Contents

1 Select Genomic Data to work with 3
  1.1 Discover Published Data ........................................... 3

2 Process Data on Google Cloud 21
  2.1 Run workflows and common tasks in parallel ...................... 21
  2.2 Create a Grid Engine cluster on Compute Engine .......... 22
  2.3 Create a Grid Engine cluster with Preemptible VM workers ....... 29
  2.4 Run SAMtools to index BAM files in Cloud Storage .......... 32
  2.5 Compress/Decompress files in Cloud Storage ................. 39
  2.6 Run Galaxy on Compute Engine ................................ 46
  2.7 Run NCBI BLAST on Compute Engine ............................ 46
  2.8 Run Bioconductor on Compute Engine .......................... 47
  2.9 Run iPython Notebooks on Compute Engine ................. 48

3 Access Genomic Data using... 51
  3.1 Integrative Genomics Viewer (IGV) .............................. 51
  3.2 Run Picard and GATK tools on Cloud-Resident Genomic Data .... 53
  3.3 Browse Reads with Bioconductor ............................... 58
  3.4 Beacon ............................................................. 58
  3.5 GABrowse .......................................................... 59
  3.6 The R client ....................................................... 60
  3.7 Python ............................................................ 60
  3.8 Java ................................................................. 61
  3.9 Go ................................................................. 61

4 Analyze Data in Google Genomics 63
  4.1 Analyze Reads ...................................................... 63
  4.2 Analyze Variants .................................................. 71
  4.3 Annotate Variants ................................................ 104
  4.4 Perform Quality Control Checks ................................. 114
  4.5 Compute and Analyze Linkage Disequilibrium .............. 117
  4.6 Advanced BigQuery Topics .................................. 123

5 Learn More 127
  5.1 Migrating from Genomics API v1beta2 to v1 .................. 127
  5.2 Build your own Google Genomics API Client .................. 130
  5.3 BioC 2015: Where Software and Biology Connect ........... 135
5.4 Find Samples on GitHub ......................................................... 139
5.5 Subscribe to the mailing list .................................................... 140
Welcome to the Google Genomics Cookbook on Read the Docs.

The official Google Genomics documentation can be found at cloud.google.com/genomics. There you will find information such as What is Google Genomics, Google Genomics fundamentals, and details of the Google Genomics API.

Here on Read the Docs, you will find documentation and tutorials for common tasks including moving, transforming, and analyzing genomic data.
Select Genomic Data to work with

Discover Published Data

Genomic Data

1,000 Genomes

This dataset comprises roughly 2,500 genomes from 25 populations around the world. See the 1,000 Genomes project website and publications for full details:

Pilot publication

An integrated map of genetic variation from 1,092 human genomes
The 1000 Genomes Project Consortium
Published: November 1, 2012
DOI: 10.1038/nature11632
Phase 1 publication

A map of human genome variation from population scale sequencing
The 1000 Genomes Project Consortium
Published: October 28, 2010
DOI: 10.1038/nature09534

Phase 3 publications

A global reference for human genetic variation
The 1000 Genomes Project Consortium
Published: September 30, 2015
DOI: 10.1038/nature15393

An integrated map of structural variation in 2,504 human genomes
The 1000 Genomes Project Consortium
Published: September 30, 2015
DOI: 10.1038/nature15394

Google Cloud Platform data locations

• Google Cloud Storage folders
  – These files were loaded into Google Genomics datasets:
    * gs://genomics-public-data/1000-genomes
    * gs://genomics-public-data/1000-genomes-phase-3

• Google Genomics Dataset IDs
  – Dataset Id 10473108253681171589
    * ReadGroupSet IDs for the Phase 3 reads
    * Phase 1 variants
      · Variant Set Id: 10473108253681171589
      · Reference Bounds
    * Phase 3 variants - 20150220 Release
      · Variant Set Id: 11027761582969783635
      · Reference Bounds
  – Dataset Id 4252737135923902652
Phase 3 variants - initial release

- Variant Set Id: 4252737135923902652
- Reference Bounds

- **Google BigQuery Dataset IDs**
  - genomics-public-data:1000_genomes phase 1 variants and sample information
  - genomics-public-data:1000_genomes_phase_3 phase 3 variants

**Beacon and GA4GH**

You can find a Global Alliance for Genomics and Health Beacon at http://webdev.dnastack.com/p/beacon/thousandgenomes?chromosome=1&coordinate=10177&allele=AC

You can find an instance of the GA4GH reference server hosting this data at http://1kgenomes.ga4gh.org/.

**Provenance**

The source files for this dataset include:

- The mapped full-genome phase 3 BAM files listed at the 1000 Genomes FTP site.
- All of the phase 1 VCF files listed at the 1000 Genomes FTP site.
- All of the phase 3 VCF files listed at the 1000 Genomes FTP site.
- These files were copied to Google Cloud Storage, uploaded to Google Genomics, and the variants were exported to Google BigQuery.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

**Illumina Platinum Genomes**

This dataset comprises the 6 member CEPH pedigree 1463. See http://www.illumina.com/platinumgenomes/ for full details.

**Google Cloud Platform data locations**

- Google Cloud Storage folder gs://genomics-public-data/platinum-genomes
- Google Genomics Dataset ID 3049512673186936334
  - ReadGroupSet IDs
  - Variant Reference Bounds
- Google BigQuery Dataset ID genomics-public-data:platinum_genomes
Beacon

You can find a Global Alliance for Genomics and Health Beacon at http://webdev.dnastack.com/p/beacon/platinum?chromosome=1&coordinate=10177&allele=AC

Provenance

- The source files for this data include:
  - All of the BAM files listed at the EBI FTP site.
  - All of the VCF files were listed at the Illumina FTP site prior to the IlluminaPlatinumGenomes_v6.0 release but they have since been taken down.
- These files were copied to Google Cloud Storage, uploaded to Google Genomics, and the variants were exported to Google BigQuery.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Platinum Genomes DeepVariant

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

This dataset comprises the 6 member CEPH pedigree 1463 called using the alpha version of the Verily DeepVariant toolchain aligned to Verily’s GRCh38 reference genome. See the DeepVariant preprint for full details:

Creating a universal SNP and small indel variant caller with deep neural networks
Ryan Poplin, Dan Newburger, Jojo Dijamco, Nam Nguyen, Dion Loy, Sam Gross, Cory Y. McLean, Mark A. DePristo
DOI: https://doi.org/10.1101/092890

Google Cloud Platform data locations

- Google Cloud Storage folder gs://genomics-public-data/platinum-genomes-deepvariant
- Google Genomics Dataset ID 14839180708999654392
  - Variant Reference Bounds
- Google BigQuery Dataset ID genomics-public-data:platinum_genomes_deepvariant

Provenance

- The FASTQ files in gs://genomics-public-data/platinum-genomes/fastq/ were run through the DeepVariant toolchain to produce the corresponding *.deepvariant.g.vcf and *.deepvariant.vcf files in gs://genomics-public-data/platinum-genomes-deepvariant/vcf/.
These files were then imported to Google Genomics and the variants were exported to Google BigQuery as table `genomics-public-data:platinum_genomes_deepvariant.single_sample_genome_calls`.

The data was then merged to produce variants-only `multisample-platinum-genomes-deepvariant.vcf` and table `genomics-public-data:platinum_genomes_deepvariant.multisample_variants`.

- The merging logic:
  - groups together only single- and multi-nucleotide polymorphisms with the same reference representation and alternate allele length that originate at the same chromosome and reference position
  - merges all insertions at the same reference position, and
  - splits complex variants into multiple records.
- Individual variants with GQ < 20 are hard-masked to no-calls, with the genotype likelihoods retained.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

**PrecisionFDA Truth Challenge**

The properly rendered version of this document can be found at Read The Docs.

If you are reading this on github, you should instead click [here](https://cloud.google.com/genomics/support).

This dataset includes both:

- the input for the PrecisionFDA Truth Challenge comprised of whole-genome sequences for HG001 (NA12878) and HG002 (NA24385)
- the output from the alpha version of the Verily DeepVariant toolchain aligned to Verily’s GRCh38 reference genome. See the DeepVariant preprint for full details:

  Creating a universal SNP and small indel variant caller with deep neural networks
  Ryan Poplin, Dan Newburger, Jojo Dijamco, Nam Nguyen, Dion Loy, Sam Gross, Cory Y. McLean, Mark A. DePristo
  DOI: [https://doi.org/10.1101/092890](https://doi.org/10.1101/092890)

**Google Cloud Platform data locations**

- Google Cloud Storage folder gs://genomics-public-data/precision-fda

**Provenance**

- The FASTQ files in gs://genomics-public-data/precision-fda/input were run through the Verily DeepVariant alpha toolchain to produce the corresponding files in gs://genomics-public-data/precision-fda/output/deepvariant-alpha.
Reference Genomes

Reference Genomes such as GRCh37, GRCh37lite, GRCh38, hg19, hs37d5, and b37 are available on Google Cloud Platform.

Google Cloud Platform data locations

- Google Cloud Storage folder gs://genomics-public-data/references
- Google Genomics reference sets

Provenance

GRCh37

Genome Reference Consortium Human Build 37 includes data from 35 gzipped fasta files:

- assembled chromosomes
- unlocalized scaffolds
- unplaced scaffolds

More information on this source data can be found in this NCBI article and in the FTP README.

GRCh37lite

GRCh37lite is a subset of the full GRCh37 reference set plus the human mitochondrial genome reference sequence in one file: GRCH37-lite.fa.gz

More information on this source data can be found in the FTP README.

GRCh38

Genome Reference Consortium Human Build 38 includes data from 39 gzipped fasta files:

- assembled chromosomes
- unlocalized scaffolds
- unplaced scaffolds
- non-nuclear references

More information on this source data can be found in this NCBI article and in the FTP README.
Verily’s GRCh38

Verily’s GRCh38 reference genome is fully compatible with any b38 genome in the autosome.

Verily’s GRCh38:

• excludes all patch sequences
• omits alternate haplotype chromosomes
• includes decoy sequences
• masks out duplicate copies of centromeric regions

The base assembly is GRCh38_no_alt_plus_hs38d1. This assembly version was created specifically for analysis, with its rationale and exact genome modifications thoroughly documented in its README file.

Verily applied the following modifications to the base assembly:

• Reference segment names are prefixed with “chr”.

  Many of the additional data files we use are provided by GENCODE, which uses “chr” naming convention.

• All 74 extended IUPAC codes are converted to the first matching alphabetical base pair as recommended in the VCF 4.3 specification.

• This release of the genome reference is named GRCh38_Verily_v1

hg19

Similar to GRCh37, this is the February 2009 assembly of the human genome with a different mitochondrial sequence and additional alternate haplotype assemblies. Includes data from all 93 gzipped fasta files from the UCSC FTP site.

More information on this source data can be found in the FTP README.

hs37d5

Includes data from GRCh37, the rCRS mitochondrial sequence, Human herpesvirus 4 type 1 and the concatenated decoy sequences in one file: hs37d5.fa.gz.

More information on this source data can be found in the FTP README.

b37

The reference genome included by some versions of the GATK software which includes data from GRCh37, the rCRS mitochondrial sequence, and the Human herpesvirus 4 type 1 in one file: Homo_sapiens_assembly19.fasta.

More information on this source data can be found in the GATK FAQs.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
MSSNG Database for Autism Researchers

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

This dataset comprises a growing collection of both Illumina and Complete Genomics genomes of families affected by autism. Apply for data access at MSSNG Database for Autism Researchers. See the journal article for full details:

Whole-genome sequencing of quartet families with autism spectrum disorder
Published January 26, 2015
DOI: 10.1038/nm.3792

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

TCGA Cancer Genomics Data in the Cloud

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

Use the power of BigQuery to analyze the wealth of data created by The Cancer Genome Atlas (TCGA) project!

