    type: string
  tool:
    default: bwa:0.7.12
    type: string
  job-script:
    default: bwa_help.sh
    type: string
  jobid:
    default: bwa-help-2018-0830-171114-00d34
    type: string
  shell:
    default: sh
    type: string
workflow:
  bwa-help-2018-0830-171114-00d34:
    tool: bwa:0.7.12
    type: GCS.Job
    resources:
      memory: ${memory}
      cpu: ${cpu}
    commands:
      - sh /obs/gcscli/bwa-help-2018-0830-171114-00d34/bwa_help.sh
volumes:
  sample-data:
    mount_path: /obs
```



```
    mount_from:
      pvc: ${GCS_DATA_PVC}
  temp-data:
    mount_path: /sfs
    mount_from:
      pvc: ${GCS_SFS_PVC}
  ref-data:
    mount_path: /ref
    mount_from:
      pvc: ${GCS_REF_PVC}
outputs:
  bwa-help-2018-0830-171114-00d34:
    path:
      - /obs/output/bwa-help-2018-0830-171114-00d34
```

生成 Kubernetes 示例

该模板将进一步解析成以下 Kubernetes 模板 ， 并提交云容器引擎：

```
kind: Job
apiVersion: batch/v1
metadata:
  name: bwa-help-2018-0830-171114-00d34-0-81419ff7ad0d11e8
  namespace: default
  selfLink: >-
    /apis/batch/v1/namespaces/default/jobs/bwa-help-2018-0830-171114-00d34-0-81419ff7ad0d11e8
  uid: 815b643b-ad0d-11e8-aba9-fa163ed435e0
  resourceVersion: '706141'
  labels:
    gcs.execution.id: 81419ff7-ad0d-11e8-bebc-0255ac1066a1
    gcs.execution.name: gcs-2018-0830-171114-1-0-2018-0830-171114-35d77
    gcs.source.name: bwa-help-2018-0830-171114-00d34
    source: gcs
    stack-name: gcs-2018-0830-171114-1-0-2018-0830-171114-35d77
spec:
  parallelism: 1
  completions: 1
  backoffLimit: 6
  selector:
    matchLabels:
      controller-uid: 815b643b-ad0d-11e8-aba9-fa163ed435e0
  template:
    metadata:
      creationTimestamp: null
      labels:
        controller-uid: 815b643b-ad0d-11e8-aba9-fa163ed435e0
        gcs.execution.id: 81419ff7-ad0d-11e8-bebc-0255ac1066a1
        gcs.execution.name: gcs-2018-0830-171114-1-0-2018-0830-171114-35d77
        gcs.source.name: bwa-help-2018-0830-171114-00d34
        job-name: bwa-help-2018-0830-171114-00d34-0-81419ff7ad0d11e8
        source: gcs
    spec:
      volumes:
        - name: sample-data
          persistentVolumeClaim:
            claimName: XXXXXXXXX
        - name: temp-data
          persistentVolumeClaim:
            claimName: XXXXXXXXX
        - name: ref-data
          persistentVolumeClaim:
            claimName: XXXXXXXXX
      containers:
        - name: bwa-help-2018-0830-171114-00d34-0-81419ff7ad0d11e8
          image: '100.125.5.235:20202/genecontainer/bwa:0.7.12-r1039-sam18-sbb67'
          command:
            - sh
            - '-c'
            - sh /obs/gcscli/bwa-help-2018-0830-171114-00d34/bwa_help.sh
```



```
      resources:
        requests:
          cpu: '0.5'
          memory: 1G
      volumeMounts:
        - name: sample-data
          mountPath: /obs
        - name: temp-data
          mountPath: /sfs
        - name: ref-data
          mountPath: /ref
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      imagePullPolicy: IfNotPresent
  restartPolicy: OnFailure
  terminationGracePeriodSeconds: 30
  dnsPolicy: ClusterFirst
  securityContext: {}
  affinity:
    podAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          podAffinityTerm:
            labelSelector:
              matchLabels:
                source: gcs
            topologyKey: kubernetes.io/hostname
  schedulerName: default-scheduler
```

4.2 单一步骤（并发执行）

其次，再看一下任务并发的情况。假如用户之前脚本是这样：

```
#run.sh
var1=$1
var2=$2
echo "$var1\t$var2"

