吉云流程开发及部署说明文档

版本: 0.8.0

作者: 刘久成

部门: 深圳基因研究院

日期: September 4, 2019

Contents

1 分析流程

分析流程是根据预设的指令利用软件对数据进行的一系列分析。生物信息分析流程则是针对生物学数据的，主要包含数据 (输入/输出)、软件和流程三部分。如常见的 RNA-Seq 分析流程 (图 1)。其中包括输入信息读取，并行计算，分支决策等。

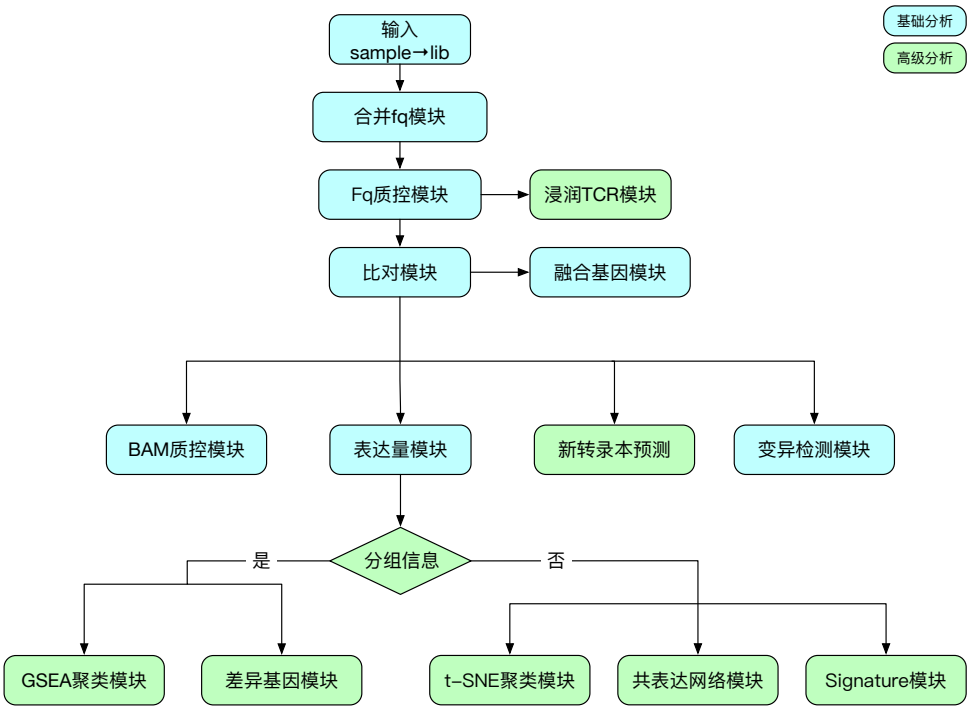

Fig. 1: RNA-Seq 分析流程

软件方面，生物信息分析流程往往包含多种分析软件，部署安装和版本控制是个令人头痛的问题，即便是费尽心力克服重重依赖，找遍 StackOverflow 解决各种报错后部署成功，高兴之余，最怕听到的需求便是“在新机器上再来一次!”，软件分发的难题一直困扰着生物信息分析人员。流程方面，如何处理好各步骤的依赖关系，合理安排并行化处理，监控流程运行进度和状态，又向分析人员提出了挑战。Docker (容器引擎) 和 WDL (Workflow Description Language，工作流描述语言，发音 [widdle]) 的出现很大程度上解决了上述问题，而且改善了生物信息分析的可重复性。本文将以 RNA-Seq 为例，对以上方面进行描述和实际操作。

2 Docker

Docker 是一个开源的应用容器引擎 (container engine)，让开发者可以打包他们的应用以及依赖包到一个可移植的镜像中 (容器)，然后发布到任何流行的 Linux/MacOS 或 Windows 机器上，也可以实现虚拟化。Docker 的三个概念分别是: 仓库 (Registry)、镜像 (Image) 和容器 (Container)。图 2展示了三者的关系。仓库是存放 Docker 镜像的地方，拉到本地的镜像通过 `docker run` 运行 (此实例称为容器)。经过修改的容器可以经过 `docker commit` 保存为新的镜像。

Docker 的特点和优势:

- 容器是轻量的、可执行的独立软件包，包含软件运行所需的所有内容：代码、运行时环境、系统工具、系统库和设置。
- 容器化软件适用于基于 Linux 和 Windows 的应用，在任何环境中都能够始终如一地运行。
- 容器赋予了软件独立性，使其免受外在环境差异（例如，开发和预演环境的差异）的影响，从而有助于减少团队间在相同基础设施上运行不同软件时的冲突。

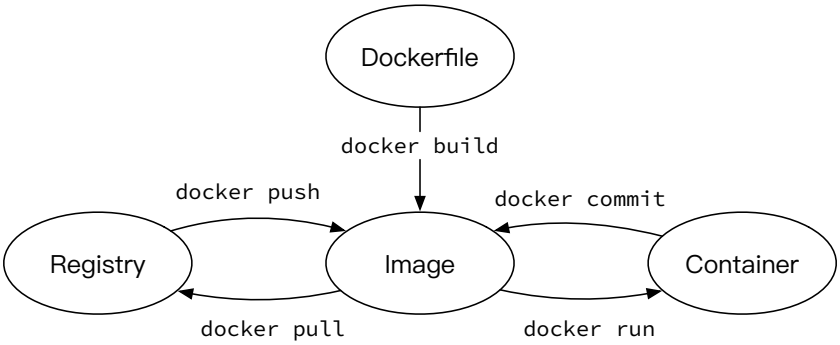

Fig. 2: Docker 核心概念及关系

Docker 支持不同的 Linux 发行版本共享同一个内核 (图 3)，主流 Linux 发行版本大都具有官方或非官方的 Docker 镜像和支持社区，可以根据实际需要进行选择。

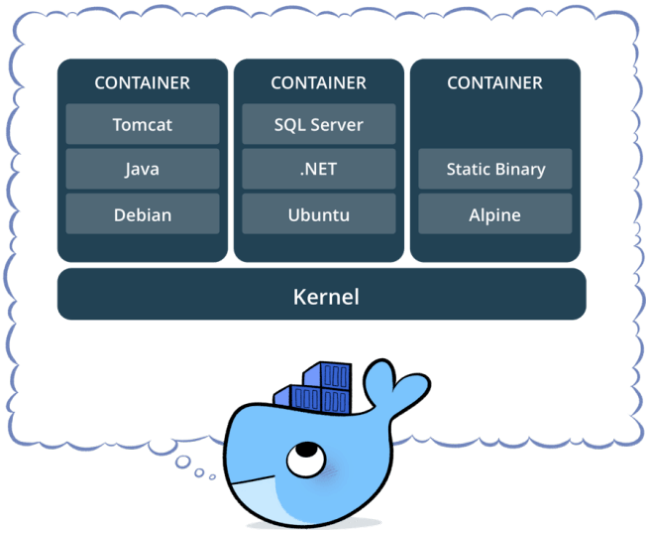

Fig. 3: Docker 容器

2.1 容器创建

创建一个容器镜像有两种方式，一种是直接下载公共的镜像或 Dockerfile，另外就是基于 Linux 基础镜像自己制作。

2.1.1 公共镜像

Docker Hub (https://hub.docker.com) 是一个由 Docker 公司运行和管理的基于云的存储仓库，Docker 镜像可以由其他用户发布和使用。我们可以通过 `docker pull xxx` 下载所需的镜像。表 1列举了一些下载基础镜像、生物信息分析软件和流程镜像的实例:

通过 `docker pull` 下载好的 Docker 镜像可以用 `docker images` 命令查看:

Tab. 1: Docker Hub 镜像下载

镜像	下载命令
CentOS	`docker pull centos`
Ubuntu	`docker pull ubuntu`
Alpine	`docker pull alpine:3.10.1`
alpine–java	`docker pull anapsix/alpine-java`
GATK3	`docker pull broadinstitute/gatk3`
GATK4	`docker pull broadinstitute/gatk`
RNACocktail	`docker pull marghoob/rnacocktail`

```
$ docker images
REPOSITORY                    TAG     IMAGE ID      CREATED       SIZE
alpine                        latest  b7b28af77ffe  3 weeks ago   5.58MB
alpine                        gatk4   8fabccb5d2cd  3 hours ago   409MB
broadinstitute/gatk           latest  9e737a9f562c  3 months ago  3.84GB
quay.io/bcbio/bcbio-rnaseq    latest  babf4a2eefa8  3 months ago  4.63GB
centos                        latest  9f38484d220f  4 months ago  202MB
anapsix/alpine-java           latest  c45785c254c5  5 months ago  126MB
bioconductor/release_base     latest  18f6a97aa589  2 years ago   1.28GB
marghoob/rnacocktail          latest  a848eb367be6  17 months ago 3.33GB
```