The Institute for Systems Biology (ISB) has created and made public a dataset based on the open-access TCGA data including somatic mutation calls, clinical data, mRNA and miRNA expression, DNA methylation and protein expression from 33 different tumor types. It’s part of their Cancer Genomics Cloud, funded by the National Cancer Institute. They’ve also created public github repositories so you can try out sample queries and analyses in R or Google Cloud Datalab,

- documentation
- examples in R
- examples in Python

Google Cloud Platform data locations

- Google BigQuery Dataset ID isb-cgc:tcga_201607_beta

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
**Supercentenarian Genomes**

This dataset comprises Complete Genomics genomes for 17 supercentenarians (110 years or older). **Apply for data access at [http://goo.gl/MGcYJ5](http://goo.gl/MGcYJ5).** See the journal article for full details:

*Whole-Genome Sequencing of the World’s Oldest People*

Hinco J. Gierman, Kristen Fortney, Jared C. Roach, Natalie S. Coles, Hong Li, Gustavo Glusman, Glenn J. Markov, Justin D. Smith, Leroy Hood, L. Stephen Coles, Stuart K. Kim

Published: November 12, 2014
DOI: 10.1371/journal.pone.0112430

**Google Cloud Platform data locations**

- Google Genomics dataset 18254571932956699773 once access has been granted.

**Provenance**

- The data are also available from [http://supercentenarians.stanford.edu/](http://supercentenarians.stanford.edu/).
- The CGI masterVar files were uploaded to Google Cloud Storage and then imported to Google Genomics.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or [file an issue](https://cloud.google.com/genomics/support).

**Personal Genome Project Data**

This dataset comprises roughly 180 Complete Genomics genomes. See the Personal Genome Project and the publication for full details:

*A public resource facilitating clinical use of genomes*


Published: July 24, 2012
DOI: 10.1073/pnas.1201904109
Google Genomics Documentation, Release v1

Google Cloud Platform data locations

- Google Cloud Storage folder gs://pgp-harvard-data-public
- Google Genomics Dataset ID 9170389916365079788
- Google BigQuery Dataset IDs
  - google.com:biggene:pgp_20150205.genome_calls

Provenance

Google Genomics variant set for dataset pgp_20150205: 9170389916365079788 contains:

- the Complete Genomics datasets from gs://pgp-harvard-data-public/**/masterVar*bz2

Appendix

Google is hosting a copy of the PGP Harvard data in Google Cloud Storage. All of the data is in this bucket: gs://pgp-harvard-data-public

If you wish to browse the data you will need to install gsutil.

Once installed, you can run the `ls` command on the pgp bucket:

```
$ gsutil ls gs://pgp-harvard-data-public
gs://pgp-harvard-data-public/cgi_disk_20130601_00C68/
gs://pgp-harvard-data-public/hu011C57/
....lots more....
```

The sub folders are PGP IDs, so if we `ls` a specific one:

```
$ gsutil ls gs://pgp-harvard-data-public/hu011C57/
```

And then keep diving down through the structure, you can end up here:

```
  --> GS01669-DNA_B05/ASM/
  →ASM/dbSNPAnnotated-GS000015172-ASM.tsv.bz2
  →ASM/gene-GS000015172-ASM.tsv.bz2
  ... and more ...
```

Your genome data is located at: gs://pgp-harvard-data-public/{YOUR_PGP_ID}

If you do not see the data you are looking for, you should contact PGP directly through your web profile.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
ICGC-TCGA DREAM Mutation Calling Challenge synthetic genomes

This dataset comprises the three public synthetic tumor/normal pairs created for the ICGC-TCGA DREAM Mutation Calling challenge. See the journal article for full details regarding how the synthetic data for challenge in silico #1 was created:

Combining tumor genome simulation with crowdsourcing to benchmark somatic single-nucleotide-variant detection
Adam D Ewing, Kathleen E Houlahan, Yin Hu, Kyle Ellrott, Cristian Caloian,
Takafumi N Yamaguchi, J Christopher Bare, Christine P’ng, Daryl Waggott,
Veronica Y Sabelnykova, ICGC-TCGA DREAM Somatic Mutation Calling Challenge participants,
Michael R Kellen, Thea C Norman, David Haussler, Stephen H Friend, Gustavo Stolovitzky,
Adam A Margolin, Joshua M Stuart & Paul C Boutros
Published: May 18, 2015
DOI: 10.1038/nmeth.3407

Google Cloud Platform data locations

- Google Cloud Storage folder gs://public-dream-data/
- Google Genomics dataset 337315832689.

Provenance

- The authoritative data location is NCBI Sequence Read Archive: SRP042948.
- The BAMs were uploaded to Google Cloud Storage and the reads were then imported to Google Genomics.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Simons Genome Diversity Project

This dataset comprises 279 publicly available genomes from 127 diverse populations for the Simons Genome Diversity Project. See the journal articles for full details:

Pilot Publication

The complete genome sequence of a Neanderthal from the Altai Mountains
Google Genomics Documentation, Release v1

Full Dataset Publication

The Simons Genome Diversity Project: 300 genomes from 142 diverse populations
Swapan Mallick, Heng Li, Mark Lipson, Iain Mathieson, Melissa Gymrek, Fernando Racimo, Mengyao Zhao, Niru Chennagiri, Susanne Nordenfelt, Arti Tandon, Pontus Skoglund, Iosif Lazaridis, Sriram Sankararaman, Qiaomei Fu, Nadin Rohland, Gabriel Renaud, Yaniv Erlich, Thomas Willems, Carla Gallo, Jeffrey P. Spence, Yun S. Song, Giovanni Poletti, Francois Balloux, George van Driem, Peter de Knijff et al.
Published 21 September 2016
DOI:10.1038/nature18964

Google Cloud Platform data locations

- Google Cloud Storage folder gs://genomics-public-data/simons-genome-diversity-project
- Google Genomics datasets
  - 279 genomes from the full dataset project 9897701284241799339.
    * Variant Reference Bounds
  - 25 genomes from the pilot project 461916304629.
    * ReadGroupSet IDs
- Google BigQuery Dataset genomics-public-data:simons_genome_diversity_project

Provenance

For the full dataset of 279 genomes:
- The public VCFs described in the README were downloaded from https://sharehost.hms.harvard.edu/genetics/reich_lab/sgdp/vcf_variants/ and extracted to gs://genomics-public-data/simons-genome-diversity-project.
- These files were then imported to Google Genomics and the variants were exported to Google BigQuery as table genomics-public-data:simons_genome_diversity_project.single_sample_genome_calls.
- The sample metadata was loaded to table genomics-public-data:simons_genome_diversity_project.sample_metadata using the following commands:

```
# Strip blank lines from end of file and white space from end of lines.
sed 'a;/^\[t\r\n]*$/{$d;N;ba}'; 10_24_2014_SGDP_metainformation_update.txt \ | sed 's/$//g' > 10_24_2014_SGDP_metainformation_update.tsv
```
pq load --autodetect \
  simons_genome_diversity_project.sample_metadata 10_24_2014_SGDP_metainformation_
  →update.tsv

• The sample metadata does not use the same sample identifiers as in the VCFs and is also missing one row, so
  sample attributes were
  – retrieved from http://www.ebi.ac.uk/ena/data/view/PRJEB9586
  – and reshaped into table genomics-public-data:simons_genome_diversity_project.sample_attributes
  – using script wrangle-simons-sample-attributes.R.
  – This script also re-maps three samples whose ids in the source VCFs did not match the corresponding
    Illumina ID attribute on EBI.

For the pilot dataset of 25 genomes, the BAMs were imported into Google Genomics.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub”
link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

1000 Cannabis Genomes Project

A genomic open dataset of approximately 850 strains of cannabis via the Open Cannabis Project has been made
available on Google Cloud Platform. See the blog post for more context and provenance details.

Google Cloud Platform data locations

See the 1000 Cannabis Genomes Project for full details.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub”
link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Annotations

The properly rendered version of this document can be found at Read The Docs.
If you are reading this on github, you should instead click here.
Tute Genomics Annotation

Tute Genomics has made available to the community annotations for all hg19 SNPs as a BigQuery table.

- For more details about the annotation databases included, see Tute’s blog post.
- For sample queries on public data, see https://github.com/googlegenomics/bigquery-examples/tree/master/platinumGenomes

Google Cloud Platform data locations

- Google Cloud Storage folder gs://tute_db
- Google BigQuery Dataset ID silver-wall-555:TuteTable.hg19

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Linkage Disequilibrium Datasets

Linkage disequilibrium was run separately for each super population and sub population within 1,000 Genomes phase 3 variants using the method defined in Box 1 of:

Linkage disequilibrium - understanding the evolutionary past and mapping the medical future
Slatkin, Montgomery
Nature Reviews Genetics, Volume 9, Issue 6, 477 - 485
DOI: http://dx.doi.org/10.1038/nrg2361

LD was computed for all pairs of variants within a window of 1,000,000 bp (1 megabase) and all pairs with absolute allelic correation of 0.4 are retained. See Compute Linkage Disequilibrium on a Variant Set for more detail.

The output files were split by chromosome with output columns indicating the identity of each pair of values and the resulting LD value. The output files have also been loaded into BigQuery with the same columns. Examples of using BigQuery to analyze LD are available as Datalab notebooks.

Google Cloud Platform data locations

- Google Cloud Storage folder gs://genomics-public-data/linkage-disequilibrium
- Google BigQuery Dataset ID genomics-public-data:linkage_disequilibrium_1000G_phase_3
ClinVar Annotations

Annotations from ClinVar were loaded into Google Genomics for use in sample annotation pipelines. This data reflects the state of ClinVar at a particular point in time.

Google Cloud Platform data locations

- Google Cloud Storage folder gs://genomics-public-data/clinvar/
- Google Genomics annotation sets

Provenance

Each of the annotation sets listed below was imported into the API from the source files. The source files are also mirrored in Google Cloud Storage.

ClinVar (downloaded 2/5/2015 10:18AM PST):

Caveats

A number of ClinVar entries were omitted during ingestion due to data incompatibility with the Google Genomics API.

- 14737 were aligned to NCBI36, which the Google Genomics API does not currently support.
- 5952 did not specify a reference assembly.
- 1324 were labeled as insertions but did not specify the inserted bases.
- 220 were labeled as SNPs, but did not specify an alternate base.
- 148 were larger than 100MBp.
UCSC Annotations

UCSC Sequence and Annotation Data were loaded into Google Genomics for use in sample annotation pipelines. This data reflects the state of UCSC Sequence and Annotation Data at a particular point in time.

Google Cloud Platform data locations

- Google Cloud Storage folder gs://genomics-public-data/ucsc/
- Google Genomics annotation sets

Provenance

Each of the annotation sets listed below was imported into the API from the source files. The source files are also mirrored in Google Cloud Storage.

UCSC GRCh38 (downloaded 12/29/2014 14:00 PST):
- http://hgdownload.cse.ucsc.edu/goldenPath/hg38/database/refFlat.txt.gz
- http://hgdownload.cse.ucsc.edu/goldenPath/hg38/database/refGene.txt.gz
- http://hgdownload.cse.ucsc.edu/goldenPath/hg38/database/knownGene.txt.gz

UCSC hg19 (downloaded 3/5/2015 17:00 PST):
- http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/refFlat.txt.gz
- http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/refGene.txt.gz
- http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/knownGene.txt.gz

COSMIC Annotations

The Institute for Systems Biology Cancer Genomics Cloud (ISB-CGC) has made COSMIC available as BigQuery tables to provide a new way to explore and understand the mutations driving cancer.


Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
Need more help? Please see https://cloud.google.com/genomics/support.

Please let us know if you have a dataset that you wish to share and have listed here for discovery.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
Run workflows and common tasks in parallel

Do you have a task that you need to run independently over dozens, hundreds, or thousands of files in Google Cloud Storage? The Google Genomics Pipelines API provides an easy way to launch and monitor tasks running in the cloud.

Overview

A “pipeline” in its simplest form is a task consisting of:

- Path(s) of input files to read from Cloud Storage
- Path(s) of output files/directories to write to Cloud Storage
- A Docker image to run
- A command to run in the Docker image
- Cloud resources to use (number of CPUs, amount of memory, disk size and type)

The Pipelines API will:

1. Create a Compute Engine virtual machine
2. Download the Docker image
3. Download the input files
4. Run a new Docker container with the specified image and command
5. Upload the output files
6. Destroy the Compute Engine virtual machine

Log files are uploaded periodically to Cloud Storage.

