Genomic data processing with GenomeFlow

Junseok Park[1,2], Eduardo A. Maury[3,4], Changhoon Oh[5], Donghoon Shin[6], and Alice Eunjung Lee[1,2]

[1] Division of Genetics and Genomics, Boston Children's Hospital, 3 Blackfan Circle, Boston, MA 02115 USA
[2] Department of Pediatrics, Harvard Medical School, 25 Shattuck St, Boston, MA 02115 USA
[3] Bioinformatics Integrative Genomics Program and Harvard/MIT MD-PHD Program, Harvard Medical School, Boston, MA, USA
[4] Program in Medical and Population Genetics, Broad Institute of MIT and Harvard, Cambridge, MA, USA
[5] Graduate School of Information, Yonsei University, 50 Yonsei-ro Seodaemun-gu, Seoul 03722 Republic of Korea
[6] Dept. of Human Centered Design and Engineering, University of Washington, Seattle WA, USA

* ealice.lee@childrens.harvard.edu

Abstract

Continuous development of genomics data analysis technologies and expansion of computing storage drive the generation of massive amounts of sequence data, which researchers can share and access through publicly open repositories. On-demand infrastructure services on cloud computing platforms support the processing of such large-scale genomics sequence data in distributed processing environments and can be used to reduce the time of analysis. However, parallel processing methods on cloud computing platforms still present a host of problems for the average user. In particular, cloud computing technology can be difficult to understand when designing an infrastructure suitable for a pipeline, and there is a risk that costs may increase exponentially if computing resources are not properly allocated. To overcome these challenges, we developed an automated infrastructure development and resource optimization program called GenomeFlow, a tool that is able to process large-scale samples at a minimal cost. Here, we describe the step-by-step protocol to use GenomeFlow according to a general sample processing scenario. We introduce the protocol for a bioinformatician with no experience in cloud computing and large data processing, which we estimate will take about 4-5 hours to execute.

Introduction

The past decade in genomics has been characterized by the creation of large amounts of next-generation sequencing data across a range of health and disease conditions [1]. Resources like The Cancer Genome Atlas (TCGA), International Cancer Genome Consortium (ICGC), Genotype-Tissue Expression (GTEx), Trans-Omics for Precision Medicine (TOPMED), and UK Biobank, among others, provide a multi-omic characterization of hundreds of thousands

of individuals at unprecedented resolution [2-5]. These resources have been made readily available for analysis through cloud computing infrastructure. Private pharmaceutical companies such as GSK and Regeneron are investing hundreds of billions of dollars to mine these large-scale genomic data to gain key insights into disease prevention, detection, diagnosis, and treatment [6]. While these investigations into therapeutic avenues have been successful, analyzing available data at scale to derive insights into basic biology remains largely unexplored, in part due to this avenue not being the primary focus of private pharmaceutical companies. In addition, the current cost for independent labs at medical and academic institutions to analyze these data poses a great barrier to such research inquiry.

To ameliorate the burden of costs associated with the deployment of algorithms and computational workflows in a cloud infrastructure for basic biology academic labs, a handful of platforms have been developed. For example, Tibanna is a tool that uses Amazon Web Service (AWS) as a cloud service provider, and it allows users to define and use a scalable portable pipeline based on Common Workflow Language (CWL) or Workflow Description Language (WDL) [7]. Another platform, Terra, provides genomic data analysis interfaces via user-defined WDL on Google Cloud Platform (GCP) [8, 9]. Despite their utility and uptake, these tools still require multiple rounds of trial and error to select the most cost-effective computational resource parameters, which can quickly become too costly, especially when trying to deploy a novel algorithm.