如果想查看镜像的内容，可以使用以下命令进入其容器的 Bash 环境，进而获得容器的内容。

```
docker run -it --rm alpine:gatk4 bash
```

值得一提的是生物信息相关流程的 Docker 镜像往往文件较大，包含了很多不必要的组件，不利于传输、部署和启动。Alpine 是一个轻量级的 Linux 发行版本，其基础 Docker 镜像大小仅 5MB 多。基于 Alpine 制作的 GATK4 大小为 409MB，较官方的 3.84GB 瘦身了很多。下面介绍下制作过程。

2.1.2 手工制作

下面是手动制作基于 Alpine 的 GATK4 镜像的过程:

1. 获取基础镜像

   ```
   docker pull anapsix/alpine-java
   ```

2. 下载 GATK

   ```
   wget https://github.com/broadinstitute/gatk/releases/download/4.1.2.0/gatk-4.1.2.0.zip
   unzip gatk-4.1.2.0.zip
   cd gatk-4.1.2.0
   ```

3. 运行 docker 镜像并拷贝 GATK

   ```
   docker run -it -v $PWD:/tmp anapsix/alpine-java bash
   cp /tmp/gatk-package-4.1.2.0-local.jar /opt/gatk4.jar
   exit
   ```

4. 获取容器 ID 并保存为新镜像

```
gatk_id=$(docker ps -a | sed '1d' | head -n1 | cut -d' ' -f1)
docker commit -m 'gatk4' $gatk_id alpine:gatk4
```

5. 镜像测试

```
docker run -it --rm alpine:gatk4 java -jar /opt/gatk4.jar -version
```

正常情况下会显示 GATK4 的版本信息:

```
The Genome Analysis Toolkit (GATK) v4.1.2.0
HTSJDK Version: 2.19.0
Picard Version: 2.19.0
```

2.1.3 Dockerfile

如果想要从一个基础镜像开始建立一个自定义镜像，可以选择一步一步进行构建，也可以选择写一个配置文件，然后一条命令（`docker build`）完成构建，显然配置文件的方式可以更好地应对需求的变更，这个配置文件就是 Dockerfile，其详细的介绍可以参阅Dockerfile 文档。RNACocktail 分析流程作者提供了Dockerfile文件，可以结合文档阅读一下。

我们将上面手动制作 GATK4 镜像的过程通过 Dockerfile 实现，其内容如下:

```
FROM anapsix/alpine-java

ENV GATK_VERSION 4.1.2.0

RUN wget https://github.com/broadinstitute/gatk/releases/download/${GATK_VERSION}/gatk-$
    {GATK_VERSION}.zip && unzip gatk-${GATK_VERSION}.zip && cp gatk-${GATK_VERSION}/gatk
    -package-${GATK_VERSION}-local.jar /opt/gatk4.jar && rm -rf gatk-${GATK_VERSION}*
```

在 Dockerfile 所在的目录运行 `docker build .` 制作镜像，成功后会显示其镜像 IMAGE_ID，再通过`docker tag`添加标签。

```
docker tag $IMAGE_ID 'alpine:gatk4'
```

此标签也可以作为`build`的参数指定（`-t`）:

```
docker build -f /path/to/Dockerfile -t 'alpine:gatk4' .
```

顺便说一下，Alpine 官网 (https://pkgs.alpinelinux.org/packages) 提供了常用的应用程序，可以通过`RUN apk add xxx`进行安装，例如 R 的镜像制作:

```
FROM alpine
RUN apk add --no-cache R
CMD ["R"]
```

生物信息分析流程根据其复杂程度，在初次封装时往往通过自动和手动相结合的方式实现。在基础镜像的选择上，可以选择和自己的 host 机器上相同的 Linux 发行版本 (如 host 机器是 CentOS 系统，我们选择`FROM centos`)，这样可以将 host 机器上编译好的程序直接拷贝到 Docker 镜像里面使用。在测试通过后，可以将手动封装的部分加到 Dockerfile 中实现全自动封装。Docker 镜像制作中常用软件安装方法在下节列出供参考。

2.2 软件安装

2.2.1 系统工具

各 Linux 发行版本都提供了界面 (UI) 和终端 (CLI) 版本的软件管理工具，这些工具通常只有 root 用户可以使用。Docker 容器实例是以 root 账户登陆的，可以使用这些系统命令。

Tab. 2: 系统软件安装工具

Linux 发行版本	安装软件命令
Ubuntu/Debian	`apt-get install xxx`
CentOS	`yum install xxx`
Alpine	`apk add xxx`

2.2.2 编译安装

C/C++ 等语言开发的软件 (如 bwa，samtools 等) 通常需要经过编译成二进制的执行文件才能使用。如果开发者提供了编译好的二进制版本，可以根据自己的平台版本下载对应的软件使用。如果只有源代码或者是我们需要对源码做一些修改，则需要编译后使用。根据维基百科对Automake过程的描述 (图 4)，此过程由软件开发者决定，可能从不同的节点开始。关键节点文件是 autogen.sh、`configure`和 Makefile。

如果源码文件夹有 Makefile 文件:

```
tar -xf bwa-0.7.17.tar.bz2 && cd bwa-0.7.17
make
cp bwa /usr/local/bin
```

如果源码文件夹有`configure`文件:

```
tar -xf samtools-1.9.tar.bz2 && cd samtools-1.9
./configure --prefix=/usr/local
make -j 16 && make install
```

如果是非 root 用户，可以指定个人目录 (如 $HOME):

```
./configure --prefix=$HOME
make -j 16 && make install
```

这样 samtools 就会安装到`/home/$USER/bin/samtools`。

如果源码文件夹没有以上两个文件，但是有 autogen.sh 或 bootstrap，可以运行该命令产生`configure`文件，然后运行`./configure`及`make`命令。需要注意的是如果提供了 test 文件，可以运行`make test`测试程序的有效性后再`make install`安装。

有些软件推荐使用`cmake`进行编译，配置文件为 CMakeLists.txt，常规用法如下:

```
cmake && make install
```

如果软件要求在单独的目录编译，则:

```
mkdir build
cd build
cmake ..
make install
```

自定义目标路径可以在`cmake`步骤或`make install`步骤指定:

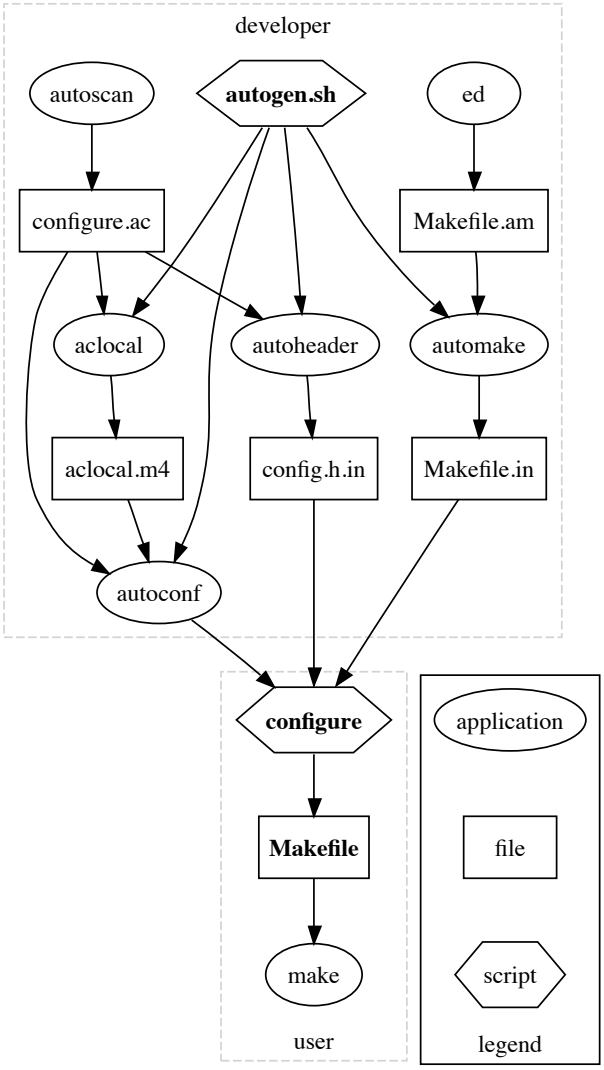

Fig. 4: Autoconf

```
cmake -D CMAKE_INSTALL_PREFIX='/usr/local'
```

或

```
make install PREFIX='/usr/local'
```

Alpine 基础镜像不带编译工具，如果要封装需要编译的软件，则需要添加 build-base，下面是安装 samtools 的 Dockerfile 实例:

```
FROM alpine:latest

ENV SAMTOOLS_VERSION 1.9

RUN apk add --update pcre-dev openssl-dev \
 && apk add --virtual build-dependencies build-base curl zlib-dev \
 && curl -L -o samtools-${SAMTOOLS_VERSION}.tar.bz2 \
    https://github.com/samtools/samtools/releases/download/${SAMTOOLS_VERSION}/samtools-
    ${SAMTOOLS_VERSION}.tar.bz2 \
 && tar jxvf samtools-${SAMTOOLS_VERSION}.tar.bz2 \
 && cd samtools-${SAMTOOLS_VERSION}/ \
 && ./configure --without-curses --disable-lzma --disable-bz2 --disable-libcurl \
 && make && make install \
 && cd .. && rm -rf samtools-${SAMTOOLS_VERSION}* \
 && apk del build-dependencies && rm -rf /var/chache/apk/*
```