Alternatives

For many cases, the Pipelines API has an advantage over fixed clusters in that Compute Engine resources (virtual machines and disks) are allocated only for the lifetime of the running pipeline, and are then destroyed.

However many existing scripts assume a fixed cluster (such as a shared disk). If you want to create a fixed cluster, see *Create a Grid Engine cluster on Compute Engine*.

Getting started examples

We have a github repository with several pipelines-api-examples to help you get started.

See the README at the top of the repository for prerequisites. Existing examples include:

- Compress or decompress files
- Run FastQC over a list of BAM or FASTQ files
- Use samtools to create a BAM index file
- Use a custom script in Cloud Storage to update a VCF header
- Use Bioconductor to count overlaps in a BAM file

Beyond Files

Note that the Pipelines API is not only for working with files. If you have tools that access data in Google Genomics, Google BigQuery, or any other Google Cloud API, they can be run using the Pipelines API.

When running a pipeline, simply include the appropriate OAuth 2.0 Scope for the Compute Engine ServiceAccount.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Create a Grid Engine cluster on Compute Engine

This document provides getting started instructions for using Elasticluster to create clusters of Google Compute Engine instances running the Grid Engine job management software.

Elasticluster “aims to provide a user-friendly command line tool to create, manage and setup computational clusters hosted on cloud infrastructures” and can be used to setup software such as Grid Engine, SLURM, or Apache Hadoop.
What you will do

By following the instructions in this document, you will learn to:

1. Install Elasticluster software on your workstation/laptop
2. Configure Elasticluster to access your Google Cloud Project
3. Configure a cluster of Compute Engine virtual machines running Grid Engine
4. Start a cluster of Compute Engine virtual machine running Grid Engine
5. List nodes of your cluster
6. Copy files to the master instance of your cluster
7. SSH to the instances of your cluster
8. Destroy your cluster
9. Update your Elasticluster installation

Install Elasticluster on your workstation/laptop

The following instructions have been tested on Linux and MacOS.

When installing on MacOS, some dependent modules must be compiled, notably pycrypto. If the command line developer tools are not installed, run:

```
xcode-select --install
```
and follow the installation instructions. See xcode-select for more information.

It is highly recommended that you install elasticluster in a python virtualenv. This will allow you to contain your Elasticluster installation and dependent libraries in one place.

The instructions here explicitly use a Python virtualenv and have only been tested in this environment.

0. **If you have not installed virtualenv, then do so with:**

```
[sudo] pip install virtualenv
```

If you do not have write permission to the global Python site-packages directory, you can install virtualenv with the Python user scheme:

```
pip install --user virtualenv
```

If you do not have pip installed, you can find instructions here.

1. **Change directory to where you want to install Elasticluster**

You don’t need to explicitly create an elasticluster directory (that will happen next).

2. **Create a virtualenv called elasticluster:**

```
virtualenv elasticluster
```

This creates a directory named elasticluster and populates it with the necessary Python library files and shell scripts to contain the Elasticluster installation. No Elasticluster software has yet been installed.

3. **Activate the elasticluster virtualenv for the current shell session:**

```
source elasticluster/bin/activate
```

This script will set environment variables necessary for the virtualenv:

2.2. Create a Grid Engine cluster on Compute Engine
• **VIRTUAL_ENV**: path to the `elasticluster` virtualenv directory

• **PATH**: adds `${VIRTUAL_ENV}/bin` to the head of the PATH

• **PYTHONHOME**: unset if currently set

The script also saves away changed environment variables and installs a `deactivate` function into the bash environment.

4. **Install elasticluster** (select one):

   The [googlegenomics](https://github.com/googlegenomics) github organization maintains a fork of elasticluster. The purpose of this fork is to provide bug fixes and enhancements relevant to Google Cloud and customer use-cases. All such changes are submitted as pull requests to the mainline branch, and development is coordinated with S3IT.

   The mainline fork is currently up-to-date with pull requests from the [googlegenomics](https://github.com/googlegenomics) fork. We suggest you use the mainline fork unless you are interested in submitting a pull request for new features and bugs, including any items from the Issues list.

   (a) From github (mainline)

   ```bash
   cd elasticluster
git clone git://github.com/gc3-uzh-ch/elasticluster.git src
cd src
pip install -e .
   ```

   (a) From github (googlegenomics fork)

   ```bash
   cd elasticluster
git clone https://github.com/googlegenomics/elasticluster.git src
cd src
pip install -e .
   ```

**Create your cluster definition file**

Elasticluster cluster definitions are driven from a configuration file. By default this file is:

```
~/.elasticluster/config
```

Details of the config file can be found at:


Elasticluster provides a command to automatically create the config file for you, however using this command will create a template configuration file which you cannot immediately use as it includes a list of clusters that are not correctly configured.

You can either:

1. Install the default template using `list-templates` and then fix it up, or

2. Install a minimal template provided below

In either case, you will need to configure the `~/.elasticluster/config` file for accessing your Google Cloud project.
Install the default template

If you install the default template using the command:

```bash
elasticcluster list-templates
```

It will copy a default file to `~/.elasticluster/config` and will emit a number of WARNINGS and ERRORS to the console. To use this configuration file, you must then comment out or remove all of the “cluster” examples. Look for the section:

```bash
# Cluster Section
```

and then comment out or remove everything up to the:

```bash
# Cluster node section
```

You can then copy each element (except `setup/ansible-gridengine`) of the following minimal template into the config file.

Install a minimal template

Copy the following into `~/.elasticluster/config` and update the fields marked with `***`.

Instructions for getting your client_id and client_secret can be found below.

Instructions for ensuring your SSH keypair exists can be found below.

```plaintext
# Gridengine software to be configured by Ansible
[setup/ansible-gridengine]
provider=ansible
frontend_groups=gridengine_master
compute_groups=gridengine_clients

# Create a cloud provider (call it "google-cloud")
[cloud/google-cloud]
provider=google
gce_project_id=***REPLACE WITH YOUR PROJECT ID***
gce_client_id=***REPLACE WITH YOUR CLIENT ID***
gce_client_secret=***REPLACE WITH YOUR SECRET KEY***

# Create a login (call it "google-login")
[login/google-login]
image_user=***REPLACE WITH YOUR GOOGLE USERID (just the userid, not email)***
image_user_sudo=root
image_sudo=True
user_key_name=elasticluster
user_key_private=~/.ssh/google_compute_engine
user_key_public=~/.ssh/google_compute_engine.pub

# Bring all of the elements together to define a cluster called "gridengine"
[cluster/gridengine]
cloud=google-cloud
login=google-login
setup_provider=ansible-gridengine
security_group=default
image_id=***REPLACE WITH OUTPUT FROM: gcloud compute images list | grep debian | cut -f 1 -d " "***
flavor=n1-standard-1
```

2.2. Create a Grid Engine cluster on Compute Engine
Setting the boot disk size

For cluster tasks you may want to create use SSD persistent disk or a boot disk larger than the default 10 GB. Elasti-cluster allows for specifying both the boot disk type and size for instances of your cluster:

**boot_disk_type** Define the type of boot disk to use. Supported values are *pd-standard* and *pd-ssd*. Default value is *pd-standard*.

**boot_disk_size** Define the size of boot disk to use. Values are specified in gigabytes. Default value is 10.

The disk type and size can be set for a cluster or for a group of nodes.

For example to set up the above Grid Engine cluster such that nodes have a 100 GB SSD persistent disk, add the following:

```
[cluster/gridengine]
  ...
  boot_disk_type=pd-ssd
  boot_disk_size=100
```

or to configure all of the compute worker nodes to have a 2 TB Standard (HDD) persistent disk:

```
[cluster/gridengine/compute]
  boot_disk_type=pd-standard
  boot_disk_size=2000
```

Generating your SSH keypair

If you have already connected to a Google Compute Engine instance using `gcloud compute ssh`, then you will have a keypair generated in:

- `~/.ssh/google_compute_engine`
- `~/.ssh/google_compute_engine.pub`

If you do not have a keypair, then the *user_key_private* and *user_key_public* file paths in the [login/google-login] section above will not be valid.

You can generate your keypair with the command:

```
gcloud compute config-ssh
```

Running Elasticluster on a remote workstation

The first time you run an Elasticluster command that needs to make Compute Engine API calls (such as `elasticluster start`), you will be required to authorize Elasticluster to issue Compute Engine API requests on your behalf.

The authorization flow by default will launch a web browser session on the machine that the Elasticluster command is run on. If that machine (such as a remote workstation or a virtual machine) is not able to open a web browser, the operation will fail with a message like:
If your browser is on a different machine then exit and re-run this application with the command-line parameter

   --noauth_local_webserver

Passing the noauth_local_webserver value to Elasticluster is done by setting the noauth_local_webserver configuration value in the cloud provider section of ~/.elasticluster/config:

```bash
# Create a cloud provider
[cloud/google-cloud]
provider=google
noauth_local_webserver=True
...
```

**Obtaining your client_id and client_secret**

Get your client_id and client_secrets by visiting the following page:

   https://console.cloud.google.com/project/_/apiui/credential

After you select your Google Cloud project, this link will automatically take you to the Credentials tab under the API Manager.

   1. Select New credentials
   2. Select OAuth client ID

If prompted, select Configure consent screen, and follow the instructions to set a “product name” to identify your Cloud project in the consent screen. Choose “Save”.

   3. Under Application Type choose Other
   4. Give your client ID a name, so you can remember why it was created (suggestion: Elasticluster)
   5. Select Create

After successful creation, the interface should display your client ID and client secret.

You can find these values at any time by returning the Credentials tab and clicking on the name you specified in step 4.

**Elasticluster operations**

**Deploy your cluster**

```
elasticluster start gridengine
```

To get verbose output during startup, use the -v flag:

```
elasticluster start -v gridengine
```

**List your cluster instances**

```
elasticluster list-nodes gridengine
```
Copy files to your instances

Elasticluster provides a convenience routine to connect to your frontend instance for SFTP:

```
elasticluster sftp gridengine
```

To script commands for sftp, you can use bash HERE DOCUMENTS:

```
elasticluster sftp gridengine << 'EOF'
put *.sh
EOF
```

See the SFTP man page for more commands.

SSH to your instances

Elasticluster provides a convenience routine to connect to your frontend instance:

```
elasticluster ssh gridengine
```

To connect to other nodes, you can use the `-n` flag command:

```
elasticluster ssh gridengine -n <nodename>
```

Where the nodename is the elasticluster name for the node (such as `compute001`).

Destroy your cluster

```
elasticluster stop gridengine
```

or without prompt:

```
elasticluster stop --yes gridengine
```

Exit the virtualenv

The `activate` command creates a function in the bash environment called `deactivate`. To exit the virtualenv, just execute the command:

```
deactivate
```

Note that any time you want to use elasticluster commands, you must re-activate the virtualenv by sourcing the `activate` script.

Updating your installation

To update your installation, active the virtualenv, pull the source from GitHub, and run the install command again:

```
source elasticluster/bin/activate
cd elasticluster/src
git pull
pip install -e .
```
Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Create a Grid Engine cluster with Preemptible VM workers

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

With Compute Engine Preemptible Virtual Machines you can create and run VMs in the cloud at a much lower price than normal instances. The trade-off for the lower price is that individual instances will run for at most 24 hours.

This trade-off is often a very good fit for distributed batch compute jobs, such as a large Grid Engine job consisting of many small stateless tasks. When a preemptible VM is terminated, only the work of the current task running on it is lost, and the lost task can be requeued for execution on another worker node. The preempted VM can also often be replaced to bring the cluster back to full strength.

This document builds on the instructions to Create a Grid Engine cluster on Compute Engine to create a Grid Engine cluster of preemptible VMs.

Note: The instructions presented here are guidelines that have been used to create clusters up to 100 nodes. However when preemption rates are high, Elasticluster’s re-provisioning of clusters (via Ansible) often converges too slowly due to repeated failures.

For best success with the instructions here, it is recommended to keep cluster sizes to 20 compute nodes or fewer. For larger clusters, use regular (non-preemptible) virtual machines.