We therefore developed a tool that provides a resource-optimized scalable genomic pipeline in GCP architecture. Specifically, we developed GenomeFlow, which uses scheduled task processing with a failover function and implements a resource optimizer to reduce cloud billing on massively parallel jobs with scalable cloud computing architecture [10]. GenomeFlow supports Python and uses WDL for its workflow design. The user-defined pipeline runs on the Kubernetes cluster using an Airflow inspired controller and consists of a task scheduler to manage the status of jobs and individual tasks with web-based log check interfaces on a MySQL database [11-13]. We ran the tool on several use cases and recorded its performance (Table 1).

Here, we describe two use cases to provide users with a reference pipeline. We additionally present detailed guides for using GenomeFlow for general sample processing. A user without expert knowledge of cloud computing will be able to use GenomeFlow to process thousands of samples on a distributed pipeline at a low cost.

Table 1. Summary of GenomeFlow use cases.

Research Title	Research Description	Number of samples	Cost per sample (avg)	Processing time per sample (avg)	Total Cost
rTEA	To detect transposon-fu	15,477	$1.94	2.9h	$28,519.92

	sions in RNA sequencing data and perform a pan-cancer analysis				
AD Transposon	To find the impact of transposable elements in Alzheimer's disease	888	$3.76	9h	$3,343.81
ROSEMAP-TE	To check the preliminary results for transposable element expression in brain regions	1,355	$2.66	6h	$3,362.4
Epilepsy	To find genetic evidence that supports whether any new unknown gene candidates may cause epilepsy by comparing cases vs controls in Epi25k epilepsy cohort	30,789	$0.1177	30min	$4095.3

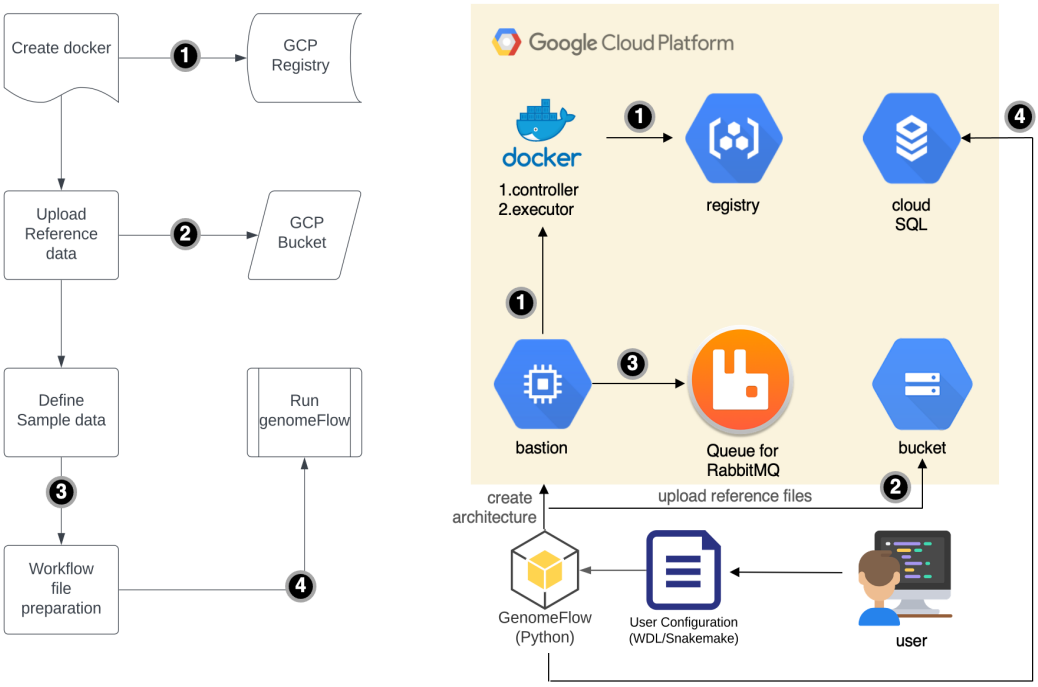
Overview of the Protocol

The main protocol of GenomeFlow is composed of two parts. The first is the distributional processing architecture design [11]. GenomeFlow is a tool that distributes samples to user-defined pipelines using cloud computing resources. The second is the user-defined workflow. The workflow contains a pipeline. It defines how to analyze input samples by combining necessary tools. A user-developed Dockerfile is also necessary and the file should contain an operational environment for the tools [14]. The Dockerfile is fed as input for the workflow. The input sample list file should contain all the sample IDs, and the user should define

