

This is your **last** free member-only story t


Using Docker in genomics workflows




Pubudu samarakoon [Follow](#)

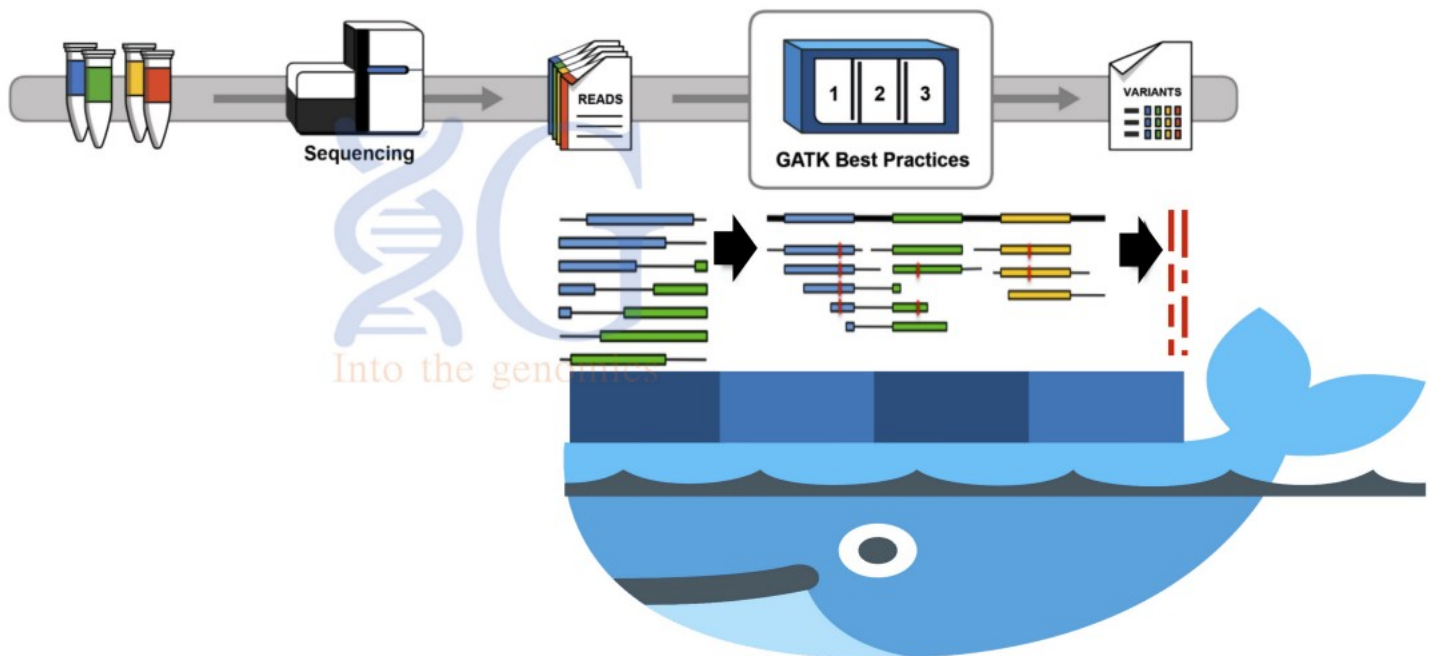
Oct 29, 2019 · 6 min read ★

Sign in to medium.com with Google

 **Andrew Judell**
andrew.s.judell@gmail.com

 **Andrew Judell**
andy.judell@gmail.com

1 more account



Why use docker in genomics workflows?

Genomics workflows implement a broad range of programs that have many dependencies. Typically, many different research groups develop these programs, and they are updated independently of one another. Therefore using such applications in a workflow can lead to various challenges. Importantly, significant issues in genomics workflows are related to their portability, reproducibility and conflicts in required dependencies. Docker provides an effective solution to address these issues.

In this article, I'll first provide an overview of the docker technology that includes a brief introduction to docker, docker concepts and a set of frequently used docker commands. Next, I'll demonstrate the utility of docker technology in DNA sequence analysis using BWA, samtools, Picard and GATK3 docker images.

What is docker?