Toolset

To run a Grid Engine workload on preemptible VMs, the instructions here employ three tools:

- Elasticluster - to create, configure, and destroy the cluster
- cluster_monitor.sh - to replace TERMINATED virtual machine instances
- array_job_monitor.sh - to requeue failed Grid Engine tasks

Steps

Setting up your cluster

To create a Grid Engine cluster with preemptible VMs, follow the instructions provided in Create a Grid Engine cluster on Compute Engine and configure the compute nodes of your cluster to be preemptible.

A gridengine cluster is composed of one frontend node, and multiple compute nodes. The frontend node should NOT be preemptible. Only the compute nodes should be.

If your cluster is named gridengine then after you Create your cluster definition file, configure the compute nodes to be preemptible by adding the following to ~/.elasticluster/config:
Monitoring your cluster

As the cluster runs, Compute Engine instances will be automatically terminated independently some time within 24 hours of being created. You typically will want the following to happen:

1. TERMINATED instances get removed from the cluster
2. New instances are created to replace TERMINATED instances

This is what the `cluster_monitor.sh` script does.

The cluster monitoring script is available in the `grid-computing-tools` github project.

Downloading the grid-computing-tools

To clone the `grid-computing-tools` github project, issue the following:

```
git clone https://github.com/googlegenomics/grid-computing-tools.git
```

It is recommended that you clone the `grid-computing-tools` project into a sibling directory of your elasticluster directory.

Running cluster_monitor.sh

Be sure that the elasticluster executable is in your PATH. You can do this by setting the PATH explicitly or by activating the `elasticluster virtualenv` in your shell:

```
source elasticluster/bin/active
```

Then run the cluster monitor script:

```
grid-computing-tools/bin/cluster_monitor.sh gridengine
```

The script will run continuously; to terminate the script, hit Ctrl-C.

By default, the monitor will check the cluster status and then sleep for 10 minutes. To change the sleep interval, you can pass an additional argument on the command line, for example:

```
grid-computing-tools/bin/cluster_monitor.sh gridengine 5
```

would sleep for 5 minutes between checks.

To grow your cluster

To increase the number of workers in your cluster while it is running, update the `compute_nodes` value in `~/.elasticluster/config`. For example, to increase the number of compute nodes from the 3 specified in the *Create a Grid Engine cluster on Compute Engine* instructions to 10, set:
The next time the cluster monitor wakes up, it will add nodes to the cluster to reach the new value.

To shrink your cluster

To reduce the number of workers in your cluster while it is running, update the `compute_nodes` value in `~/elasticluster/config`.

As the preemptible VMs are terminated, the cluster monitor will remove them from the cluster, and will only replace instances if the total number in the cluster is less than the configured value. You can also manually terminate instances if desired.

Monitoring your job

Note

Grid Engine provides built-in mechanisms for detecting dead nodes (configured via `reschedule_unknown` and `max_unheard`). In practice this detection and rescheduling of tasks was found to be unreliable.

When nodes are TERMINATED, any tasks running on those nodes need to be restarted. If the TERMINATED node is re-added by the cluster monitor, and the task is NOT submitted for restart, then the new node may sit idle (if the new node has the same name as the TERMINATED node).

Independent of node terminations, tasks can also stall due to programming bugs or unexpected resource contention. Failing to restart stalled tasks results in a node effectively sitting idle.

To detect tasks that need to be restarted, either due to a TERMINATED node or a stalled task, you can use the `array_job_monitor.sh` script in the `grid-computing-tools` github project, which will:

- For each task allocated to a node:
  - Get the associated node’s uptime
    - Restart the task if
      - the node is down
      - the node’s uptime is less than the task’s running time (meaning that the node has been replaced since the task started)
      - the task runtime is longer than a configurable timeout interval (optional)

Note: when you launch your job on the Grid Engine cluster, be sure to mark the job as “restartable”. This can be done by passing the flag `-r y` to the `qsub` command.

Upload the job monitor script

The job monitor script must be run on the cluster’s `frontend` node. To upload `array_job_monitor.sh`:

2.3. Create a Grid Engine cluster with Preemptible VM workers 31
```bash
elasticcluster sftp gridengine << EOF
mkdir tools
put tools/array_job_monitor tools/
EOF
```

### Run the job monitor script

To run the `array_job_monitor.sh`, ssh to the frontend instance:

```bash
elasticcluster ssh gridengine
```

Parameters for `array_job_monitor.sh` are:

- **job_id**  Grid Engine job ID to monitor
- **monitor_interval**  Minutes to sleep between checks of running tasks
  - Default: 15 minutes
- **task_timeout**  Number of minutes a task may run before it is considered stalled, and is eligible to be resubmitted.
  - Default: None
- **queue_name**  Grid Engine job queue the job_id is associated with
  - Default: all.q

For example, to monitor job 1, every 5 minutes, for jobs that should not take more than 10 minutes:

```
./tools/array_job_monitor.sh 1 5 10
```

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

### Run SAMtools to index BAM files in Cloud Storage

Suppose you have thousands of BAMs, which you have stored in Google Cloud Storage, and you need to create index files (BAI) for them.

The Grid Computing Tools github repo provides code and instructions for running SAMtools many times in parallel to create those index files.

#### Overview

This tool takes advantage of two key technologies to process a large number of files:

- Google Compute Engine
- Grid Engine (SGE)
Google Compute Engine provides virtual machines in the cloud. With sufficient quota in your Google Cloud project, you can start dozens or hundreds of instances concurrently. The more instances you add to your cluster, the more quickly you can process your files.

Grid Engine is used to distribute the file operation tasks across all of the instances such that each instance takes the responsibility to download a single file, run the operation, and upload it back to Cloud Storage.

**Directory structure**

To use the tool, you will need to download both the Grid Computing Tools github repo and the Elasticluster repo to your local workstation or laptop.

No specific relationship exists between these two repositories, but in the following instructions, it is assumed that the directories:

- grid-computing-tools
- elasticluster

are siblings under a workspace root (WS_ROOT) directory.

**Running the samples**

The quickest way to get familiar with the samtools tool is by trying the sample.

The sample provided here lists just 6 files to work on, and the instructions below demonstrate spreading the processing over 3 worker instances.

1. **Create a cluster of Compute Engine instances running Grid Engine**
   
   In your current shell:
   
   (a) cd ${WS_ROOT}
   
   (b) Follow the instructions to *Create a Grid Engine cluster on Compute Engine*

2. **Download the grid-computing-tools repository (if you have not already done so)**

   ```
   cd ${WS_ROOT}
   git clone https://github.com/googlegenomics/grid-computing-tools.git
   ```

3. **Upload the src and samples directories to the Grid Engine master instance:**

   ```
   cd grid-computing-tools
   elasticluster sftp gridengine << 'EOF'
   mkdir src
   mkdir src/common
   mkdir src/samtools
   put src/common/* src/common/
   put src/samtools/* src/samtools/
   mkdir samples
   mkdir samples/samtools
   put samples/samtools/* samples/samtools/
   EOF
   ```

4. **SSH to the master instance**

2.4. Run SAMtools to index BAM files in Cloud Storage
5. **Set up the configuration files for the samples**

The syntax for running the sample is:

```
./src/samtools/launch_samtools.sh [config_file]
```

The `config_file` lists two sets of key parameters:

- What operation to perform
- What are the source and destination locations

The operation to perform is controlled by the following:

- **SAMTOOLS_OPERATION**: index

The locations are determined by:

- **INPUT_LIST_FILE**: file containing a list of GCS paths to the input files to process
- **OUTPUT_PATH**: GCS path indicating where to upload the output files. If set to `source`, the output will be written to the same path as the source file (with the extension `.bai` appended)
- **OUTPUT_LOG_PATH**: (optional) GCS path indicating where to upload log files

To use the samples, you must update the **OUTPUT_PATH** and **OUTPUT_LOG_PATH** to contain a valid GCS bucket name. The sample config file sets a placeholder for the **OUTPUT_PATH** and **OUTPUT_LOG_PATH** such as:

```
export OUTPUT_PATH=gs://MY_BUCKET/output_path/samtools_index
export OUTPUT_LOG_PATH=gs://MY_BUCKET/log_path/samtools_index
```

You can do this manually with the editor of your choice or you can change the config file with the command:

```
sed --in-place -e 's#MY_BUCKET#your_bucket#' samples/samtools/*_config.sh
```

Where `your_bucket` should be replaced with the name of a GCS bucket in your Cloud project to which you have write access.

6. **Run the sample**:

   - Index a list of files using `samtools index` [Estimated time to complete: 5 minutes ]

```
./src/samtools/launch_samtools.sh ./samples/samtools/samtools_index_config.sh
```

When successfully launched, Grid Engine should emit a message such as:

```
Your job-array 1.1-6:1 ("samtools") has been submitted
```

This message tells you that the submitted job is a gridengine array job. The above message indicates that the job id is 1 and that the tasks are numbered 1 through 6. The name of the job `samtools` is also indicated.

7. **Monitoring the status of your job**

Grid Engine provides the `qstat` command to get the status of the execution queue.

While the job is in the queue, the `state` column will indicate the status of each task. Tasks not yet allocated to a compute node will be collapsed into a single row as in the following output:
The above output indicates that tasks 1-6 of job 1 are all in a qw (queue waiting) state.

When tasks get allocated, the output will look something like:

```
$ qstat
+-----------------+----------+----------+----------------+-------------------+----------------+---------+----------------+-------------------+-------------------+
| job-ID | prior | name   | user    | state | submit/start at | queue | slots | ja-task-ID | job-ID | prior | name   | user    | state | submit/start at | queue | slots | ja-task-ID |
+--------+-------+--------+---------+-------+----------------+-------+-------+------------+--------+-------+--------+---------+-------+----------------+-------+-------+------------+
| 1      |       | my-job | janedoe | qw    | 06/16/2015 18:03:32 |       | 6     | 1-6:1     | 1      |       | my-job | janedoe | r      | 06/16/2015 18:03:45 | all.q@compute002 |       |            |
|        |       |        |         |       |                |       |       |            |        |       |        |         |        |                |       |       |            |
| 1      | 0.50000 | my-job | janedoe | r     | 06/16/2015 18:03:45 | all.q@compute001 |       |            | 1      | 0.50000 | my-job | janedoe | r     | 06/16/2015 18:03:45 | all.q@compute003 |       |            |
| 1      | 0.50000 | my-job | janedoe | r     | 06/16/2015 18:03:45 | all.q@compute003 |       |            | 1      | 0.00000 | my-job | janedoe | qw    | 06/16/2015 18:03:32 |       | 6     | 1-6:1     |
+--------+-------+--------+---------+-------+----------------+-------+-------+------------+--------+-------+--------+---------+-------+----------------+-------+-------+------------+
```

which indicates tasks 1-3 are all in the r (running) state, while tasks 4-6 remain in a waiting state.

When all tasks have completed qstat will produce no output.

8. Checking the logging output of tasks

Each gridengine task will write to an “output” file and an “error” file. These files will be located in the directory the job was launched from (the HOME directory). The files will be named respectively:

- `job_name.ojob_id.task_id` (for example: `my-job.o1.10`)
- `job_name.ejob_id.task_id` (for example: `my-job.e1.10`)

The error file will contain any unexpected error output, and will also contain any download and upload logging from gsutil.

9. Viewing the results of the jobs

When tasks complete, the result files are uploaded to GCS. You can view the list of output files with gsutil ls, such as:

```
gsutil ls OUTPUT_PATH
```

Where the `OUTPUT_PATH` should be the value you specified in the job config file (step 6 above).

10. Viewing log files

When tasks complete, the result log files are uploaded to GCS if `OUTPUT_LOG_PATH` was set in the job config file. The log files can be of value both to verify success/failure of all tasks, as well as to gather some performance statistics before starting a larger job.

- Count number of successful tasks

```
gsutil cat OUTPUT_LOG_PATH/* | grep SUCCESS | wc -l
```

Where the `OUTPUT_LOG_PATH` should be the value you specified in the job config file (step 6 above).
**Google Genomics Documentation, Release v1**

- Count number of failed tasks

  ```bash
  gsutil cat OUTPUT_LOG_PATH/* | grep FAILURE | wc -l
  ```

  Where the `OUTPUT_LOG_PATH` should be the value you specified in the job config file (step 6 above).