the location where the samples will be downloaded as well as the tool to use for download. The controller of GenomeFlow passes a sample ID to each task in a job, and the user must define the commands actually executed in the task in the workflow. Users can define a single step or multiple steps for a task. Tasks with multiple steps can use different computing resources to optimize resource usage. To obtain the optimization parameters, the user can use the resource optimizer of GenomeFlow. In the case of downloading reference files, a user needs to upload their files to the Cloud Bucket and define the location of the workflow file. Finally, defining the output location of the Cloud Bucket is essential. GenomeFlow will store the result of each sample in the folder name in the defined result location.

The workflow of GenomeFlow is configured according to the user-defined command and parameter set, so it is not limited to a specific kind of analysis. Any pipeline should work as long as the user-generated Docker file and command set are properly configured (Figure 1). In this paper, we show two case reports for a more comprehensive overview, and we present a general scenario.

Figure 1. General protocol of GenomeFlow. Users need to provide a workflow file using WDL or Snakemake syntax. Then, they can load the GenomeFlow library from Python and run it. (A) 1. Users need to create a Dockerfile, after which GenomeFlow will create a Docker image and place the image into the Google Cloud Platform (GCP) registry to deploy it on Google Kubernetes Environment (GKE) 2. Users need to upload the reference data to the bucket and set its location in the workflow file. 3. Sample ID file preparation 4. All the commands in the user-defined pipeline should be set in the workflow. (B) 1. GenomeFlow creates a bastion to build a Docker image, then uploads the image to the GCP container registry. 2. GenomeFlow is able to take in reference files locally if this setting is enabled by a user. 3. GenomeFlow inputs the sample IDs to the queue management service in GKE 4. GenomeFlow deploys the controller and prepares the controller's database for management purposes.



rTEA case report

Many RNA sequences are composed of transposon-derived transcripts but the function of transposon-fusion RNA in humans is still uncertain [15-18]. Therefore, to elucidate the function of transposon-fusion RNA in a pan-cancer setting, we performed an analysis using rTEA. rTEA finds transposon-fusions in RNA sequencing data. We collected 13,251 samples from GTEX, TCGA, ICGC, and CoPM and ran rTEA on each sample [2, 3, 19]. We prepared three different sample groups: (1) SRAtoolkit for GTEx (2) gdctool for TCGA and ICGC (3) and on-premise downloader for CoPM [20, 21]. We uploaded necessary hg38 reference files to the GCP repository [22]. We built a Dockerfile containing the rTEA pipeline, which executes various tools including fastp, hisat2, samtools, scallop, bamtools and bwa based on r-base:3.6.2 [23-28]. We prepared a one-step task for three jobs. Then, GenomeFlow injected the queue consumption module into the Dockerfile and ran all the tasks. After using the recommended resource allocation parameters from GenomeFlow, we were able to reduce memory from 128GB to 64GB, CPU from 16 Core to 8 Core, and storage from 500GB to 250GB. The total running time was around 2 days for all samples including failover tasks.

Epilepsy case report

This pipeline was developed to consider somatic mutations from epilepsy cases unable to be diagnosed by whole exome sequencing [29]. We found that many unknown genes might be related to epilepsy from the EPI4K repository, which we explored further by performing additional sample processing to determine the role of new unknown gene candidates related to epilepsy from the Epi25K epilepsy cohort [30]. We processed 30,789 samples from the Epi25k Repository. Then, we revised the MosaicHunter tool to detect single-nucleotide mosaicism from exon sequencing samples based on Java 1.8 [31]. As an rTEA analysis case, we prepared a one-step task for one job and ran all samples.