Docker is a widely-used container technology that allows users to bundle software into standardised packages. In other words, docker allows users to create an isolated copy (instance) of one or more applications, including their dependencies and deploy them on a host system.

Docker is a type of container. “A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.”

Docker is not a virtual machine (VM). A VM can isolate an entire operating system (OS) and deploy as guest systems. Therefore multiple VMs can deploy multiple guest systems on a host. However, since a single VM contains a complete copy of the guest OS, implementing multiple VMs on a host is a resource-intensive task. In contrast to VMs, docker only packages the code (of an app) along with dependencies and enables them to run them on a host. When deploying multiple containers on a single host, they can

share the host OS kernel. Therefore Docker containers require less computational resources than VMs.

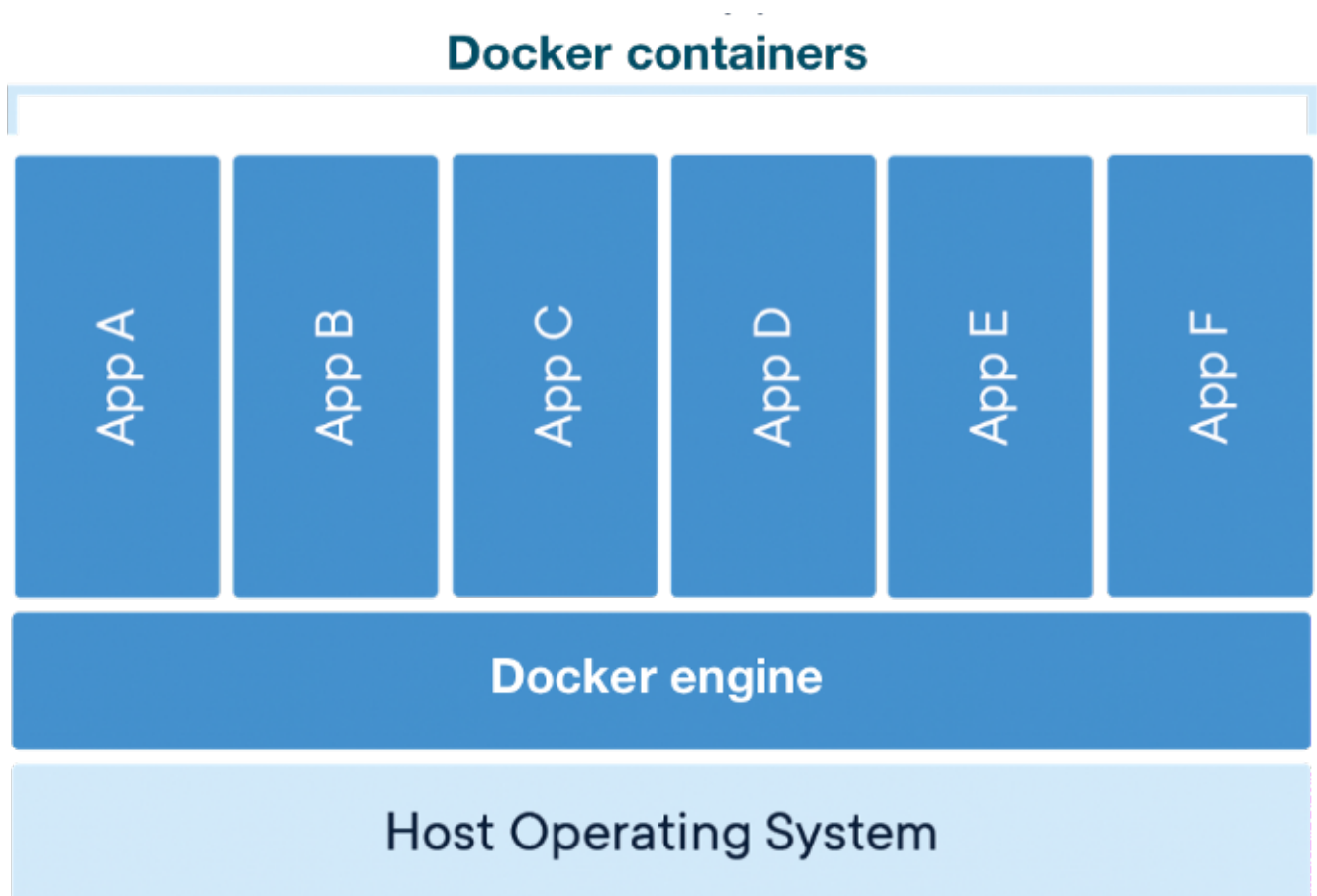
Why we need docker?