- Compute total task time

  ```bash
  gsutil cat OUTPUT_LOG_PATH/* | \
  sed -n -e 's#^Task time.*: \([0-9]*\) seconds\1#p' | \
  awk '{ sum += $1; } END { print sum/NR " seconds"}'}
  ```

- Compute average task time

  ```bash
  gsutil cat OUTPUT_LOG_PATH/* | \
  sed -n -e 's#^Task time.*: \([0-9]*\) seconds\1#p' | \
  awk '{ sum += $1; } END { print sum " seconds"}'}
  ```

11. **Destroying the cluster**

   When you are finished running the samples, disconnect from the master instance and from your workstation shut down the gridengine cluster:

   ```bash
   elasticluster stop gridengine
   ```

**Running your own job**

To run your own job to index a list of BAM files requires the following:

1. Create an input list file
2. Create a job config file
3. Create a gridengine cluster with sufficient disk space attached to each compute node
4. Upload input list file, config file, and `grid-computing-tools` source to the gridengine cluster master
5. Do a “dry run” *(optional)*
6. Do a “test run” *(optional)*
7. Launch the job

The following instructions provide guidance on each of these steps. It is recommended, though not a requirement, that you save your `input list file` and `job config file` to a directory outside the `grid-computing-tools` directory. For example, you might create a directory `${WS_ROOT}/my_jobs`.

1. **Create an input list file**

   If all of your input files appear in a single directory, then the easiest way to generate a file list is with `gsutil`. For example:

   ```bash
   gsutil ls gs://MY_BUCKET(PATH/.*.bam > ${WS_ROOT}/my_jobs/bam_indexing_list_file.txt
   ```

2. **Create a job config file**

   The easiest way to create a job config file is to base it off the sample and update:

   - `INPUT_LIST_FILE`
   - `OUTPUT_PATH`
• **OUTPUT_LOG_PATH**

To have the generated BAM index file written to the same location as the source BAM, set:

```
OUTPUT_PATH=source
```

Save the job config file to `${WS_ROOT}/my_jobs/`.

3. **Create a gridengine cluster with sufficient disk space attached to each compute node**

   (a) **Determine disk size requirements**

   Each compute node will require sufficient disk space to hold the input and output files for its current task. Determine the largest file in your input list and estimate the total space you will need. It may be necessary to download the file and perform the operation manually to get a maximum combined input and output size.

   Persistent disk performance also scales with the size of the volume. Independent of storage requirements, for consistent throughput on long running jobs, use a standard persistent disk of at least 1TB, or use SSD persistent disk. More documentation is available for selecting the right persistent disk.

   (b) **Verify or increase quota**

   Your choice for number of nodes and disk size must take into account your Compute Engine resource quota for the region of your cluster.

   Quota limits and current usage can be viewed with `gcloud compute`:

   ```
gcloud compute regions describe *region*
```

   or in Cloud Platform Console:

   ```
https://console.cloud.google.com/project/_/compute/quotas
```

   Important quota limits include CPUs, in-use IP addresses, and disk size.

   To request additional quota, submit the Compute Engine quota request form.

   (c) **Configure your cluster**

   Instructions for setting the boot disk size for the compute nodes of your cluster can be found at *Setting the boot disk size*.

   You will likely want to set the number of compute nodes for your cluster to a number higher than the 3 specified in the example cluster setup instructions.

   Once configured, start your cluster.

4. **Upload input list file, config file, and grid-computing-tools source to the gridengine cluster master**

```
elasticcluster sftp gridengine << EOF
put ../my_jobs/*
mkdir src
mkdir src/common
mkdir src/samtools
put src/common/* src/common/
put src/samtools/* src/samtools/
EOF
```

5. **Do a “dry run” (optional)**

The tool supports the `DRYRUN` environment variable. Setting this value to 1 when launching your job will cause the queued job to execute without downloading or uploading any files.
The local output files, however, will be populated with useful information about what files would be copied. This can be useful for ensuring your file list is valid and that the output path is correct.

For example:

```bash
$ DRYRUN=1 ./src/samtools/launch_samtools.sh ./samples/samtools/samtools_index_config.sh
Your job-array 2.1-6:1 ("samtools") has been submitted
```

Then after waiting for the job to complete, inspect:

```
$ head -n 5 samtools.o3.1
Task host: compute002
Task start: 1
Input list file: ./samples/samtools/samtools_index_file_list.txt
Output path: gs://cookbook-bucket/output_path/samtools_index
Output log path: gs://cookbook-bucket/log_path/samtools_index

$ grep "\^Will download:" samtools.o3.*
```

<table>
<thead>
<tr>
<th>Task</th>
<th>Download Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>samtools.o3.2</td>
<td>gs://genomics-public-data/ftp-trace.ncbi.nih.gov/1000genomes/ftp/technical/pilot2_high_cov_GRCh37_bams/data/NA12878/alignment/NA12878.chrom1.LS454.ssaha2.CEU.high_coverage.20100311.bam to /scratch/samtools.3.2/in/</td>
</tr>
</tbody>
</table>

```
$ grep "\^Will upload:" samtools.o3.*
```

<table>
<thead>
<tr>
<th>Task</th>
<th>Upload Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>samtools.o3.1</td>
<td>/scratch/samtools.3.1/in/NA12878.chrom9.SOLID.bfast.CEU.high_coverage.20100125.bam.bai to gs://cookbook-bucket/output_path/samtools_index/</td>
</tr>
<tr>
<td>samtools.o3.2</td>
<td>/scratch/samtools.3.2/in/NA12878.chrom1.LS454.ssaha2.CEU.high_coverage.20100311.bam.bai to gs://cookbook-bucket/output_path/samtools_index/</td>
</tr>
<tr>
<td>samtools.o3.3</td>
<td>/scratch/samtools.3.3/in/NA12878.chrom11.SOLID.corona.SRP000032.2009_08.bam.bai to gs://cookbook-bucket/output_path/samtools_index/</td>
</tr>
<tr>
<td>samtools.o3.4</td>
<td>/scratch/samtools.3.4/in/NA12878.chrom12.SOLID.corona.SRP000032.2009_08.bam.bai to gs://cookbook-bucket/output_path/samtools_index/</td>
</tr>
<tr>
<td>samtools.o3.5</td>
<td>/scratch/samtools.3.5/in/NA12878.chrom10.SOLID.corona.SRP000032.2009_08.bam.bai to gs://cookbook-bucket/output_path/samtools_index/</td>
</tr>
<tr>
<td>samtools.o3.6</td>
<td>/scratch/samtools.3.6/in/NA12878.chromX.SOLID.corona.SRP000032.2009_08.bam.bai to gs://cookbook-bucket/output_path/samtools_index/</td>
</tr>
</tbody>
</table>

6. **Do a “test run” (optional)**

The tool supports environment variables to indicate which lines in the input list to run:
• LAUNCH_MIN - lowest line number to process
• LAUNCH_MAX - highest line number to process

For example to launch a Grid Engine array job that only processes line 1:

```bash
$ LAUNCH_MIN=1 LAUNCH_MAX=1 ./src/samtools/launch_samtools.sh ./samples/samtools/
→ samtools_index_config.sh
Your job-array 5.1-1:1 ("samtools") has been submitted
```

The LAUNCH_MIN and LAUNCH_MAX values can be used with the DRYRUN environment variable:

```bash
$ DRYRUN=1 LAUNCH_MIN=1 LAUNCH_MAX=5 ./src/samtools/launch_samtools.sh ./samples/
→ samtools/samtools_index_config.sh
Your job-array 6.1-5:1 ("samtools") has been submitted
```

7. Launch the job

On the master instance, run the launch script, passing in the config file:

```bash
./src/samtools/launch_samtools.sh my_job_config.sh
```

where `my_job_config.sh` is replaced by the name of your config file created in step 2.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

---

**Compress/Decompress files in Cloud Storage**

Suppose you have thousands of VCFs, which you have stored *compressed* in Google Cloud Storage, and you need to perform some operation on them in their *decompressed* state.

A few examples:

- You want to run some check or update on all of the headers
- You want to import them into Google Genomics

Or suppose you have thousands of VCFs, and you did not compress them when originally copying them to Google Cloud Storage, but these VCFs can now be compressed and archived.

The `compress` tool available in the Grid Computing Tools github repo can be used for any of these situations if your compression scheme is either `gzip` or `bzip2`.

**Overview**

This tool takes advantage of two key technologies to process a large number of files:

- Google Compute Engine
- Grid Engine (SGE)
Google Compute Engine provides virtual machines in the cloud. With sufficient quota in your Google Cloud project, you can start dozens or hundreds of instances concurrently. The more instances you add to your cluster, the more quickly you can process your files.

Grid Engine is used to distribute the file operation tasks across all of the instances such that each instance takes the responsibility to download a single file, run the operation, and upload it back to Cloud Storage.

**Workstation directory structure**

To use the tool, you will need to download both the Grid Computing Tools github repo and the Elasticluster repo to your local workstation or laptop.

No specific relationship exists between these two repositories, but in the following instructions, it is assumed that the directories:

- grid-computing-tools
- elasticluster

are siblings under a workspace root (WS_ROOT) directory.

**Running the samples**

The quickest way to get familiar with the compress tool is by trying one or more of the samples. Samples are provided for the following uses:

- Download bzip2-compressed files from Cloud Storage, decompress them, and upload the results into Cloud Storage
- Download decompressed files from Cloud Storage, compress them with bzip2, and upload the results into Cloud Storage
- Download gzip-compressed files from Cloud Storage, decompress them, and upload the results into Cloud Storage
- Download decompressed files from Cloud Storage, compress them with gzip, and upload the results into Cloud Storage

The samples provided here each list just 6 files to work on, and the instructions below demonstrate spreading the processing over 3 worker instances.

1. **Create a cluster of Compute Engine instances running Grid Engine**
   
   In your current shell:
   
   (a) cd ${WS_ROOT}
   
   (b) Follow the instructions to *Create a Grid Engine cluster on Compute Engine*

2. **Download the grid-computing-tools repository (if you have not already done so)**

   ```
   cd ${WS_ROOT}
   git clone https://github.com/googlegenomics/grid-computing-tools.git
   ```

3. **Upload the src and samples directories to the Grid Engine master instance:**

   ```
   cd grid-computing-tools
   elasticluster sftp gridengine << 'EOF'
   mkdir src
   EOF
   ```
mkdir src/common
mkdir src/compress
put src/common/* src/common/
put src/compress/* src/compress/
mkdir samples
mkdir samples/compress
put samples/compress/* samples/compress/

4. SSH to the master instance

```
elasticluster ssh gridengine
```

5. Set up the configuration files for the samples

The syntax for running each of the samples is the same:

```
./src/compress/launch_compress.sh [config_file]
```

The `config_file` lists two sets of key parameters:

- What operation to perform
- What are the source and destination locations

The operation to perform is controlled by the following:

- `COMPRESS_OPERATION`: `compress` or `decompress`
- `COMPRESS_TYPE`: `bzip2` or `gzip`
- `COMPRESS_EXTENSION`: Typically `.bz2` or `.gz`

The locations are determined by:

- `INPUT_LIST_FILE`: file containing a list of GCS paths to the input files to process
- `OUTPUT_PATH`: GCS path indicating where to upload the output files
- `OUTPUT_LOG_PATH`: (optional) GCS path indicating where to upload log files

To use the samples, you must update the `OUTPUT_PATH` and `OUTPUT_LOG_PATH` to contain a valid GCS bucket name. Each of the sample config files sets a placeholder for the `OUTPUT_PATH` and `OUTPUT_LOG_PATH` such as:

```
export OUTPUT_PATH=gs://MY_BUCKET/output_path/compress_bzip2
export OUTPUT_LOG_PATH=gs://MY_BUCKET/log_path/compress_bzip2
```

You can do this manually with the editor of your choice or you can change all of the `config` files at once with the command:

```
sed --in-place -e 's#MY_BUCKET#your_bucket#' samples/compress/*_config.sh
```

Where `your_bucket` should be replaced with the name of a GCS bucket in your Cloud project to which you have write access.

6. Run the sample:

You can run all of the samples, or just those that model your particular use-case.