其中的`--virtual build-dependencies`参数将后面的参数 (build-base curl zlib-dev) 命名为 build-dependencies 组,方便在编译安装软件后删除,节省空间。这种情况下,要求以静态的方式进行软件编译, 如 fastp 软件的安装, 默认是动态编译的, 我们需要明确指定`export CXX="c++ -static"`来实现静态编译。如果以默认的动态编译进行，在删除 build-dependencies 后 fastp 程序会报错找不到相关依赖。

```
FROM alpine:latest

ENV FASTP_VERSION 0.20.0

RUN apk add --update \
 && apk add --virtual build-dependencies build-base curl zlib-dev \
 && curl -L -o v${FASTP_VERSION}.tar.gz \
    https://github.com/OpenGene/fastp/archive/v${FASTP_VERSION}.tar.gz \
 && tar -xvf v${FASTP_VERSION}.tar.gz && cd fastp-${FASTP_VERSION} \
 && export CXX="c++ -static" && make && mv fastp /usr/local/bin \
 && cd .. && rm -rf v${FASTP_VERSION}.tar.gz fastp-${FASTP_VERSION} \
 && rm -rf /var/chache/apk/* && apk del build-dependencies
```

2.2.3 Java 程序

除了 Java 开发者，从源码编译 Java 程序需求很少，而且生物信息分析中常用的 Java 程序都有编译好的发行版本，只要把 jar 包下载好放到指定的位置即可，如 Picard 工具放置在`/opt/jar/picard.jar`路径下。

2.2.4 Python 模块

Python 模块的安装有以下两种常用方式:

1. 手动安装，下载 Python 模块源码 tar 包解压:

   ```
   python setup.py build
   python setup.py install
   ```

   如果有权限问题或想自定义安装路径，可以指定安装前缀:

   ```
   python setup.py install --prefix=/usr/local
   ```

2. 使用 pip 安装

```
pip install xxx
pip install xxx --target=/path/to/install
```

Python 版本 2 和 3 的模块可能要按需安装

```
pip2 install xxx
pip3 install xxx
```

2.2.5 Perl 模块

Perl 模块的安装有以下两种常用的方式:

1. 手动安装

```
wget https://cpan.metacpan.org/authors/id/J/JM/JMCNAMARA/Excel-Writer-XLSX-1.00.tar.gz
tar -xf Excel-Writer-XLSX-1.00.tar.gz
cd Excel-Writer-XLSX-1.00
perl Makefile.PL
make
make install
```

如果为非 root 用户，遇到安装权限问题时，可以指定模块的安装路径:

```
perl Makefile.PL PREFIX=$HOME
```

手动安装适用于 Perl 模块依赖较少或服务器未联网的情况，如果有联网，优先使用 CPAN 进行安装，依赖的模块会自动安装。

2. 利用 CPAN 模块安装

```
perl -MCPAN -e shell
cpan> install Excel::Writer::XLSX
```

2.2.6 R 模块/包

R 语言模块来源主要有两个: CRAN与BioConductor。
CRAN 包安装有以下三种方式:

1. 下载 R 包到到本地并上传至未联网的服务器上，在 Linux 命令行安装。适用于 R 包依赖较少且机器无法联网的情况。

```
R CMD INSTALL package.tar.gz
```

2. 在 Linux 命令行使用 `install.packages` 安装，这种安装方式可用于 Dockerfile(2.1.3)

```
R -e "install.packages('package', repos = 'http://cran.us.r-project.org')"
```

3. 在交互界面 (IDE) 安装

```
> install.packages("package_name")
```

在 IDE 安装时 R 会让用户选择 CRAN 镜像，可以在安装时直接指定镜像地址，如:

```
> install.packages("pheatmap", repos='http://cran.us.r-project.org')
```

另外，已经下载好的 R 包或本地修改过的 R 包源码也可以通过以下方式安装:

```
> install.packages("package.tar.gz", repos = NULL, type="source")
```

BioConductor 包在 R 的 IDE 下通过以下命令进行安装，以安装“GenomicFeatures”和“AnnotationDbi”为例:

```
> if (!requireNamespace("BiocManager", quietly = TRUE))
+     install.packages("BiocManager")
> BiocManager::install(c("GenomicFeatures", "AnnotationDbi"))
```

对于 Github 上的 R 源码包，可以使用 devtools 进行安装，例如:

```
> library(devtools)
> install_github("wch/ggplot2")
```

2.3 软件路径

Docker 中默认用户为 root，如果不指定软件安装路径，则除 jar 程序包外，其他执行程序及文库/模块会安装在 `/usr` 或 `/usr/local` 目录下。可执行文件在 `/usr/bin` 或 `/usr/local/bin` 目录下，这些路径已经配置在系统 `$PATH` 变量中，可以直接执行，免于指定绝对路径，方便流程调用。

对于 jar 包，我们统一放置在`/opt`下，在使用时指定绝对路径，如:

```
java -d64 -server -XX:+UseParallelGC -XX:ParallelGCThreads=8 \
  -Xms8g -Xmx16g -Djava.io.tmpdir=tmp \
  -jar /opt/jar/picard.jar MarkDuplicates \
  I=input.bam O=out_markdup.bam \
  METRICS_FILE=out.metrics ASO=coordinate \
  VALIDATION_STRINGENCY=LENIENT
```

有些软件是由多个程序组成 (如 STAR-Fusion)，内涵调用流程，这种情况下也适合将其放置到 `/opt` 目录，将其中的主程序软连接到 `/usr/local/bin` 使用。比如将 STAR-Fusion 放置在 `/opt/app` 路径下，其目录结构如下:

```
/opt/app/STAR-Fusion-v1.6.0/
├─ FusionAnnotator
├─ FusionFilter
├─ FusionInspector
├─ LICENSE
├─ Makefile
├─ PerlLib
├─ README.md
├─ STAR-Fusion
├─ notes
├─ plugins
└─ util
```

其调用程序为`STAR-Fusion`，我们为其创建软连接:

```
cd /opt/app/STAR-Fusion-v1.6.0/
ln -s $PWD/STAR-Fusion /usr/local/bin
```

这里有一点需要注意，STAR-Fusion 为 Perl 脚本，内部使用了 FindBin 模块来找到 STAR-Fusion 所在的目录，当我们创建了软连接后，`$FindBin::Bin` 失效，需要使用 `$FindBin::RealBin`。

```
use FindBin;
use lib ("$FindBin::RealBin/PerlLib");

my $BASEDIR = "$FindBin::Bin";
                          ↓
my $BASEDIR = "$FindBin::RealBin";
```

经过上述小改动后，就可以直接运行 `STAR-Fusion` 了，无需指定其绝对路径。当然我们可以通过绝对路径运行（`/opt/app/STAR-Fusion-v1.6.0/STAR-Fusion`），或者是将其路径加入到用户的$PATH中，通过以下命令实现:

```
export PATH="/opt/app/STAR-Fusion-v1.6.0:$PATH"
```

如果令其在用户登录后自动生效，可以将上述命令写入 `~/.bashrc` 中，这样就无需指定绝对路径了。

2.4 版本记录

软件版本信息可以统一记录到文本文件中，如`/opt/versions.txt`。

2.5 平台部署

一般用户本地制作的 docker 镜像部署到吉云平台需要以下步骤:

1. 镜像标签 ⟶ 吉云平台要求镜像的标签为 `usrname/imgname` 格式。例如我制作的 samtools 镜像，需要更新标签后再保存 (`usrname`为吉云平台用户名):