Docker provides a single solution to multiple key issues

- Lack of isolation of processes within a system
- Higher demand for resources when implementing multiple isolated environments
- Issues related to reusability of application in different operating systems

In essence, docker enables users to develop and test applications locally, and ship them to any other computer, cluster or cloud in a way that guarantees the same behaviour.

Docker concepts



[source image](#)

Docker has three main components — Docker engine, containers and images.

Docker engine

- Docker engine lives on the host computer and can carry many containers
- Docker engine manage resources accordingly

Docker containers

- Containers enable to run processes by exposing them to the host OS
- A single container can have a single isolated process (or multiple processes on rare occasions)

Docker images

- A Docker image is a file with all the instructions needed to execute an application in a Docker container. Therefore docker images are used to initiate docker containers.
- You can use any Docker image in docker-engines independent to the underline OS
- By creating a docker image, you can improve the portability of your application. i.e., You can transfer a docker image into a docker engine that runs on a different OS and expects the same behaviour.

How to build a docker container

The process of building a docker container in three main steps.

- Step 1: Create the Dockerfile that provide instructions to build the container
- Step 2: Use docker build command to pass the Dockerfile to docker engine. Docker engine then build a docker image that helps run the specified process
- Step 3: Use docker run command to start a container using a docker image. Docker container initiated in this step runs alongside with other containers in the Docker engine.

The blog-post — A fast and easy Docker tutorial for beginners (video series) provides a well described overview of the docker technology

Useful Docker commands

Check the docker version

```
> docker --version
```

Download (pull) docker images

```
> docker pull <IMAGE NAME>:<TAG>
```

List docker images

```
> docker images
```

Build a docker image - docker build looks for the dockerfile and the "context" in the given path

```
> docker build .
```

Create a Docker container - create the container and print the containerID to STDOUT (do not start the container)

```
> docker create
```

List Docker containers

```
> docker ps -a # list all
```

```
> docker ps # list running containers
```

```
> docker ps --filter "status=exited" # stopped containers
```

Start and run a Docker container (docker create + docker start)

```
> docker run [ OPTIONS ] IMAGE[:TAG] [COMMAND] [ARG...]
```

docker run options

-i: Starts the container in interactive mode.

-ti: Run shell commands inside a particular container

-name: User friendly name for the container. If the name is not specified, random string will be assigned as the container name.

-v <path to host directory>:<path to container directory>: map host directory to specified directory in the container

-rm: remove the container after executing the command

-p: map ports in the host computer to the Docker container

Executing an initiated container

```
> docker container exec
```

Stop a running a Docker Container

```
> docker stop [-t | --time[=10]] CONTAINERID [CONTAINER...]  
> docker stop $(docker ps -q) ## stop all containers
```