- Compress a list of files using bzip2 [ Estimated time to complete: 35 minutes ]
• Decompress a list of files using bzip2 [Estimated time to complete: 4 minutes]

• Compress a list of files using gzip [Estimated time to complete: 15 minutes]

• Decompress a list of files using gzip [Estimated time to complete: 5 minutes]

When successfully launched, Grid Engine should emit a message such as:

Your job-array 1.1-6:1 ("compress") has been submitted

This message tells you that the submitted job is a gridengine array job. The above message indicates that the job id is 1 and that the tasks are numbered 1 through 6. The name of the job, compress, is also indicated.

7. Monitoring the status of your job

Grid Engine provides the qstat command to get the status of the execution queue.

While the job is in the queue, the state column will indicate the status of each task. Tasks not yet allocated to a compute node will be collapsed into a single row as in the following output:

```
$ qstat
job-ID prior name user     state submit/start at  queue
←slots ja-task-ID
----------------------------------------------------------------------------------
  1 0.00000 my-job janedoe qw 06/16/2015 18:03:32
  1 1-6:1

The above output indicates that tasks 1-6 of job 1 are all in a qw (queue waiting) state.
```

When tasks get allocated, the output will look something like:

```
$ qstat
job-ID prior name user     state submit/start at  queue
←slots ja-task-ID
----------------------------------------------------------------------------------
  1 0.50000 my-job janedoe r  06/16/2015 18:03:45 all.q@compute002
  1 1
  1 0.50000 my-job janedoe r 06/16/2015 18:03:45 all.q@compute001
  1 2
  1 0.50000 my-job janedoe r 06/16/2015 18:03:45 all.q@compute003
  1 3
  1 0.00000 my-job janedoe qw 06/16/2015 18:03:32
  1 4-6:1
```

which indicates tasks 1-3 are all in the r (running) state, while tasks 4-6 remain in a waiting state.

When all tasks have completed qstat will produce no output.

8. Checking the logging output of tasks
Each gridengine task will write to an “output” file and an “error” file. These files will be located in the directory the job was launched from (the HOME directory). The files will be named respectively:

- `job_name.job_id.task_id` (for example: `my-job.01.10`)
- `job_name.job_id.task_id` (for example: `my-job.e1.10`)

The error file will contain any unexpected error output, and will also contain any download and upload logging from gsutil.

9. **Viewing the results of the jobs**

When tasks complete, the result files are uploaded to GCS. You can view the list of output files with `gsutil ls`, such as:

```
gsutil ls OUTPUT_PATH
```

Where the `OUTPUT_PATH` should be the value you specified in the job config file (step 6 above).

10. **Viewing log files**

When tasks complete, the result log files are uploaded to GCS if `OUTPUT_LOG_PATH` was set in the job config file. The log files can be of value both to verify success/failure of all tasks, as well as to gather some performance statistics before starting a larger job.

- Count number of successful tasks
  
  ```
gsutil cat OUTPUT_LOG_PATH/* | grep SUCCESS | wc -l
  ```

  Where the `OUTPUT_LOG_PATH` should be the value you specified in the job config file (step 6 above).

- Count number of failed tasks
  
  ```
gsutil cat OUTPUT_LOG_PATH/* | grep FAILURE | wc -l
  ```

  Where the `OUTPUT_LOG_PATH` should be the value you specified in the job config file (step 6 above).

- Compute total task time
  
  ```
gsutil cat OUTPUT_LOG_PATH/* | \
  sed -n -e 's#^Task time.*: \([0-9]*\) seconds\#\1#p' | \
  awk '{ sum += $1; } END { print sum/NR " seconds"}'
  ```

- Compute average task time
  
  ```
gsutil cat OUTPUT_LOG_PATH/* | \
  sed -n -e 's#^Task time.*: \([0-9]*\) seconds\#\1#p' | \
  awk '{ sum += $1; } END { print sum " seconds"}'
  ```

11. **Destroying the cluster**

When you are finished running the samples, disconnect from the master instance and from your workstation shut down the gridengine cluster:

```
elasticluster stop gridengine
```

**Running your own job**

To run your own job to compress/decompress a list of files requires the following:

1. Create an input list file
2. Create a job config file
3. Create a gridengine cluster with sufficient disk space attached to each compute node
4. Upload input list file, config file, and grid-computing-tools source to the gridengine cluster master
5. Do a “dry run” (optional)
6. Do a “test run” (optional)
7. Launch the job

The following instructions provide guidance on each of these steps. It is recommended, though not a requirement, that you save your input list file and job config file to a directory outside the grid-computing-tools directory. For example, you might create a directory `$(WS_ROOT)/my_jobs`.

1. Create an input list file

   If all of your input files appear in a single directory, then the easiest way to generate a file list is with `gsutil`. For example:

   ```
   gsutil ls gs://MY_BUCKET/PATH/*.vcf.bz2 > $(WS_ROOT)/my_jobs/compressed_vcf_list_file.txt
   ```

2. Create a job config file

   The easiest way to create a job config file is to base it off the appropriate sample and update
   
   - `INPUT_LIST_FILE`
   - `OUTPUT_PATH`
   - `OUTPUT_LOG_PATH`

   Save the job config file to `$(WS_ROOT)/my_jobs/`.

3. Create a gridengine cluster with sufficient disk space attached to each compute node

   (a) Determine disk size requirements

   Each compute node will require sufficient disk space to hold the input and output files for its current task. Determine the largest file in your input list and estimate the total space you will need. It may be necessary to download the file and perform the operation manually to get a maximum combined input and output size.

   Persistent disk performance also scales with the size of the volume. Independent of storage requirements, for consistent throughput on long running jobs, use a standard persistent disk of at least 1TB, or use SSD persistent disk. More documentation is available for selecting the right persistent disk.

   (b) Verify or increase quota

   Your choice for number of nodes and disk size must take into account your Compute Engine resource quota for the region of your cluster.

   Quota limits and current usage can be viewed with `gcloud compute`:

   ```
   gcloud compute regions describe *region*
   ```

   or in Cloud Platform Console:

   `https://console.cloud.google.com/project/_/compute/quotas`

   Important quota limits include CPUs, in-use IP addresses, and disk size.

   To request additional quota, submit the Compute Engine quota request form.
(c) Configure your cluster

Instructions for setting the boot disk size for the compute nodes of your cluster can be found at Setting the boot disk size.

You will likely want to set the number of compute nodes for your cluster to a number higher than the 3 specified in the example cluster setup instructions.

Once configured, start your cluster.

4. Upload input list file, config file, and grid-computing-tools source to the gridengine cluster master

```
elasticcluster sftp gridengine << EOF
put ../my_jobs/*
mkdir src
mkdir src/common
mkdir src/compress
put src/common/* src/common/
put src/compress/* src/compress/
EOF
```

5. Do a “dry run” (optional)

The tool supports the DRYRUN environment variable. Setting this value to 1 when launching your job will cause the queued job to execute without downloading or uploading any files.

The local output files, however, will be populated with useful information about what files would be copied. This can be useful for ensuring your file list is valid and that the output path is correct.

For example:

```
$ DRYRUN=1 ./src/compress/launch_compress.sh ./samples/compress/gzip_compress_file_list.txt
Your job-array 5.1-6:1 ("compress") has been submitted
```

Then after waiting for the job to complete, inspect:

```
$ head -n 5 compress.o3.1
Task host: compute001
Task start: 1
Input list file: ./samples/compress/gzip_compress_file_list.txt
Output path: gs://cookbook-bucket/output_path/compress_gzip
Output log path: gs://cookbook-bucket/log_path/compress_gzip

$ grep "^Will download:" compress.o5.*
compress.o5.1:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12877_S1.genome.vcf to /scratch/compress.5.1/in/
compress.o5.2:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12878_S1.genome.vcf to /scratch/compress.5.2/in/
compress.o5.3:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12879_S1.genome.vcf to /scratch/compress.5.3/in/
compress.o5.4:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12880_S1.genome.vcf to /scratch/compress.5.4/in/
compress.o5.5:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12881_S1.genome.vcf to /scratch/compress.5.5/in/
compress.o5.6:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12882_S1.genome.vcf to /scratch/compress.5.6/in/
```

6. Do a “test run” (optional)

The tool supports environment variables to indicate which lines in the input list to run:

```
$ head -n 5 compress.o3.1
Task host: compute001
Task start: 1
Input list file: ./samples/compress/gzip_compress_file_list.txt
Output path: gs://cookbook-bucket/output_path/compress_gzip
Output log path: gs://cookbook-bucket/log_path/compress_gzip

$ grep "^Will download:" compress.o5.*
compress.o5.1:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12877_S1.genome.vcf to /scratch/compress.5.1/in/
compress.o5.2:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12878_S1.genome.vcf to /scratch/compress.5.2/in/
compress.o5.3:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12879_S1.genome.vcf to /scratch/compress.5.3/in/
compress.o5.4:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12880_S1.genome.vcf to /scratch/compress.5.4/in/
compress.o5.5:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12881_S1.genome.vcf to /scratch/compress.5.5/in/
compress.o5.6:Will download: gs://genomics-public-data/platinum-genomes/vcf/NA12882_S1.genome.vcf to /scratch/compress.5.6/in/
```
• LAUNCH_MIN - lowest line number to process
• LAUNCH_MAX - highest line number to process

For example to launch a Grid Engine array job that only processes line 1:

```
$ LAUNCH_MIN=1 LAUNCH_MAX=1 ./src/compress/launch_compress.sh ./samples/compress/
→gzip_compress_config.sh
Your job-array 5.1-1:1 ("compress") has been submitted
```

The LAUNCH_MIN and LAUNCH_MAX values can be used with the DRYRUN environment variable:

```
$ DRYRUN=1 LAUNCH_MIN=1 LAUNCH_MAX=1 ./src/compress/launch_compress.sh ./samples/
→compress/gzip_compress_config.sh
Your job-array 6.1-5:1 ("compress") has been submitted
```

7. **Launch the job**

On the master instance, run the launch script, passing in the config file:

```
./src/compress/launch_compress.sh my_job_config.sh
```

where `my_job_config.sh` is replaced by the name of your config file created in step 2.

---

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

---

**Run Galaxy on Compute Engine**

The DREAM Tumor Heterogeneity challenge using Galaxy and Docker is in full swing now. See the quick-start videos for how to set up Galaxy on Google Compute Engine.

---

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

---

**Run NCBI BLAST on Compute Engine**

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

The Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences. The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches. BLAST can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families.

• For more information, see NCBI BLAST Cloud Documentation and NCBI BLAST.
• To deploy BLAST to Google Compute Engine, you can click-to-deploy NCBI BLAST.
Run Bioconductor on Compute Engine

This will create a virtual machine on Google Cloud Platform with a locked down network (only SSH port 22 open). Your local machine will securely connect to the VM via an ssh tunnel.

Within the docker container the directory `/home/rstudio/data` will correspond to directory `/mnt/data` on the virtual machine. This is where the persistent data disk is attached to the VM. Store important files there. Docker containers are stateless, so if the container restarts for any reason, then files you created within the container will be lost.

Bioconductor maintains Docker containers with R, Bioconductor packages, and RStudio Server all ready to go! Its a great way to set up your R environment quickly and start working. The instructions to deploy it to Google Compute Engine are below but if you want to learn more about these containers, see http://www.bioconductor.org/help/docker/.

1. Click on click-to-deploy Bioconductor to navigate to the launcher page on the Cloud Platform Console.
2. Optional: change the Machine type if you would like to deploy a machine with more CPU cores or RAM.
3. Optional: change the Data disk size (GB) if you would like to use a larger persistent disk for your own files.
4. Optional: change Docker image if you would like to run a container with additional Bioconductor packages preinstalled.
5. Click on the Deploy Bioconductor button.
6. Follow the post-deployment instructions to log into RStudioServer via your browser!

If you want to deploy a different docker container, such as the one from BioC 2015: Where Software and Biology Connect or from https://github.com/isb-cgc/examples-R:

1. In field Docker Image choose item custom.
2. Click on More to display the additional form fields.
3. In field Custom docker image paste in the docker image path, such as `gcr.io/bioc_2015/devel_sequencing` or `b.gcr.io/isb-cgc-public-docker-images/r-examples`.