# 执行该脚本
sh run.sh 1 a
sh run.sh 2 a
sh run.sh 1 b
sh run.sh 2 b
```

则这里模板有两种写法。第一种是写出所有可能情况。为了简化，这里只展示 workflow 部分：

```
workflow:
  splitfq:
    tool: zsplit:0.2
    type: GCS.Job
    resources:
      memory: 1g
      cpu: 0.5c
    commands:
      - sh /obs/gcscli/run-xxx/run.sh 1 a
      - sh /obs/gcscli/run-xxx/run.sh 2 a
      - sh /obs/gcscli/run-xxx/run.sh 1 b
      - sh /obs/gcscli/run-xxx/run.sh 2 b
```

第二种是使用并发变量：

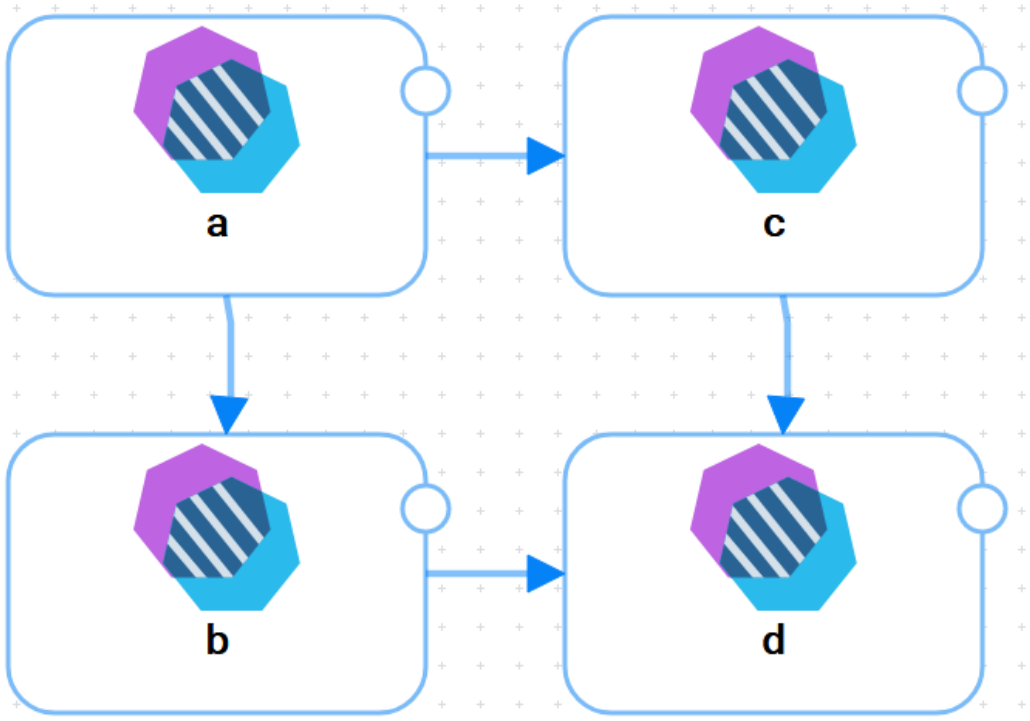
```
inputs:
  var1:
    default:
      - 1
      - 2
```



```
  description: var1
  type: array
  label: basic
 var2:
  default:
   - a
   - b
  description: var2
  type: array
  label: basic
workflow:
 splitfq:
  tool: zsplit:0.2
  type: GCS.Job
  resources:
   memory: 1g
   cpu: 0.5c
  commands_iter:
   command: >
    sh /obs/gcscli/run-xxx/run.sh ${1} ${2}
   vars_iter:
    - '${var1}'
    - '${var2}'
```

4.3 复杂依赖任务（DAG）

如果有四个任务，a/b/c/d 我们希望首先执行 任务 a，然后执行 b/c，b/c 同时完成后，最后执行 d，如下图所示：



为了实现这个目标，需要引入 depends 字段。以任务 d为例，完成任务 d 需要首先完成任务 b/c，则depends 字段为：

```
  depends:
   - target: b
     type: whole
   - target: c
     type: whole
```


完整代码如下：

```
version: genecontainer_0_1
inputs:
  memory:
    default: 1g
    type: string
  cpu:
    default: 0.5c
    type: string
  tool:
    default: bwa:0.7.12
    type: string
  shell:
    default: sh
    type: string

workflow:
  a:
    tool: bwa:0.7.12
    type: GCS.Job
    resources:
      memory: 1g
      cpu: 0.5c
    commands:
      - echo "A"
  b:
    tool: bwa:0.7.12
    type: GCS.Job
    resources:
      memory: 1g
      cpu: 0.5c
    commands:
      - echo "B"
    depends:
      - target: a
        type: whole
  c:
    tool: bwa:0.7.12
    type: GCS.Job
    resources:
      memory: 1g
      cpu: 0.5c
    commands:
      - echo "C"
    depends:
      - target: a
        type: whole
  d:
    tool: bwa:0.7.12
    type: GCS.Job
    resources:
      memory: 1g
      cpu: 0.5c
    commands:
      - echo "D"
    depends:
      - target: b
        type: whole
      - target: c
        type: whole
volumes:
  sample-data:
    mount_path: /obs
    mount_from:
      pvc: ${GCS_DATA_PVC}
  temp-data:
    mount_path: /sfs
    mount_from:
      pvc: ${GCS_SFS_PVC}
```



```
ref-data:
  mount_path: /ref
  mount_from:
    pvc: ${GCS_REF_PVC}
```