Kill all running containers and clean the memory

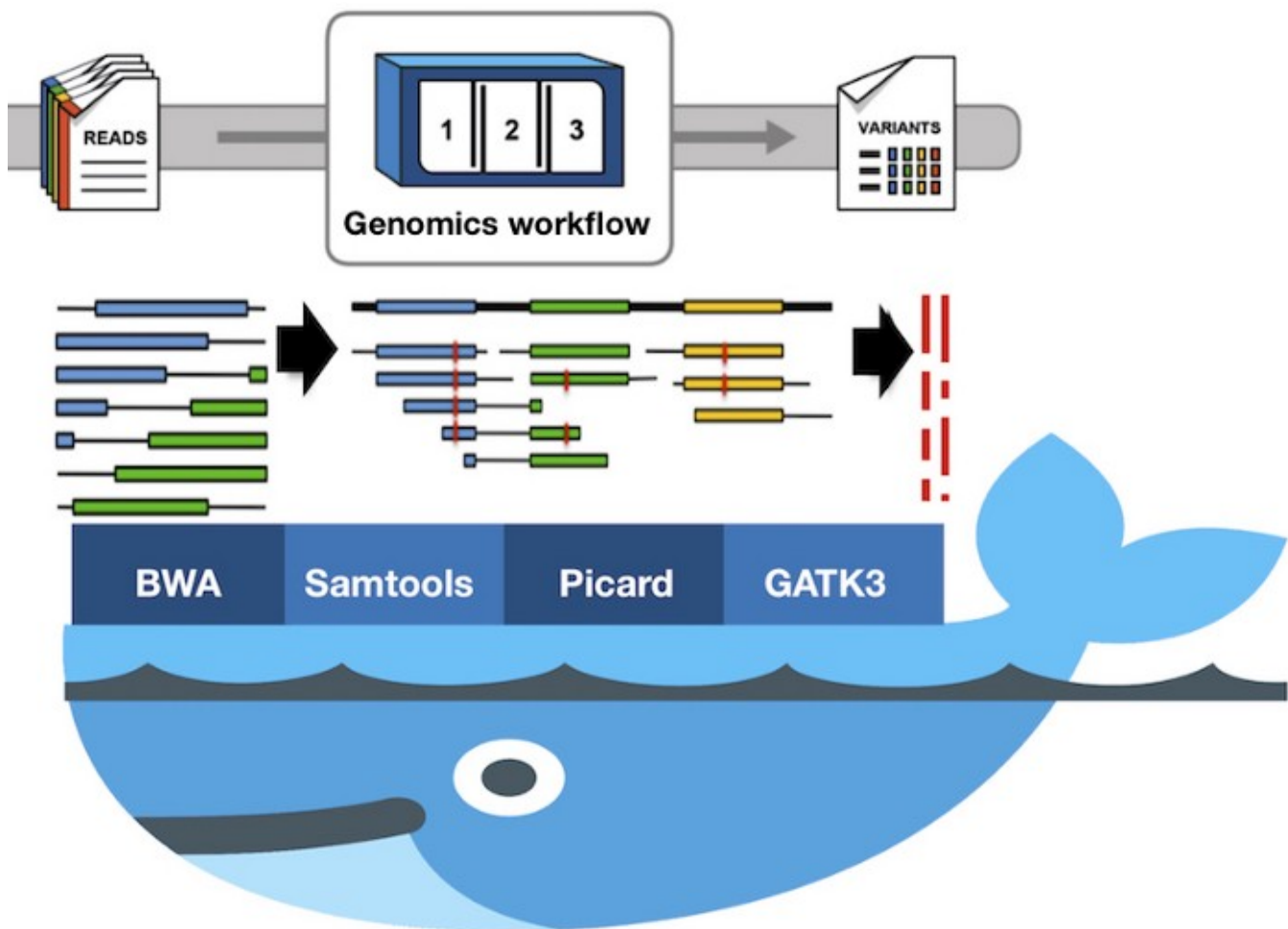
```
> docker kill $(docker ps -q)
```

Deleting a Docker container (Stop the container before deleting)

```
> docker rm [ OPTIONS ] CONTAINER [ CONTAINER ]
```

Implementing genomics workflows using docker

Here, I'll use a test dataset and implement a basic genomics workflow to highlight the use of docker technology in DNA sequence analysis.



Sequence analysis and variant calling workflow using BWA, Samtools, Picard and GATK3 docker images

- Input

7859_GPI.read1.fq and 7859_GPI.read2.fq FASTQ files
- from GATK google drive folder – tutorial_7859.tar.gz

- Reference sequence

chr19_KI270866v1_alt
- two contigs from human GRCh38/hg38: chr19

Download docker images and initiate relevant containers

- BWA

```
> docker pull biocontainers/bwa:v0.7.17-3-deb_cv1
```

- Samtools

```
> docker pull biocontainers/samtools:v1.7.0_cv3
```

- Picard

```
> docker pull biocontainers/picard:v2.3.0_cv3
```

- GATK

Download latest GATK release

```
> docker pull broadinstitute/gatk
```

Download a specific GATK release

```
> docker pull quay.io/biocontainers/gatk:3.6-7 ## specific GATK
release
```