Protocols - General Use Case

The two use cases presented in the previous section used a specific pipeline to generate a result for each research objective. However, GenomeFlow supports a user-defined pipeline based on a user-input workflow. Thus, we now show an example of a general use case with a detailed explanation.

Equipment

Currently, GenomeFlow is only compatible with GCP. Although GCP has various resource types for their virtual machines, GenomeFlow only supports a few types of resources due to development limitations: e2-series, n2-series, n1-series [32]. A user is able to select a resource type for the machine; the default type is e2-standard-16 (16 vCPU and 64 GB Memory).

Required software and hardware

GenomeFlow uses Kubernetes to deploy tasks of a job defined by the workflow. The current supported version of Google Kubernetes Engine (GKE) is 1.23.8-gke.1900 which is the default setting of GKE [11]. GenomeFlow first creates the default nodes on the prepared GKE, then creates a service and controller for the queue (RabbitMQ) and controller (Flask), before finally deploying them to the default node of GKE [33, 34]. Afterwards, GenomeFlow creates nodes upon the user's request, and a pipeline in the workflow starts tasks as deployed pods in GKE nodes.

Required data

The user-defined files are as follows: reference file and sample ID list, Dockerfile, and workflow file.

Reference file and sample ID file preparation

If a user's pipeline requires reference files, the user should consider uploading their files to the GCP bucket. Default size limitation of the Docker container is 10GB, which is not enough for large reference files. Users can indicate the GCP bucket location in the workflow file to skip the reference upload step, or users can use GenomeFlow to upload their files to a random GCP bucket location within the created project space. Stored reference files will automatically be transferred to the persistent disk (PD) of each pod (task), and tasks will use the reference files. In addition to the reference files, it is necessary to prepare a sample ID list file and sample file download commands including download tools. The queue controller of GenomeFlow consumes the sample ID file to distribute sample IDs to each task. Users must also provide the sample download location and sample download commands with required tools. The required tools must be included in the user-built Dockerfile.

Dockerfile preparation

GenomeFlow is composed of a GCP architecture creation function that runs a user-defined pipeline from a workflow file. Users must provide a Dockerfile since the workflow file uses a Docker image to run a pipeline. Thus, a user must build a Dockerfile before preparing the workflow file. A Docker image is created in a bastion instance which is deployed as a service

pod in a GKE cluster (Docker version 20.10 of Ubuntu 20.04 LTS is installed in the bastion). A user should build their own Dockerfile with compatible syntax to the specified version of Docker, and provide the Dockerfile location to the workflow file. The Dockerfile will be uploaded to a GCP bucket by GenomeFlow, and GenomeFlow will create a Docker image and transfer this image to a GCP container registry from the bastion instance. Simultaneously, the address of the registry will be stored in the GenomeFlow controller database for the next step.

Workflow file preparation

The workflow file is a pipeline description file that prepares the GKE cluster on GCP and deploys tasks to a cluster node as specified. GenomeFlow supports two types of syntax: (1) Workflow Description language (WDL) and (2) Snakemake style language [35, 36]. In this example, we adopted Snakemake rules. The essential rules for GenomeFlow are as follows:

```
rule           = "rule" (identifier | "") ":" ruleparams
workdir        = "workdir:" stringliteral
configfile     = "configfile" ":" stringliteral
input          = "input" ":" parameter_list
output         = "output" ":" parameter_list
params         = "params" ":" parameter_list (including sampleID)
resources      = "resources" ":" parameter_list
shell          = "shell" ":" stringliteral
script         = "script" ":" stringliteral
metawrapper    = "meta_wrapper" ":" stringliteral
config         = "config" ":" stringliteral
image          = "image" ":" stringliteral
referencefile  = "referencefile" ":" stringliteral
testsamplesize = "testsamplesize" ":" integer
```