   ```
   docker tag 'alpine:samtools' 'liujc/samtools'
   ```

2. 保存镜像

   ```
   docker save -o samtools.tar liujc/samtools:latest
   ```

3. 上传镜像 ⟶ 将 samtools.tar 上传至吉云服务器，然后通过以下命令推送至 docker 仓库:

   ```
   $ imgcli push samtools.tar
   Uploading image 36.88 MiB / 36.88 MiB [============================]

   Succeed to push image.
   ```

4. 查看镜像

   ```
   $ imgcli list
   Listing liujc images.

   +-----+----------------+
   | Seq |          Image |
   +=====+================+
   | 0   | liujc/samtools |
   +-----+----------------+
   ```

管理员或者具有 docker 执行权限的用户也可以通过直接运行 docker 命令实现镜像部署。有可能 load 后的镜像的标签为 <none>，在这种情况下，我们需要根据 IMAGE–ID 添加标签。

```
docker load samtools.tar
docker tag IMAGE_ID 'liujc/samtools'
docker push liujc/samtools
```

3 Workflow

3.1 WDL 介绍

Broad Institute 在发布 GATK4 正式版的同时,还发布了 WDL (https://software.broadinstitute.org/wdl) 和 GATK 系列最佳实践 WDL 流程 (https://software.broadinstitute.org/gatk/best-practices/)，相比 CWL (Common Workflow Language，https://www.commonwl.org)，WDL 语法简单、易学、易用。WDL 将工作流程分为`变量`，`workflow`，`task`，`call`，`command`和`output`等几部分，下面分别简要介绍下。

3.1.1 变量

WDL 有两种不同层次的变量，位于 workflow 或 task 中。变量可以通过 task 的 output 指定功能从一个 task 传递到下一个 task，形式为`task_name.variable_name`。按照变量的组成又可以分为简单变量和复合变量 (表 3)。复合变量由简单变量组成。

Tab. 3: WDL 变量类型

简单变量	复合变量
String	Array
Int	Map
Float	Object
File	Pair
Boolean	

3.1.2 流程 (workflow)

WDL 的顶层组件为`workflow`(流程)，`task`(任务) 和`call`(调用)。workflow 在顶层，通过 call 去执行 tasks，tasks 在 workflow 模块外被定义，workflow 定义原始输入，`task` 之间通过输入/输出 variable(变量) 来确定依赖关系及执行顺序 (图 5)。

3.1.3 任务 (task)

task 的核心组件: `command`、`output`和`runtime` (图 6)。以 RNA-Seq WDL 流程的`mergefq`为例:

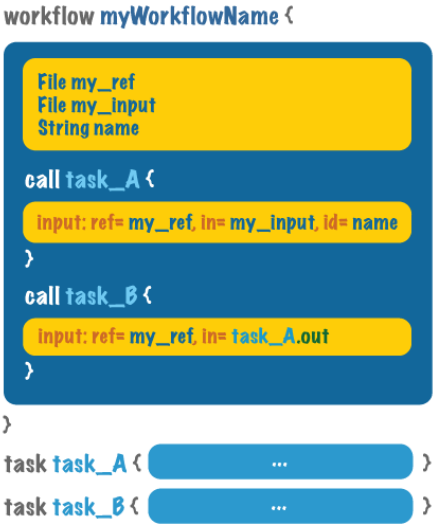

Fig. 5: Workflow

Fig. 6: Task

```
task mergefq {
    File list
    File fqdir
    File outdir
    command {
        mergefq -s ${list} -i ${fqdir} -o ${outdir}
    }
    output {
        Array[Array[String]] sm2fq = read_tsv(stdout())
    }
    parameter_meta {
        list: "sample to library list file"
        fqdir: "input directory containing fqs of samples"
        outdir: "output directory"
    }
    meta {
        author: "liujc"
        email: "liujc@geneplus.org.cn"
    }
    runtime {
        docker: "liujc/os"
        cpu: "1"
        memory: "100M"
        retry: 1
    }
}
```

- `command` 是程序实际执行的命令，变量通过 `${...}` 形式调用，其中花括弧是必须的。
- `output` 输出是二维的字符串数组，内容为 sample→fq1→fq2 映射表。输出的赋值需用双引号括起来。
- `runtime` 运行时的参数指定:

`docker`	调用的 docker 镜像名
`cpu`	使用的 cpu 数目
`memory`	使用的内存大小
`retry`	失败后重试次数

- `parameter_meta` 参数说明 (可选)
- `meta` 作者信息 (可选)

`command`内环境为 sh，命令逐行执行，支持两种格式: 简单`{...}`和高级`<<<...>>>`。简单的命令可以使用前者，这里通过举例介绍下后者的用法 (省略`runtime`部分)。

1. 使用 Bash 环境及命令 (有些命令 sh 是不支持的)

```
task vcomb {
    File f1
    File f2
    command <<<
        bash <<CODE
            paste <(sort -k1,1n ${f1}) <(sort -k1,1n ${f2}) | cut -f1,2,4
        CODE
    >>>
    output { Array[String] res = read_tsv(stdout()) }
}
```

2. 使用 awk 命令 (awk 命令中可能有`{`和`}`，简单格式不支持)

```
task colsum {
    File f
    command <<<
        awk '{a+=$NF;}END{print a}' ${f}
    >>>
    output { Int sum = read_int(stdout()) }
}
```

3. 使用其他语言环境 (如 Python)

```
task test {
    command <<<
        python <<CODE
            for i in range(3):
                print("key_{idx}\t{idx}".format(idx=i)
        CODE
    >>>
    output {
        Map[String, Int] my_ints = read_map(stdout())
    }
}
```

3.2 WDL 流程开发

我们以 RNA-Seq 流程为例分别 WDL 流程的关键步骤进行介绍。

3.2.1 输入处理

万事开头难，流程的难点在于输入的设计和处理。一般将样本名称 ID 作为分析过程中的唯一标识符使用。NGS 测序中样本是单独建库 (样本 ID 和文库号一一对应)，然后混合 (pooling) 在一起测序的 (通过样本建库时添加的标签序列-barcode 拆分成每个样本的测序数据)，常有同一个样本在测序仪 Flowcell(流动池，俗称“片子”) 上多条 Lane(泳道) 测序的情况，如果是 Pair-End 测序，则每条 Lane 测得的序列又分为 fq1 和 fq2 两个文件:

```
.
└── 199003852TR
    ├── 190501_Q100020006_V300018490_L001_199003852TR_HUM_C_GC1A_504
    │   ├── 190501_Q100020006_V300018490_L001_199003852TR_HUM_C_GC1A_504_1.fq.gz
    │   └── 190501_Q100020006_V300018490_L001_199003852TR_HUM_C_GC1A_504_2.fq.gz
    ├── 190501_Q100020006_V300018490_L002_199003852TR_HUM_C_GC1A_504
    │   ├── 190501_Q100020006_V300018490_L002_199003852TR_HUM_C_GC1A_504_1.fq.gz
    │   └── 190501_Q100020006_V300018490_L002_199003852TR_HUM_C_GC1A_504_2.fq.gz
    ├── 190501_Q100020006_V300018490_L003_199003852TR_HUM_C_GC1A_504
    │   ├── 190501_Q100020006_V300018490_L003_199003852TR_HUM_C_GC1A_504_1.fq.gz
    │   └── 190501_Q100020006_V300018490_L003_199003852TR_HUM_C_GC1A_504_2.fq.gz
    └── 190501_Q100020006_V300018490_L004_199003852TR_HUM_C_GC1A_504
        ├── 190501_Q100020006_V300018490_L004_199003852TR_HUM_C_GC1A_504_1.fq.gz
        └── 190501_Q100020006_V300018490_L004_199003852TR_HUM_C_GC1A_504_2.fq.gz
```

测序下机文件的名称中一定含有文库号 (HUM_*)，样本 ID 不保证一定含有，所以我们通过输入`样本 ID`与`文库号`的对应表 (sam2lib.txt) 来实现样本与数据的对应关系。

```
Cc1Xtao:rnaseq $ cat sam2lib.txt
199003847TR    HUM_C_GC1A_501
199003848TR    HUM_C_GC1A_502
199003850TR    HUM_C_GC1A_503
199003852TR    HUM_C_GC1A_504
```