Change your virtual machine type (number of cores, amount of memory)

1. First, make sure results from your current R session are saved to the data disk (underneath `/home/rstudio/data`) or another location outside of the container.
2. Follow these instructions to stop, resize, and start your VM: https://cloud.google.com/compute/docs/instances/changing-machine-type-of-stopped-instance

Details

This will create a virtual machine on Google Cloud Platform with a locked down network (only SSH port 22 open). Your local machine will securely connect to the VM via an ssh tunnel.

Within the docker container the directory `/home/rstudio/data` will correspond to directory `/mnt/data` on the virtual machine. This is where the persistent data disk is attached to the VM. Store important files there. Docker containers are stateless, so if the container restarts for any reason, then files you created within the container will be lost.

Bioconductor maintains Docker containers with R, Bioconductor packages, and RStudio Server all ready to go! Its a great way to set up your R environment quickly and start working. The instructions to deploy it to Google Compute Engine are below but if you want to learn more about these containers, see http://www.bioconductor.org/help/docker/.

1. Click on click-to-deploy Bioconductor to navigate to the launcher page on the Cloud Platform Console.
2. Optional: change the Machine type if you would like to deploy a machine with more CPU cores or RAM.
3. Optional: change the Data disk size (GB) if you would like to use a larger persistent disk for your own files.
4. Optional: change Docker image if you would like to run a container with additional Bioconductor packages preinstalled.
5. Click on the Deploy Bioconductor button.
6. Follow the post-deployment instructions to log into RStudioServer via your browser!

If you want to deploy a different docker container, such as the one from BioC 2015: Where Software and Biology Connect or from https://github.com/isb-cgc/examples-R:

1. In field Docker Image choose item custom.
2. Click on More to display the additional form fields.
3. In field Custom docker image paste in the docker image path, such as `gcr.io/bioc_2015/devel_sequencing` or `b.gcr.io/isb-cgc-public-docker-images/r-examples`.

Change your virtual machine type (number of cores, amount of memory)

1. First, make sure results from your current R session are saved to the data disk (underneath `/home/rstudio/data`) or another location outside of the container.
2. Follow these instructions to stop, resize, and start your VM: https://cloud.google.com/compute/docs/instances/changing-machine-type-of-stopped-instance
“Stop” or “Delete” your virtual machine

If you would like to pause your VM when not using it:

1. Go to the Google Cloud Platform Console and select your project: https://console.cloud.google.com/project/_/compute/instances
2. Click on the checkbox next to your VM.
3. Click on Stop to pause your VM.
4. When you are ready to use it again, Start your VM. For more detail, see: https://cloud.google.com/compute/docs/instances/stopping-or-deleting-an-instance

If you want to delete your deployment:

1. First copy any data off of the data disk that you wish to keep. The data disk will be deleted when the deployment is deleted.
2. Click on Deployments to navigate to your deployment and delete it.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Run iPython Notebooks on Compute Engine

The properly rendered version of this document can be found at Read The Docs.
If you are reading this on github, you should instead click here.

Google Cloud Datalab is built on Jupyter (formerly IPython) and enables analysis of your data in Google BigQuery, Google Compute Engine, Google Cloud Storage, and Google Genomics using Python, SQL, and JavaScript (for BigQuery user-defined functions).

Read Example Notebooks

There are several example notebooks for genomics use cases upon public data such as the Illumina Platinum Genomes, 1,000 Genomes, and TCGA Cancer Genomics Data in the Cloud. You can read them on github:

- Exploring Genomic Data
- Explore the 1000 Genomes Sample Information
- Genome-wide association study
- Getting started with the Google Genomics API
- And find more sample notebooks in:
  - https://github.com/googlegenomics/datalab-examples
  - https://github.com/GoogleCloudPlatform/datalab
  - https://github.com/ibb-cgc/examples-Python
  - https://github.com/googlegenomics/linkage-disequilibrium
Run Notebooks

To run the examples yourself:

1. Launch your own Cloud Datalab instance in the cloud or run it locally.
2. Work through the introductory notebooks that are pre-installed on Cloud Datalab.
3. Run `git clone https://github.com/googlegenomics/datalab-examples.git` on your local file system to download the notebooks.
4. Import the genomics notebooks into your Cloud Datalab instance by navigating to the notebook list page and uploading them.

If you are running in the cloud, be sure to shut down Cloud Datalab when you are no longer using it. Shut down instructions and other tips are here.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
CHAPTER 3

Access Genomic Data using...

Integrative Genomics Viewer (IGV)

IGV Web

igv.js is an embeddable interactive genome visualization component. The source code for igv.js is available on github:

https://github.com/igvteam/igv.js

Documentation is available at:

https://github.com/igvteam/igv.js/wiki

Try it now with public data in Google Genomics:

http://igv.org/web/examples/google-demo.html

IGV Desktop

IGV Desktop supports browsing of reads from the Google Genomics Reads API and also from BAM files in Google Cloud Storage. It implements an OAuth flow to facilitate access to private data in addition to public data.

Setup

1. Install or upgrade IGV Desktop to ensure you have a recent version. IGV Desktop can be obtained from http://www.broadinstitute.org/software/igv/download
2. Change preferences so that the Google menu is displayed. Choose menu item View -> Preferences and check the box next to Enable Google access.

3. Log into Google. Choose menu item Google -> Login ... and follow the OAuth prompts.

View a Google Genomics ReadGroupSet

Choose menu item Google -> Load Genomics ReadGroupSet and enter the readGroupSet ID for the readGroupSet you wish to view. For example, a readGroupSet ID of CMvnhpKTFhD3he72j4KZuyc will display the reads for NA12877 from Illumina Platinum Genomes.

View a BAM from Google Cloud Storage

Choose menu item File -> Load from URL and enter the Google Cloud Storage path for the BAM you wish to view. For example, a path of gs://genomics-public-data/platinum-genomes.bam/NA12877_S1.
Run Picard and GATK tools on Cloud-Resident Genomic Data

**Introduction**

Picard/GATK tools are command line utilities for genomic sequencing data processing that typically take BAM and other files as input and produce modified BAM files. These tools are frequently chained together into pipelines to perform step-by-step processing of the sequencing data all the way from unaligned sequencer output to variant calls (e.g. see Broad best practices).

We are teaching these tools to take cloud based datasets as a possible input. The foundation for cloud data access is now in HTSJDK library and we have converted a number of Picard tools.

If your dataset is loaded into a cloud provider supporting GA4GH API (e.g. Google Genomics) or you use one of the available datasets from Discover Published Data, you will be able to run a Picard tool against it, reading data directly from the cloud.

**New:** see the video version of this tutorial.

Below is a step by step guide on how to build Picard tools with GA4GH support, set-up access to genomics data in the cloud and run the tools.

By the end of this tutorial you will be able run a Picard tool, giving it a URL identifying a genomic dataset in the cloud and see the output of processing the data directly from the cloud.

We also have detailed description of changes to HTSJDK and Picard to help you write your own cloud-aware client on top of HTSDK or help us convert more Picard tools.

**Set up access to genomics data in Google Cloud**

We will assume you are starting from a completely blank slate so please skip the steps that are redundant for you.

If you are already using Google Genomics API and have a project set up for this you can skip this section and go directly to Build Picard tools with GA4GH support.

These instructions are based on Genomics Tools tutorial.
Set up your account and a cloud project

If you don’t already have one, sign up for a Google Account

Details

If you already have a Google Cloud Platform project, this link will take you to your list of projects.

Sign up for Google Cloud Platform by clicking on this link: https://console.cloud.google.com/billing/freetrial
If you do not yet have a cloud project, create a Genomics and Cloud Storage enabled project via the Google Cloud Platform Console.

Install gcloud tool and validate access to genomics data

If you have not done so before, install gcloud tool: Show/Hide Instructions

Follow the Windows, Mac OS X or Linux instructions to install gcloud on your local machine: https://cloud.google.com/sdk/

- Download and install the Google Cloud SDK by running this command in your shell or Terminal:

  ```
  curl https://sdk.cloud.google.com | bash
  ```

  Or, you can download google-cloud-sdk.zip or google-cloud-sdk.tar.gz, unpack it, and launch the ./google-cloud-sdk/install.sh script.

  Restart your shell or Terminal.

- Authenticate:

  ```
  $ gcloud auth login
  ```

- Configure the project:

  ```
  $ gcloud config set project <YOUR_PROJECT_ID>
  ```

- Install the Genomics tools:

  ```
  $ gcloud components update alpha
  ```

- Confirm the access to Genomics data works:

  ```
  $ gcloud alpha genomics readgroupsets list 10473108253681171589 --limit 10
  ID   NAME   REFERENCE_SET_ID
  CMvnhpKTfDq9e2Yy9G-Bg  HG02573  EOSt9JOVhp3jkwE
  CMvnhpKTfCEmf_d_o_JCQ  HG03894  EOSt9JOVhp3jkwE
  ...
  ```

Set up credentials for programs accessing the genomics data

If you do not have it already, get your client_secrets.json file: Show/Hide Instructions

Get your client_secrets.json file by visiting the following page:

  https://console.cloud.google.com/project/_/apiui/credential
After you select your Google Cloud project, this link will automatically take you to the Credentials tab under the API Manager.

1. Select **New credentials**
2. Select **OAuth client ID**

If prompted, select **Configure consent screen**, and follow the instructions to set a “product name” to identify your Cloud project in the consent screen. Choose “Save”.

3. Under **Application Type** choose **Other**
4. Give your client ID a name, so you can remember why it was created (suggestion: Genomics Tools)
5. Select **Create**

After successful creation, the interface should display your client ID and client secret.

To download the `client_secrets.json` file:

1. Select **OK** to close the dialog
2. Select the name of your new client id (which you specified in step 4)
3. Select **Download JSON**

Note that by convention the downloaded file is referred to as `client_secrets.json` though the file name is something much longer.

Copy `client_secrets.json` to the directory where you installed the Genomics tools.

The first time you query the API you will be authenticated using the values in the client_secrets file you downloaded. After this initial authentication, the Genomics tools save a token to use during subsequent API requests.

### Build Picard tools with GA4GH support

You will need Maven and Ant build tools installed on your machine.

1. Fetch Picard, HTSJDK and gatk-tools-java projects required for building Picard with GA4GH support.

```bash
$ git clone https://github.com/broadinstitute/picard.git
$ cd picard
$ git clone https://github.com/samtools/htsjdk
$ cd ..
$ git clone https://github.com/googlegenomics/gatk-tools-java
```

2. Build gatk-tools-java and copy the resulting JAR into Picard library folder:

```bash
$ cd gatk-tools-java
$ mvn compile package
$ mkdir ../picard/lib/gatk-tools-java
$ cp target/gatk-tools-java*minimized.jar ../picard/lib/gatk-tools-java/
```

3. Build Picard version with GA4GH support:

```bash
// Assuming you are still in gatk-tools-java directory
$ cd ..
$ ant -lib lib/ant package-commands-ga4gh
```

4. Make sure you put `client_secrets.json` file in the parent folder just above Picard. You should end up with the following directory structure:

```bash
// Assuming you are still in gatk-tools-java directory
$ cd ..
$ ant -lib lib/ant package-commands-ga4gh
```

4. Make sure you put `client_secrets.json` file in the parent folder just above Picard. You should end up with the following directory structure:
Run Picard tools with an input from the cloud

You can now run ViewSam tool that prints the contents of the supplied INPUT

```
$ cd picard
$ java \
  -jar dist/picard.jar ViewSam \
  INPUT=https://www.googleapis.com/genomics/v1beta2/readgroupsets/ \
  ->CK256frpGBD441WHwLP22R4/ \
  GA4GH_CLIENT_SECRETS=../client_secrets.json
```

This command uses an older, slower REST based API. To run using GRPC API implementation (which is much faster) use the following command that utilizes ALPN jars that come with gatk-tools-java and enables GRPC support:

```
java \
-Xbootclasspath/p:../gatk-tools-java/lib/alpn-boot-8.1.3.v20150130.jar \
-Dga4gh.using_grpc=true \
-jar dist/picard.jar ViewSam \
INPUT=https://www.googleapis.com/genomics/v1beta2/readgroupsets/ \
->CK256frpGBD441WHwLP22R4/ \
GA4GH_CLIENT_SECRETS=../client_secrets.json
```

For Java 7 (as opposed to 8) use alpn-boot-7.1.3.v20150130.jar.