Each task of GenomeFlow uses a concept of Directed Acyclic Graph (DAG) to represent its steps [12]. However, GenomeFlow does not support sub-DAGs for parallel steps within a task. A job has multiple tasks, while a workflow file mainly focuses on a single task and every task runs the same pipeline in a job. Tasks execute multiple steps and each step is assigned to a pod in a node of GKE. Each step can have different resource types based on the node type. This serves to optimize the resources per step and reduce costs. If a user does not allocate resources, GenomeFlow uses default resource types and downloads the default number of samples to find optimized resource parameters. Then, the user can check the default parameters and set them to reduce the cost.

Executing the workflow

GenomeFlow is a Python library. After preparing all the required files, users can load GenomeFlow to execute the workflow. A procedure of GenomeFlow includes (1) loading the workflow file, (2) building the GCP architecture, (3) executing a test run to obtain resource optimization parameters, (4) running the pipeline, (5) getting the results and removing the GCP environment. Below, we present a detailed outline of the workflow execution.

After installing GenomeFlow using pip, users load GenomeFlow:

```
>>> import genomeflow as gf
```

Users then load the workflow file along with other essential files, including the sample ID list and Dockerfile:

```
>>> workflow=gf.loadJobFile("exampleWorkflow.snakemake")
```

Users next create a GCP architecture for the pipeline in the workflow:

```
>>> gf.createArchitecture(workflow)
```

Optimized parameters (optParams) are returned from the test run of GenomeFlow. The optParams will replace the previous resource settings of the workflow.

```
>>> optParam=gf.findOptimizedParam(workflow)
```

Then, users run all the samples with optimized resource parameters. Users can check the progress through the GenomeFlow monitoring interface which preserves all generated logs from the GenomeFlow controller.

```
>>> gf.runPipeline(optParam)
```

Obtain results and check costs

Results are stored in a user designated GCP bucket location. Users are able to download each result per sample and check the file metadata using the GCP web interface (Figure 2). Users are also able to download all the data using gstutil of Cloud SDK with the following command:

```
$ gsutil cp -n -r [Bucket Address] [Destination Address]
```

After downloading all the sample results, users can remove GCP buckets and GCP architectures. The GCP architectures include GKE, Cloud SQL and GCP buckets, but do not include the GCP project itself.