RNA-Seq 流程的第一步是按样本将不同 lane 的数据合并至 fq1 和 fq2，通过`mergefq`程序实现。如前面`mergefq`任务 (3.1.3) 所列出的那样，输入为样本文库对应表、含有 fq 序列文件的目录和输出目录。`mergefq`程序在指定的输出文件夹按样本 ID 创建文件夹，并合并该样本的所有`*_1.fq.gz`文件至`sampleID_1.fq.gz`，`*_2.fq.gz`至`sampleID_2.fq.gz`：

```
run
├── 199003847TR
│   └── RawFq
│       ├── 199003847TR_1.fq.gz
│       └── 199003847TR_2.fq.gz
├── 199003848TR
│   └── RawFq
│       ├── 199003848TR_1.fq.gz
│       └── 199003848TR_2.fq.gz
├── 199003850TR
│   └── RawFq
│       ├── 199003850TR_1.fq.gz
│       └── 199003850TR_2.fq.gz
└── 199003852TR
    └── RawFq
        ├── 199003852TR_1.fq.gz
        └── 199003852TR_2.fq.gz
```

`mergefq`程序会同时输出 sampleID→fq1→fq2 的对应关系输出到标准输出 (`stdout`)：

```
199003847TR    /path/to/199003847TR_1.fq.gz    /path/to/199003847TR_2.fq.gz
199003848TR    /path/to/199003848TR_1.fq.gz    /path/to/199003848TR_2.fq.gz
199003850TR    /path/to/199003850TR_1.fq.gz    /path/to/199003850TR_2.fq.gz
199003852TR    /path/to/199003852TR_1.fq.gz    /path/to/199003852TR_2.fq.gz
```

这些输出会由 WDL 的`read_tsv(stdout())`读取并保存成二维数组 (sm2fq) 作为任务的输出，作为后续依赖任务的输入。表 4 列出了 WDL 中常用的读取输入的命令。

Tab. 4: WDL 读取输入命令

命令	返回类型	说明
`read_lines(String|File)`	Array[String]	按行读取文件内容到字符串数组
`read_tsv(String|File)`	Array[Array[String]]	读取表格文件到二维字符串数组，各列按索引获取
`read_map(String|File)`	Map[String, String]	按 key→value 对读取两列的文件，返回字典结构
`read_int(String|File)`	Int	读取文件中的整型数
`read_string(String|File)`	String	读取整个文件内容到字符串
`read_float(String|File)`	Float	读取文件中的浮点数
`read_boolean(String|File)`	Boolean	读取文件中的布尔值

3.2.2 并行分析

若样本之间互相独立，则适合进行并行分析，WDL 中通过`scatter`实现。我们以对合并的 fq 进行质控的`fqqc`为例进行说明。`rnaseq`流程在执行`mergefq`后对其返回的二维数组进行遍历，每一个元素又是一个数组 (含 3 个元素，分别为 sampleID，fq1 和 fq2)，利用索引 (0, 1, 2) 访问这些元素。

```
workflow rnaseq {
    File list
    File fqdir
    File wkdir

    call mergefq {
        input:
            list = list,
            fqdir = fqdir,
            outdir = wkdir
    }

    scatter (line in mergefq.sm2fq) {
        call fqqc {
            input:
                sample = line[0],
                in_fq1 = line[1],
                in_fq2 = line[2],
                thread = 4,
                outdir = wkdir
        }
    }
}
```

以下为`fqqc`的内容，`output`返回的是过滤掉低质量的 fq1 和 fq2 文件。

```
task fqqc {
    File in_fq1
    File in_fq2
    File outdir
    Int thread
    String sample

    command {
        fastp \
            -i ${in_fq1} \
            -I ${in_fq2} \
            -o ${outdir}/${sample}/ClnFq/${sample}_1.fq.gz \
            -O ${outdir}/${sample}/ClnFq/${sample}_2.fq.gz \
            -h ${outdir}/${sample}/ClnFq/${sample}.html \
            -j ${outdir}/${sample}/ClnFq/${sample}.json \
            -w ${thread} &> ${outdir}/${sample}/ClnFq/${sample}.log
    }

    output {
        File fq1 = "${outdir}/${sample}/ClnFq/${sample}_1.fq.gz"
        File fq2 = "${outdir}/${sample}/ClnFq/${sample}_2.fq.gz"
    }
```

```
    runtime {
        docker: "liujc/os"
        cpu: "16"
        memory: "16G"
        retry: 1
    }
}
```

3.2.3 汇总分析

`scatter`中的各个样本在资源充足的情况下会并行完成分析，产生各自的分析结果，`scatter`则返回的是样本结果的数组 (图 7)。如果我们需要对这些结果进行汇总，则需要待`scatter`步骤完成后进行，这个过程在 WDL 中称为`gather`(图 8)。

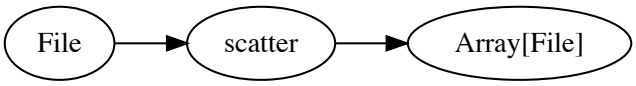

Fig. 7: `scatter`前后的类型变化

Array[] 往往需要转化成 String 后才能作为程序的输入。比如比对步骤`align`输出的是 bam 文件，经过`scatter`后返回的是一组 bam，我们可以做以下转换:

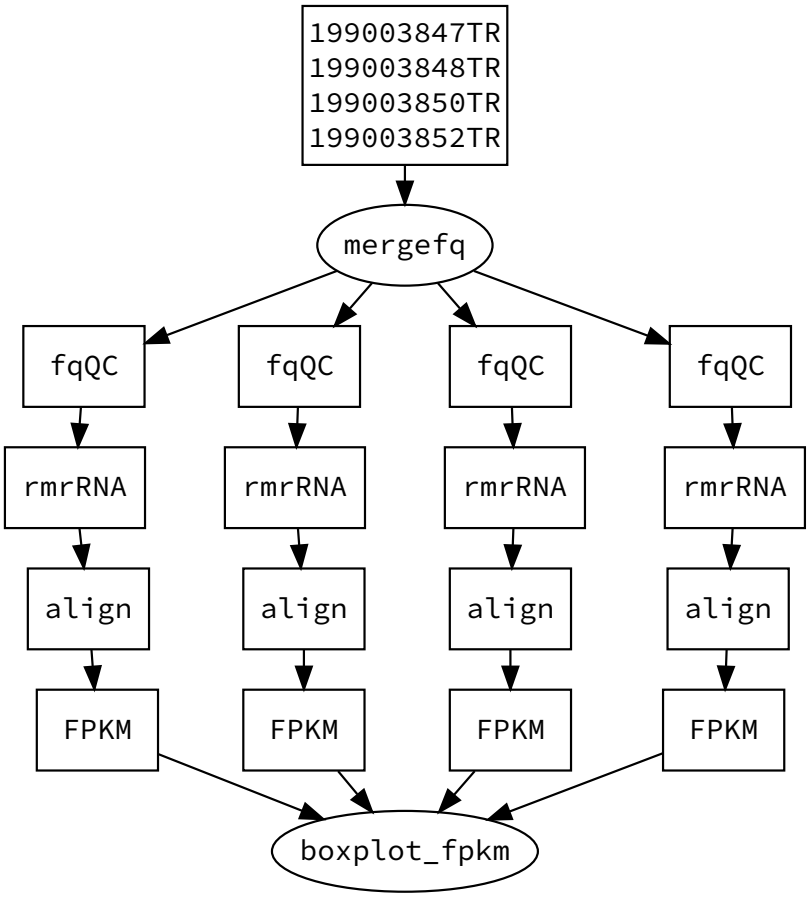

Fig. 8: RNA-Seq 流程中的`scatter`与`gather`

```
task align {
    ...
    output {
       File bam = ...
    }
    ...
}
workflow xxx {
    ...
    scatter {align...}
    Array[File] bams = align.bam
    String bam_list_space_seperated = ${sep=" " bams}
    String bam_list_comma_seperated = ${sep="," bams}
    String bam_list_for_picard = ${sep=" -I " bams}
    ...
}
```

3.2.4 流程可视化

WDL 流程可以经 WOMtool 转成 `.dot` 实现可视化，具体实现的命令如下:

```
java -jar womtool.jar graph rnaseq.wdl > rnaseq.dot
dot -Tsvg -o rnaseq.svg rnaseq.dot
#or using my graphviz docker
docker run --rm -v $PWD:/tmp liujc/graphviz dot -Tsvg -o /tmp/rnaseq.svg /tmp/rnaseq.dot
```

图 9 为 RNA-Seq 基础分析流程的 WDL 可视化图，展示了 WDL 的各个`task`之间的关系，方框内是流程按样本并行处理的部分（`scatter`），`boxplot_fpkm`则依赖所有样本的 FPKM 结果，待`scatter`步骤结束后开始执行。