Run and initiate the gatk_3.6 image and follow GATK3 setup (download a licensed copy of GATK from the Broad Institute and copy GATK into your conda environment)

```
docker run --name gatk_3.6 -it -v <host directory>:/data <IMAGE ID>
docker container exec -it gatk_3.6 bash ## Follow GATK3 setup
```

Run basic steps in a sequence analysis workflow using docker

- Indexing the reference genome

Initiate bwa container... and

```
> docker run --name bwa_v0.7.17 -it -v <host directory>:/data <IMAGE
ID> bash
```

Index the reference

```
> bwa index chr19_KI270866v1_alt.fasta
# Create BWA index with
    *fasta.amb,
    *fasta.ann,
    *fasta.bwt,
    *fasta.pac,
    *fasta.sa files
```

Create *fasta.fai and *dict indexes for GATK analysis

```
> docker run --name samtools_v1.7 -it -v <host directory>:/data
samtools faidx chr19_KI270866v1_alt.fasta
```

```
> docker run -v <host directory>:/data --rm <IMAGE ID> picard
CreateSequenceDictionary R=chr19_KI270866v1_alt.fasta
O=chr19_KI270866v1_alt.dict
```

- Read-mapping using BWA mem

Execute BWA container... and

```
> docker container exec -it bwa_v0.7.17 bash
```


Align *fq files to the reference

```
> bwa mem -R '@RG\tID:t1\tSM:t1' chr19_KI270866v1_alt.fasta
7859_GPI.read1.fq 7859_GPI.read2.fq > 7859_GPI.aln_pe.sam
# Created 7859_GPI.aln_pe.sam file
```

- Sorting SAM file, marking duplicated reads and indexing BAM files

Sort sam file and create a bam file, then index the bam file

```
> docker run -v <host directory>:/data --rm <IMAGE ID> picard
SortSam INPUT=7859_GPI.aln_pe.sam OUTPUT=7859_GPI.aln_pe.bam
SORT_ORDER=coordinate
> docker run -v <host directory>:/data --rm <IMAGE ID> picard
BuildBamIndex VALIDATION_STRINGENCY=LENIENT I=7859_GPI.aln_pe.bam
```

Mark duplicated reads in the indexed bam file and create a new marked-bam file

```
> docker run -v <host directory>:/data --rm <IMAGE ID> picard
MarkDuplicates VALIDATION_STRINGENCY=LENIENT AS=true
REMOVE_DUPLICATES=true I=7859_GPI.aln_pe.bam
O=7859_GPI.aln_pe.md.bam M=7859_GPI.aln_pe.md.metrics
```

Index the marked-bam file

```
> docker run -v <host directory>:/data --rm <IMAGE ID> picard
BuildBamIndex VALIDATION_STRINGENCY=LENIENT I=7859_GPI.aln_pe.md.bam
```

- GATK3 analysis and calling variants

Execute GATK3 container... and

```
> docker container exec -it gatk_3.6 bash
```

Define intervals to target for local realignment

```
> java -jar GenomeAnalysisTK.jar -T RealignerTargetCreator \
-nt 4 -R chr19_KI270866v1_alt.fasta -I 7859_GPI.aln_pe.md.bam \
-o 7859_GPI.aln_pe.intervals
```

Perform local realignment of reads around indels

```
> java -jar GenomeAnalysisTK.jar -T IndelRealigner \  
-R chr19_KI270866v1_alt.fasta -I 7859_GPI.aln_pe.md.bam \  
-targetIntervals 7859_GPI.aln_pe.intervals -o  
7859_GPI.aln_pe.md.bam.realigned.bam
```

Call variants using HaplotypeCaller

```
> java -jar GenomeAnalysisTK.jar -T HaplotypeCaller \  
-R chr19_KI270866v1_alt.fasta -ERC GVCF \  
-I 7859_GPI.aln_pe.md.bam.realigned.bam -o  
7859_GPI.aln_pe.md.bam.realigned.g.vcf
```

In summary, this workflow uses multiple docker images that take *.fastq files and a reference genome file, and create a *.vcf file with a list of variants.

[Docker](#)[Bioinformatics](#)[Sequence Analysis](#)[Mutation](#)[Genomic Concepts](#)[About](#) [Help](#) [Legal](#)

Get the Medium app