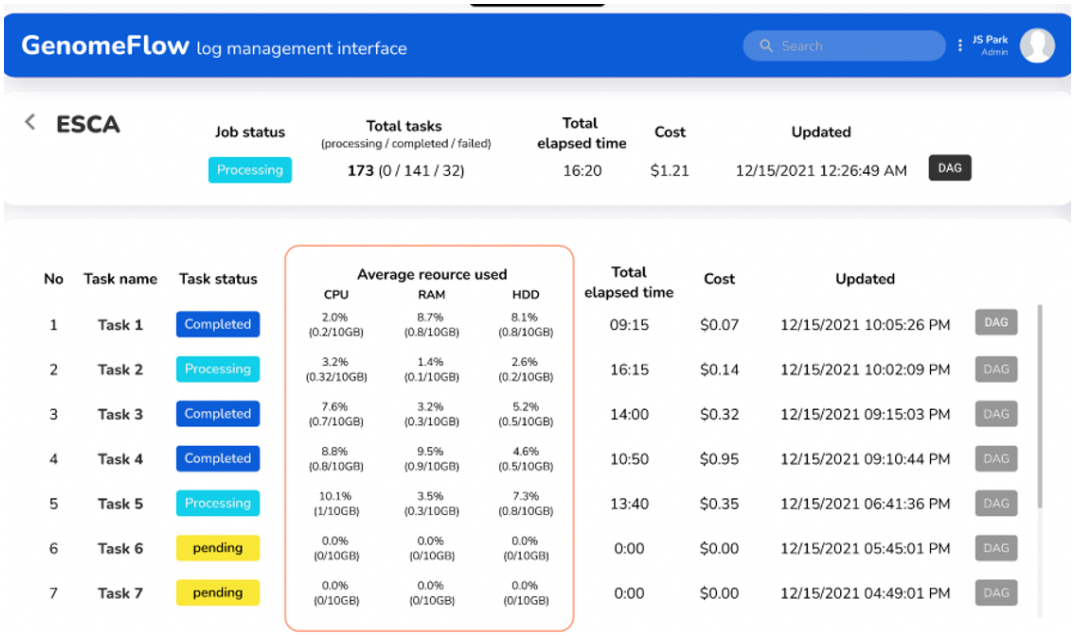
```
>>> gf.removeProject(workflow)
```

If a user runs the remove function of GenomeFlow without any input parameters, it leaves the GCP project untouched. The purpose of preventing removal of the GCP project is to provide a cost check for the user. If users wish to remove the GCP project too, they can use the following command:

```
>>> gf.removeProject(workflow, all=True)
```

Figure 2. Job and task monitoring interface of GenomeFlow. A user can check the job status and total tasks belonging to the job. The interface provides the total cost and the last

update time. Once the user clicks on the job title, the user can see the detailed task status list as shown. Users can find resource usage, current cost, and task status to manage their job in the interface. (Disclaimer: The user interface may be updated upon publication.)



Code Availability and Requirements

All the related code and Jupyter notebooks are available at https://github.com/ealeelab/genomeflow. Users can refer to the project's README.MD file for additional commands and instructions. The detailed requirements are:

- Project name: GenomeFlow
- Project home page: https://github.com/ealeelab/genomeflow
- Operating system: Does not depend on operating system
- Programming language and version: Python 3.6 and 3.7
- License: GNU GPL
- Any restriction to use by non-academics: No

Acknowledgement

We thank Rich Fellmann, Ashrut Vora and Andrew Carey of Google Cambridge office for their help with technological issues on cloud computing platforms. We also thank Danielle Denisko for revising the manuscript. This work was supported by the NHGRI AnVIL Project (AnVIL AC3 Award 2022), the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R1A6A3A14046389), NIH (DP2 AG0724), and Suh Kyungbae Foundation.

References

1. Marx, V., *The big challenges of big data.* Nature, 2013. **498**(7453): p. 255-260.
2. Tomczak, K., P. Czerwińska, and M. Wiznerowicz, *Review The Cancer Genome Atlas (TCGA): an immeasurable source of knowledge.* Contemporary Oncology/Współczesna Onkologia, 2015. **2015**(1): p. 68-77.