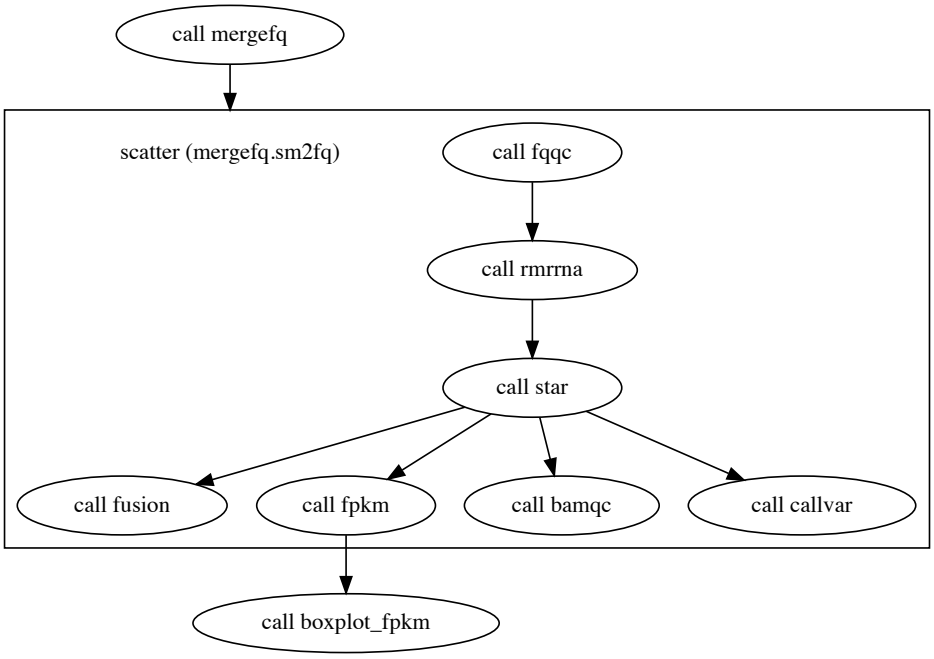

Fig. 9: RNA-Seq 分析流程图

除此之外，可以使用`wdl2cwl`（`pip install wdl2cwl`安装）将 WDL 转换成 CWL 格式，再使用 Rabix Composer 等工具进行可视化。cwl2wdl 工具可以进行反向转换。

3.2.5 注意事项

- WDL 的流程所用到的数据库文件均需明确指定，例如 bwa 比对我们需要指定参考序列 hg19.fa，但 bwa 在运行时会自动搜索并使用参考序列的索引文件 (hg19.fa.fai)，这个索引文件也要明确指定。
- 语法检查：使用 WOMtool (https://github.com/broadinstitute/cromwell/releases) 对写好的 WDL 流程进行验证:

```
java -jar womtool.jar validate rnaseq.wdl
```

- WDL `task`之间的依赖关系均需通过`output`指定。

4 Bioflow

XTAO Bioflow (以下简称 Bioflow) 是荣之联 (UEC) 公司旗下极道团队 (XTAO) 使用 Go 语言开发的生物信息分析任务调度系统。支持 BSL 和 WDL 两种流程开发语言。BSL 是极道团队基于 Bpipe 语法开发的流程语言，针对 DNA 突变检测做了许多贴心设计，方便流程开发。深度定制的 BSL 流程语言在开发其他流程是灵活度和可控性较 WDL 差一些，所以我们针对不同场景选择 BSL 或 WDL 进行开发。

4.1 常用命令

Bioflow 在权限管理方面设计了两组命令：管理员 (administer) 使用`bioadm`，普通用户 (client) 使用`biocli`。本文档仅介绍流程开发和测试相关的`biocli`命令`pipeline`(流程) 和`job`(作业) 并在

实例中有进一步说明。更详细的信息可以参阅【Bioflow 命令行使用手册】。

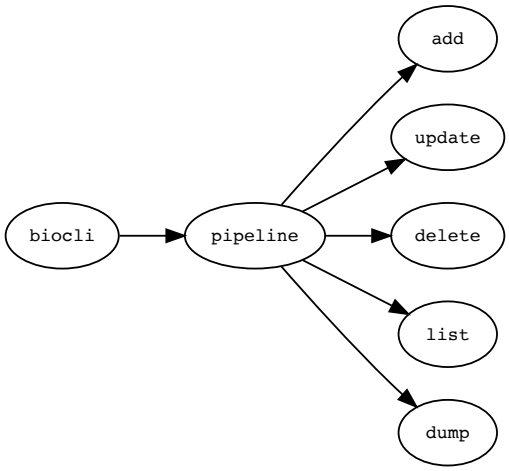

Fig. 10: `biocli pipeline` 命令

如图 (图 10, 11) 所示，`biocli` 命令采用三段式设计，名称很好地反映了命令功能，比如说添加流程的命令是 `biocli pipeline add` 。除了 `list` 外，这些命令还需要指定第四个参数，目前还不支持自动补齐命令，输入比较繁琐，我们可以通过设定快捷方式的方式改善输入体验，比如在 `~/.bashrc` 中添加以下快捷方式:

```
alias bpa="biocli pipeline add"
alias bpl="biocli pipeline list"
alias bpu="biocli pipeline update"
alias bjl="biocli job list"
alias bjs="biocli job submit"
alias bjS="biocli job status"
alias bjc="biocli job cancel"
```

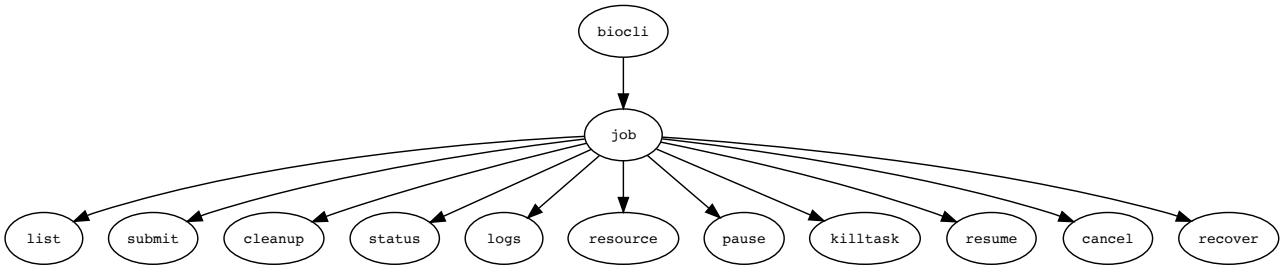

Fig. 11: `biocli job` 命令

`biocli` 命令的各段命令都可以通过 `--help` 获得说明，比如:

```
biocli job --help
biocli job --help-long
biocli job list --help
```

4.2 客户端设置

在用户首次登陆 Bioflow 集群后，有可能需要设置 `biocli` 客户端环境变量:

```
$ biocli env set server bioflow.marathon.mesos:1027
$ biocli env get
{"server":"bioflow.marathon.mesos:1027"}
```

配置文件自动保存在家目录: `~/bioflow_client.conf`。

4.3 流程实例

4.3.1 上传流程

WDL 格式的分析流程部署到 Bioflow 系统需要通过一个 json 文件实现，以 RNA-Seq 分析流程为例:

```
{
    "Name" : "rnaseq",
    "Type" : "WDL",
    "Description" : "A pipeline for RNA-Seq analysis",
    "WorkDir" : "annavol1@xtao:liujc/log",
    "Wdl" : {
        "WorkflowFile" : "rnaseq.wdl"
    }
}
```

其中`Name`为流程的唯一标识符，`Type`为固定的 WDL 格式，`WorkDir`为流程默认的工作目录，存放流程日志等相关文件，`Wdl`部分指定了流程的核心文件`WorkflowFile`的具体位置。通过以下命令上传到 Bioflow 流程管理系统，其中`./wdl`路径下含有 WDL 流程文件`rnaseq.wdl`。

```
biocli pipeline add -d ./wdl rnaseq.json
```

`WorkDir`目录也可以在 job 实例中指定。在流程中指定的好处是可以统一存放日志并定期清理。上传的流程会放置在 Bioflow 的`pipelinestore`文件夹内，如果误删了本地的流程，可以从这个文件夹中将对应版本的流程拷回来，或者是通过以下命令自动实现:

```
biocli pipeline dump -v 12 rnaseq rnaseq_v12.json
```

如此，版本 12 的 RNA-Seq 流程会输出到文件`rnaseq_v12.json`，流程的 WDL 等文件会放到`rnaseq-12`文件夹。

4.3.2 查看流程

使用`biocli pipeline list`查看已有的 Bioflow 流程，不便之处就是会显示所有的流程，没有只显示当前用户流程的参数选项。如果我们知道流程的名字，可以通过以下命令查看该流程信息:

```
$ biocli pipeline info rnaseq
Pipeline RNASEQ:
   Type: WDL
   Items: 0
   State: InUse
   WorkDir:
   IgnoreDir:
   Parent: RNASEQ
   LastVersion: 55
   Version: 56
   Description: A pipeline for RNAseq analysis
   Owner: liujc
   WorkflowFile: rnaseq.wdl
   Items:
```

4.3.3 更新流程

更新后的流程通过 update 命令提交到 Bioflow，流程的版本号自动更新。

```
biocli pipeline update -d ./wdl rnaseq.json
```

4.4 作业实例

4.4.1 提交作业

Bioflow 作业投递需要通过指定输入/输出/参数和流程的 json 文件实现，json 的基本格式如下:

```
{
    "Name" : "rnaseq-run",
    "Pipeline" : "rnaseq",
    "WorkDir" : "annavol1@xtao:liujc/test/wdl/rnaseq/log",
    "InputDataSet" : {
        "WorkflowInput" : {
            "rnaseq.list" : "annavol1@xtao:liujc/test/wdl/rnaseq/sam2lib.txt",
            "rnaseq.fqdir" : "annavol1@xtao:liujc/Org-Gene/Esophageal_Cancer_RNA-Seq",
            "rnaseq.wkdir" : "annavol1@xtao:liujc/test/wdl/rnaseq/run",
    ...
            "rnaseq.ref_fa" : "annavol1@xtao:liujc/db/ref/hs37d5.fa",
            "rnaseq.ref_fai" : "annavol1@xtao:liujc/db/ref/hs37d5.fa.fai"
        }
    },
    "Priority" : 10
}
```

其中`Name`为本作业的名字，`Pipeline`为使用的流程的名字 (对应流程的唯一标识符)，`WorkDir`为存放流程日志的目录 (也可以不指定,使用流程默认的工作目录),对于 WDL 流程来说,`InputDataSet`内的`WorkflowInput`用来指定流程的相关参数 (格式为`流程名`.`参数`)，`Priority`为流程的优先级，从 1 到 10，最高优先级为 10。其中的`WorkflowInput`部分可以通过 WOMtool 生成模版后 (5.1.2) 修改得到。json 文件格式特点是每个部分的最后一项行末是没有逗号的，使用时需要注意。

投递作业的命令如下:

```
$ biocli job submit job.json    # or bjs job.json
The job added success, job ID is: 69c2778d-a36a-4190-5934-4c7bb6703eea
```

4.4.2 查看作业

查看正在运行的任务:

```
$ biocli job list    # or bjl
1 running jobs:
 Job 1:
    ID: 69c2778d-a36a-4190-5934-4c7bb6703eea
    Name: tcr-run
    Pipeline: TCR
    Created: 2019-08-09T09:59:57+08:00
    Finished: N/A
    State: RUNNING
    Owner: liujc
    Priority: 10
```

查看任务运行状态:

```
$ biocli job status 4c7bb6703eea
Status of Job 69c2778d-a36a-4190-5934-4c7bb6703eea:
 Name: tcr-run
 Pipeline: TCR
 State: RUNNING
 Owner: liujc
 WorkDir: annavol1@xtao:liujc/test/wdl/rnaseq/tcr/log
 ...
```

Bioflow 对作业的各个步骤 (stage) 的运行时间有记录,通过 `biocli job status 4c7bb6703eea` 查看，以 RNA-Seq 流程中的 TCR 分析为例，其比对历时 32 分钟，是从提交时间开始算的。我们可以通过计算 `运行结束时间` 和 `开始运行时间` 的时间差来得到该步骤的执行时间，这个是我们在测试流程性能时最常用到的。

```
Stage tcr-tcr-Scatter-3-main-tcr.align-shard2:
    Submited:  Fri Aug  9 09:59:57 CST 2019 # 提交时间
    Scheduled: Fri Aug  9 10:00:07 CST 2019 # 开始运行
    Finished:  Fri Aug  9 10:32:36 CST 2019 # 运行结束时间
    Duration: 32.650000                     # 历时(单位为min)
```

Bioflow 定义了以下几种作业状态:

`CREATED` 作业刚提交，尚未调度执行

`RUNNING` 作业正在运行。作业在执行流程的过程中，流程的每个阶段都会提交一个或多个任务到后端执行。这些阶段可能暂时没有资源，不能运行，处于等待状态。但是作业仍然会处于 RUNNING 状态。

`FINISHED` 作业执行完成，没有异常退出。如果流程有很多个阶段，表示每个阶段都正常完成。

`CANCELED` 作业被用户取消，作业正在执行且未完成的所有阶段的任务都将会被结束。

`PAUSED` 作业被暂停。作业将记住暂停时候的状态，如果用户恢复作业，作业将被恢复为暂停时候的状态继续执行。注意暂停作业时，作业产生的正在运行的任务不会被停止。但是作业将停止调度新的阶段和任务。

`FAIL` 作业的一个或者多个阶段执行失败。

`PSUDONE` 作业执行部分完成，同时由于某些阶段的失败造成作业无法继续执行的结束状态。

4.5 Logs

Bioflow 作业 jsob 文件中的 `WorkDir` 指定的是作业的工作目录，是存放 Bioflow 信息和流程日志的地方，不同于分析流程的结果输出文件，通常指定到独立的文件夹。Bioflow 的日志目录结构如下，其中的 `boltstore.db` 是 Bioflow 记录流程信息的数据库文件， `wdllibiofiles` 为流程是 WDL 时存放文件的地方，暂时未发现里面有内容。 `*-shard#/` 文件夹中是标准输出和标准错误两个文件。

```
.
├── boltstore.db
├── logs
│   ├── WORKFLOW.TASK-paladin-task.JOBID.cmd_info
│   ├── WORKFLOW.TASK-paladin-task.JOBID.err
│   ├── WORKFLOW.TASK-paladin-task.JOBID.info
│   └── WORKFLOW.TASK-paladin-task.JOBID.profiling
├── PIPELINE-WORKFLOW-Scatter-#-main-WORKFLOW.TASK-shard#
│   ├── stderr
│   └── stdout
└── wdllibiofiles
```

logs 文件夹中有 4 种后缀名的文件，其功能描述如下。

`.err` 记录每个任务的实际运行命令，其中的文件路径（`vol@xtao`）已经替换为实际路径。此信息可以使用命令 `biocli job status JOB_ID -d` 获得（–d 表示显示任务细节 detail）。
`.cmderr` 记录任务的标准错误信息
`.info` 记录容器和和任务信息
`.profiling` 性能剖析信息 (profiling information)

这些文件所含的信息可以使用 `biocli job logs JOB_ID` 命令显示。

需要注意的是如果任务由于输入的问题而失败后，可以修正输入，然后通过 `biocli job recover JOB_ID` 从出错的地方开始重分析。如果需要从头开始分析的话，需要删掉或清空 `WorkDir` 再使用 `biocli job submit job.json` 投递任务，不删除 `WorkDir` 的话，投递任务会一直是 `CREATED` 状态，不会运行，直至 timeout 退出。

4.6 XTAO 存储目录

在作业中指定流程实际参数时我们使用了诸如 `annavol1@xtao:liujc/db/ref/hs37d5.fa` 样式的路径格式，没有使用绝对路径。这是 XTAO 存储的推荐路径格式，其中的 `annavol1@xtao:` 会在运行时转换成实际路径 `/mnt/annavol1` 或其他实际挂载的存储位置，这样的好处是当挂载的存储发生变动时，流程不会受影响。

5 备选调度工具

5.1 Cromwell

Broad Institute 为 WDL 流程开发了调度引擎 Cromwell(在线文档, 软件下载地址) 和 WDL 工具 WOMtool (Workflow Object Model (WOM))。Cromwell 支持多种后端任务管理系统 (详细信息)，在未部署 Bioflow 工具的集群可以通过 Cromwell 来进行作业调度。

5.1.1 流程验证

同 WDL 的语法检查 (3.2.5)

5.1.2 输入准备

```
$ java -jar womtool.jar inputs rnaseq.wdl > rnaseq_inputs.json
```

```
$ cat rnaseq_inputs.json
{
  "rnaseq.wkdir": "File",
  "rnaseq.fqdir": "File",
  "rnaseq.list": "File",
  "rnaseq.ref_fa": "File",
  "rnaseq.ref_fai": "File",
  ...
}
```

然后将 `rnaseq_inputs.json` 中的参数全部替换为真实的数据并保存。

5.1.3 本地运行

Cromwell 支持 WDL 在本地运行，如安装了 Docker 的个人电脑，OncoBox 一体机等。运行分析任务的命令如下:

```
java -jar cromwell.jar run rnaseq.wdl --inputs rnaseq_inputs.json
```

需要注意的是 Cromwell 不支持 XTAO 的存储映射 (vol@host) 结构，也暂时不支持嵌套的`scatter`结构 (可以通过 sub workflow 的方式实现)。