3. Zhang, J., et al., *International Cancer Genome Consortium Data Portal—a one-stop shop for cancer genomics data.* Database, 2011. **2011**.
4. Kowalski, M.H., et al., *Use of >100,000 NHLBI Trans-Omics for Precision Medicine (TOPMed) Consortium whole genome sequences improves imputation quality and detection of rare variant associations in admixed African and Hispanic/Latino populations.* PLoS genetics, 2019. **15**(12): p. e1008500.
5. Biobank, U.K., *About uk biobank*. 2014.
6. Van Hout, C.V., et al., *Exome sequencing and characterization of 49,960 individuals in the UK Biobank.* Nature, 2020. **586**(7831): p. 749-756.
7. Lee, S., et al., *Tibanna: software for scalable execution of portable pipelines on the cloud.* Bioinformatics, 2019. **35**(21): p. 4424-4426.
8. *Terra: a cloud-native platform for biomedical researchers to access data, run analysis tools, and collaborate*. Available from: https://app.terra.bio/.
9. *Google Cloud Platform*. Available from: https://cloud.google.com/.
10. Junseok Park, E.M., Changhoon Oh, Donghoon Shin, Eunjung Alice Lee, *GenomeFlow.* Submission is In preparation to bioRxiv.org, 2022.
11. Burns, B., et al., *Kubernetes: up and running*. 2022: " O'Reilly Media, Inc.".
12. Singh, P., *Airflow*, in *Learn PySpark*. 2019, Springer. p. 67-84.
13. DuBois, P., *MySQL*. 2008: Pearson Education.
14. Boettiger, C., *An introduction to Docker for reproducible research.* ACM SIGOPS Operating Systems Review, 2015. **49**(1): p. 71-79.
15. Chu, C., et al., *Comprehensive identification of transposable element insertions using multiple sequencing technologies.* Nature communications, 2021. **12**(1): p. 1-12.
16. Tubio, J.M.C., et al., *Extensive transduction of nonrepetitive DNA mediated by L1 retrotransposition in cancer genomes.* Science, 2014. **345**(6196): p. 1251343.
17. Jung, H., J.K. Choi, and E.A. Lee, *Immune signatures correlate with L1 retrotransposition in gastrointestinal cancers.* Genome research, 2018. **28**(8): p. 1136-1146.
18. Lee, E., et al., *Landscape of somatic retrotransposition in human cancers.* Science, 2012. **337**(6097): p. 967-971.
19. Consortium, G.T., et al., *The Genotype-Tissue Expression (GTEx) pilot analysis: multitissue gene regulation in humans.* Science, 2015. **348**(6235): p. 648-660.
20. Sherry, S., et al. *NCBI SRA toolkit technology for next generation sequence data*.
21. Torcivia-Rodriguez, J., et al., *A primer for access to repositories of cancer-related genomic big data.* Cancer Bioinformatics, 2019: p. 1-37.
22. Rosenbloom, K.R., et al., *The UCSC genome browser database: 2015 update.* Nucleic acids research, 2015. **43**(D1): p. D670-D681.
23. Chen, S., et al., *fastp: an ultra-fast all-in-one FASTQ preprocessor.* Bioinformatics, 2018. **34**(17): p. i884-i890.
24. Li, H., et al., *The sequence alignment/map format and SAMtools.* Bioinformatics, 2009. **25**(16): p. 2078-2079.
25. Shao, M. and C. Kingsford, *Accurate assembly of transcripts through phase-preserving graph decomposition.* Nature biotechnology, 2017. **35**(12): p. 1167-1169.
26. Barnett, D.W., et al., *BamTools: a C++ API and toolkit for analyzing and managing BAM files.* Bioinformatics, 2011. **27**(12): p. 1691-1692.
27. Li, H., *Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM.* arXiv preprint arXiv:1303.3997, 2013.
28. Tippmann, S., *Programming tools: Adventures with R.* Nature, 2015. **517**(7532): p. 109-110.
29. Koh, H.Y. and J.H. Lee, *Brain somatic mutations in epileptic disorders.* Molecules and cells, 2018. **41**(10): p. 881.

30. Epi, K.C., *Epi4K: gene discovery in 4,000 genomes*. 2012, Wiley Online Library.
31. Huang, A.Y., et al., *MosaicHunter: accurate detection of postzygotic single-nucleotide mosaicism through next-generation sequencing of unpaired, trio, and paired samples.* Nucleic acids research, 2017. **45**(10): p. e76-e76.
32. *Machine families of Google Cloud Platform*. Available from: https://cloud.google.com/compute/docs/machine-types.
33. Ionescu, V.M. *The analysis of the performance of RabbitMQ and ActiveMQ*. IEEE, 2015.
34. Relan, K., *Building REST APIs with Flask.* Building REST APIs with Flask, 2019.
35. Nandra, C.I. and D. Gorgan. *Workflow description language for defining big earth data processing tasks*. IEEE, 2015.
36. Köster, J. and S. Rahmann, *Snakemake—a scalable bioinformatics workflow engine.* Bioinformatics, 2012. **28**(19): p. 2520-2522.