5.1.4 SGE 集群

Cromwell 支持 WDL 流程在 SGE 集群 (SGE) 运行，需要 SGE 配置文件，下面是网上找到的一个例子。backend 的设置参数不少，这部分待整理。

- java 命令:

```
java -Dconfig.file=sge.conf -jar cromwell.jar run hello.wdl
```

- hello.wdl:

```
task hello { command { echo "Hello world" } }
workflow wf_hello { call hello }
```

- sge.conf (tested working):

```
# Include the application.conf file.
include required(classpath("application"))
backend {
  # Switch the default backend to "SGE"
  default = "SGE"
  providers {
    # Configure the SGE backend
    SGE {
      # Use the config backend factory
      actor-factory = "cromwell.backend.impl.sfs.config.
    ConfigBackendLifecycleActorFactory"
      config {
          #run-in-background = false
        # Limits the number of concurrent jobs
        concurrent-job-limit = 500
        # Define runtime attributes for the SGE backend.
        # memory_gb is a special runtime attribute. See the cromwell README for more
    info.
        runtime-attributes = """
          Int cpu = 1
          Float? memory_gb
          String? sge_queue
          String? sge_project
          String? docker
        """
```

```
        # Script for submitting a job to SGE, using runtime attributess.
        submit = """
          qsub \
          -terse \
          -V \
          -b n \
          -N ${job_name} \
          -wd ${cwd} \
          -o ${out}.qsub \
          -e ${err}.qsub \
          -pe mpi ${cpu} \
          ${"-l h_vmem=" + memory_gb + "g"} \
          ${"-q " + sge_queue} \
          ${"-P " + sge_project} \
          ${script}
        """
        # See the cromwell README for more info.
        submit-docker = """
          qsub \
          -V \
          -wd ${cwd} \
          -N ${job_name} \
          -o ${out}.qsub \
          -e ${err}.qsub \
          -l ${"vf=" + memory_gb + "g"},p=${cpu} \
          -b y docker run -v ${cwd}:${docker_cwd} ${docker} /bin/bash ${docker_script}
        """
        # command for killing/aborting
        kill = "qdel ${job_id}"
        kill-docker = "docker stop ${docker_cid}"
        # Command used at restart to check if a job is alive
        check-alive = "qstat -j ${job_id}"
        # How to search the submit output for a job_id
        job-id-regex = "(\\d+)"
      }
    }
  }
}
```

5.1.5 Cromwell 的不足之处

经过测试，发现 Cromwell 有以下不足:

1. Cromwell 支持任务从失败处重启分析,但必须保持其`cromwell-executions`⟶`workflow`⟶`task`样式的目录结构，而且目录中包含为了区分不同分析的哈希值，导致分析的实际路径是事先不可知的。
2. 如果想做成 BNC 分析结果那样的文件结构，涉及到大量文件拷贝，查看官方文档后得知此问题暂时无解。
3. Cromwell 通过 SGE 或其他后端投递任务，任务的状态获取需要额外的开发。

5.2 Toil

除了 Cromwell，UCSC 计算基因组实验室使用 Python 开发的Toil也支持WDL 流程调度，具有易学、稳定和高效等特点。对于熟悉 Python 的朋友来说是个不错的任务调度方案。

5.2.1 安装 Toil

```
pip install 'toil[wdl]'
```

5.2.2 创建流程

- WDL 流程 (wdl–helloworld.wdl)

```
workflow write_simple_file {
    call write_file
}
```

```
task write_file {
    String message
    command { echo ${message} > wdl-helloworld-output.txt }
    output { File test = "wdl-helloworld-output.txt" }
}
```

- 作业文件 (wdl–helloworld.json)

```
{
    "write_simple_file.write_file.message": "Hello world!"
}
```

5.2.3 运行流程

与 Cromwell 相似，toil 运行任务需指定流程 wdl 文件和任务实例 json 文件:

```
toil-wdl-runner wdl-helloworld.wdl wdl-helloworld.json
```

5.2.4 Toil 总结

经过测试，我们发现 Toil 有如下优缺点:

1. Toil 系统基于 Python 开发，易安装和维护，开发 wraper 程序和实现并行化，灵活性较好
2. 测试单机版任务调度通过，可用于 OncoBox 任务调度
3. 对 HPC(SGE) 的支持还处于 alpha 版本阶段

综上，不在吉云平台中使用。

6 总结

吉云平台的流程涉及到 Bioflow，WDL 和 Docker，通过 json 文件建立关联，三者密不可分 (图 12)。WDL 流程定义了分析过程和所用的 Docker，Bioflow 系统则负责 WDL 流程的执行调度，对指定的作业实例进行分析。

TODO: 更新关联图，将 Cromwell 独立出来，加入 SGE

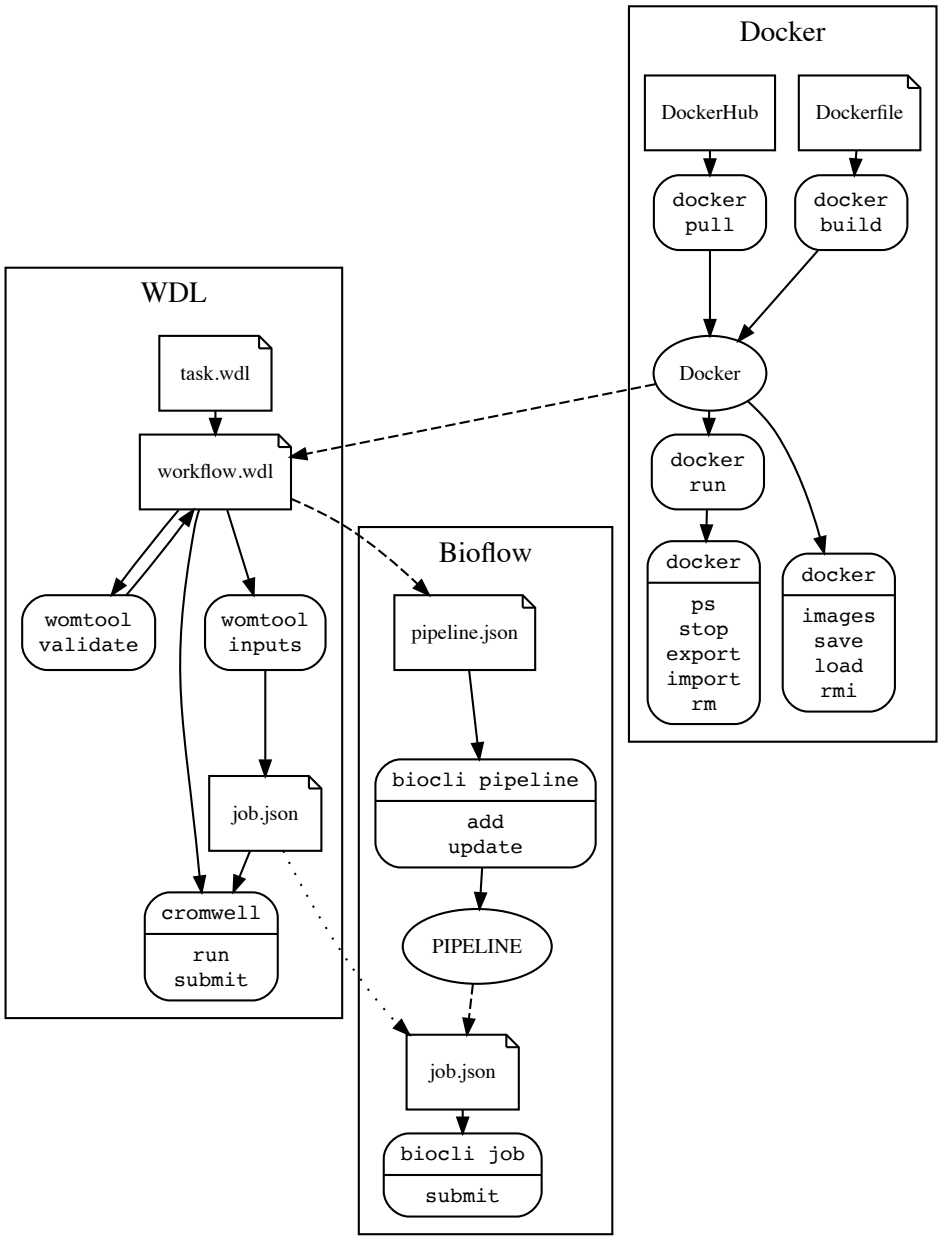

Fig. 12: WDL–Docker–Bioflow 关系图

Todo list