We use a test readset here from genomics-test-data project.

Specifying a genomics region to use from the readset

The INPUT urls are of the form `https://<GA4GH provider>/readgroupsets/<readgroupset id>[/sequence][/start-end]`.

For example:

```
java -jar dist/picard.jar ViewSam \
INPUT=https://www.googleapis.com/genomics/v1beta2/readgroupsets/ \
/CMvnhpKTFhD3he72j4KZuyyc/chr17/41196311-41207499 \
GA4GH_CLIENT_SECRETS=../client_secrets.json
```

Timing the reading speed from the cloud

You can run gatk-tools-java/src/main/scripts/example.sh with and without “grpc” command line parameter to see the difference in reading speed. The timing statistics are dumped to the terminal. We benchmarked x11 speed improvements with GRPC compared to REST, giving ~12,000 reads/second.

The tests were done on Platinum Genomes NA12877_S1.bam dataset, please see the detailed writeup of the test procedure and results if you want to repeat the test.

We therefore recommend running GRPC variants of command line.
Other Picard tools you can run

You can run MarkDuplicates or MarkDuplicatesWithMateCigar tools like this:

```
java \
-Xbootclasspath/p:../../gatk-tools-java/lib/alpn-boot-8.1.3.v20150130.jar \
-Dga4gh.using_grpc=true \
-jar dist/picard.jar MarkDuplicates \
INPUT=https://www.googleapis.com/genomics/v1beta2/readgroupsets/CK256frpGBD44IWHwLP22R4/ \
OUTPUT=output.bam \
METRICS_FILE=output.metrics \
GA4GH_CLIENT_SECRETS=../client_secrets.json
```

Figuring out a url for your dataset

In the examples above we have been using urls of the form https://www.googleapis.com/genomics/v1beta2/readgroupsets/XXX where XXX is the id of the readset.

How do you find an ID of the readset from the Discover Published Data set or from your own project?

We will do it step by step using the command line API client.

- Lets say we want to use Platinum Genomes NA12877_S1.bam readgroupset from 1,000 Genomes project.
- The documentation page states that the dataset id for this set of files is 10473108253681171158.
- To list readgroupsets under this dataset:

  ```
  $ gcloud alpha genomics readgroupsets list 10473108253681171589 --limit 10
  ID          NAME                     REFERENCE_SET_ID
  CMvnhpKTFlTqJ2y9G-Bg  HG02573  EOSt9JOVhp3jkwE
  CMvnhpKTFlTMd_0_JCQ  HG03894  EOSt9JOVhp3jkwE
  ...
  ```

- Note the NAME column - it will correspond to the file name. The ID column is the ID of the readgroupset we are looking for.

Now lets suppose we are not looking for one of the readgroupsets form the genomics public data but instead want to use one from our own project. In this case we need to figure out the dataset id for our files first, before we can use “readgroupsets list” command to list the individual readgroupsets.

- Lets say we want to figure out which dataset ids are present under genomics test data project.
- First we need to set the project id for subsequent commands to be our project using

  ```
  $ gcloud config set project genomics-test-data
  ```

- Now we can issue this command:

  ```
  $ gcloud alpha genomics datasets list --limit 10
  ```

- The output will list dataset(s) present in the project together with their ids and we can then use the “readgroupsets list” command to get the id of the readgroupset under one of the datasets.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
**Browse Reads with Bioconductor**

Bioconductor provides a convenient way to browse regions of the genome.

```r
require(ggbio)
require(GoogleGenomics)
GoogleGenomics::authenticate("/PATH/TO/YOUR/client_secrets.json")
galignments <- getReads(readGroupSetId="CMvnhpKTfhDnk4_9zcKO3_YB", chromosome="17", start=41218200, end=41218500, converter=readsToGAlignments)
strand_plot <- autoplot(galignments, aes(color=strand, fill=strand))
coverage_plot <- ggplot(as(galignments, "GRanges")) + stat_coverage(color="gray40", fill="skyblue")
tracks(strand_plot, coverage_plot, xlab="chr17")
```

A more extensive example of read browsing with Bioconductor is documented towards the end of codelab Data Analysis using Google Genomics.

To make use of this upon your own data:

1. First, load your data into Google Genomics. See `/use_cases/load_data/index` for more detail as to how to do this.
2. If you do not have them already, install the necessary Bioconductor packages. See Using Bioconductor for more detail as to how to do this. Alternatively, you can Run Bioconductor on Compute Engine.
3. Update the parameters to the `getReads` call the example above to match that of your data and desired genomic region to view.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

**Beacon**

A beacon is a simple web service that answers questions of the form, “Do you have any genomes with an ‘A’ at position 100,735 on chromosome 3?” (or similar data). It responds simply with either “Yes” or “No.” This open web service is designed both to be technically simple (so it is easy to implement) and to mitigate risks associated with genomic data sharing.

We call these applications “Beacons” because, like the SETI project, many dedicated people have been scanning the universe of human research for signs of willing participants in far-reaching data sharing efforts, but despite many assurances of interest, it has remained a dark and quiet place. Once your “Beacon” is lit, you can start to take the next steps to add functionality to it, and finding the other groups who may help by following their Beacons.

---

1 http://ga4gh.org/#beacon
There is an AppEngine implementation of the Beacon API from the Global Alliance for Genomics and Health written in Go. Here is an example query that is running against a private copy (for demonstration purposes) of the *Illumina Platinum Genomes* variants:

http://goapp-beacon.appspot.com/?chromosome=chr17&coordinate=41196407&allele=A

To turn on a beacon for your own data:

1. First, load your data into Google Genomics. See Load Genomic Variants for more detail as to how to do this.
2. Then follow the instructions on https://github.com/googlegenomics/beacon-go to deploy the AppEngine implementation of Beacon.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

---

**GABrowse**

The properly rendered version of this document can be found at Read The Docs.  
If you are reading this on github, you should instead click here.

Try it now: https://gabrowse.appspot.com/

GABrowse is a sample application designed to demonstrate the capabilities of the Global Alliance for Genomics and Health API GA4GH v0.5.1. Currently, you can view data from Google and Ensembl.

- Use the button on the left to select a Read group set or Call set.
- Once loaded, choose a chromosome and zoom or drag the main graph to explore Read data.
- Individual bases will appear once you zoom in far enough.

To make use of this upon your own data:

1. First, load your data into Google Genomics. See /use_cases/load_data/index for more detail as to how to do this.
2. Navigate to the auth-enabled endpoint http://gabrowse-with-auth.appspot.com/ and go through the oauth flow.
3. View some data, for example http://gabrowse-with-auth.appspot.com/#=&readsetId=CMvnhpKTfHcJyLrAurGOnrAB&backend=GOOGLE&callsetId=1047310825368117589-538&cBackend=GOOGLE&location=5%3A90839366
4. Then modify the ReadGroupSetId and/or CallsetId in the URL to those of your data.

The code for this sample application is on GitHub https://github.com/googlegenomics/api-client-python

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
The R client

The GoogleGenomics Bioconductor package provides R methods to search for and retrieve Reads and Variants stored in the Google Genomics API.

Additionally it provides converters to Bioconductor datatypes such as:

- GAlignments
- GRanges
- VRanges

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Python

Data stored in Google Genomics is accessible via the Google Genomics API. This means that any programming language that can make network requests over HTTPS can be used to access it.

We have examples in github, which can help get you started. The code within each language-specific folder demonstrates the same things:

- Getting the read bases for NA12872 at a specific position
- Getting the variant overlapping that same position, and outputting the genotype

If this is what you are looking for, then take a look at the Python getting started example.

Java

Data stored in Google Genomics is accessible via the Google Genomics API. This means that any programming language that can make network requests over HTTPS can be used to access it.

We have examples in github, which can help get you started. The code within each language-specific folder demonstrates the same things:

- Getting the read bases for NA12872 at a specific position
- Getting the variant overlapping that same position, and outputting the genotype
If this is what you are looking for, then take a look at the Java getting started example.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

---

Go

Data stored in Google Genomics is accessible via the Google Genomics API. This means that any programming language that can make network requests over HTTPS can be used to access it.

We have examples in github, which can help get you started. The code within each language-specific folder demonstrates the same things:

- Getting the read bases for NA12872 at a specific position
- Getting the variant overlapping that same position, and outputting the genotype

If this is what you are looking for, then take a look at the Go getting started example.

---

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
CHAPTER 4

Analyze Data in Google Genomics

Analyze Reads

Here are some analyses that operate on cloud-resident genomic reads.

Count Reads

This simple pipeline counts reads and can be run either against a BAM file in Google Cloud Storage or against data accessed via the Google Genomics Reads API. It demonstrates the decoupling of reads data processing from ways of getting the read data and shows how to use common classes for getting reads from BAM or API data sources.

The pipeline produces a small text file with the number of reads counted.

The pipeline is implemented on Google Cloud Dataflow.

Contents

- Count Reads
  - Setup Dataflow
  - Run the pipeline
  - Additional details
Setup Dataflow

To launch the job from your local machine: Show/Hide Instructions

Most users launch Dataflow jobs from their local machine. This is unrelated to where the job itself actually runs (which is controlled by the --runner parameter). Either way, Java 8 is needed to run the Jar that kicks off the job.

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart including installing gcloud and running gcloud init.

To launch the job from Google Cloud Shell: Show/Hide Instructions

If you do not have Java on your local machine, the following setup instructions will allow you to launch Dataflow jobs using the Google Cloud Shell:

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart.
3. Use the Cloud Console to activate the Google Cloud Shell.
4. Run the following commands in the Cloud Shell to install Java 8.

```
sudo apt-get update
sudo apt-get install --assume-yes openjdk-8-jdk maven
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

Note: Depending on the pipeline, Cloud Shell may not have sufficient memory to run the pipeline locally (e.g., without the --runner command line flag). If you get error java.lang.OutOfMemoryError: Java heap space, follow the instructions to run the pipeline using Compute Engine Dataflow workers instead of locally (e.g. use --runner=DataflowPipelineRunner).

If you want to run a small pipeline on your machine before running it in parallel on Compute Engine, you will need ALPN since many of these pipelines require it. When running locally, this must be provided on the boot classpath but when running on Compute Engine Dataflow workers this is already configured for you. You can download it from here. For example:

```
```

Download the latest GoogleGenomics dataflow runnable jar from the Maven Central Repository. For example:

```
wget -O google-genomics-dataflow-runnable.jar \\n  https://search.maven.org/remotecontent?filepath=com/google/cloud/genomics/google-\\n  genomics-dataflow/v1-0.1/google-genomics-dataflow-v1-0.1-runnable.jar
```

Run the pipeline

The following command will count reads from a BAM in Google Cloud Storage, specifically those in the BRCA1 region for sample NA12877 within the Illumina Platinum Genomes dataset:
The following command will count those same reads but from the Google Genomics Reads API:

```
java -Xbootclasspath/p:alpn-boot.jar \
  -cp google-genomics-dataflow-runnable.jar \
  com.google.cloud.genomics.dataflow.pipelines.CountReads \
  --references=chr17:41196311:41277499 \
  --readGroupSetId=CMvnhpKTFhD3he72j4KZuyC \
  --output=gs://YOUR-BUCKET/dataflow-output/NA12877-API-reads.tsv
```

You can check your results by ensuring that both of these examples return the answer 45,081 in their output files.

The above command line runs the pipeline locally over a small portion of the genome, only taking a few minutes. If modified to run over a larger portion of the genome or the entire genome, it may take a few hours depending upon how many virtual machines are configured to run concurrently via `--numWorkers`. Add the following additional command line parameters to run the pipeline on Google Cloud instead of locally:

```
--runner=DataflowPipelineRunner \ 
--project=YOUR-GOOGLE-CLOUD-PLATFORM-PROJECT-ID \ 
--stagingLocation=gs://YOUR-BUCKET/dataflow-staging \ 
--numWorkers=#
```

Use a comma-separated list to run over multiple disjoint regions. For example to run over BRCA1 and BRCA2:

```
```

To run this pipeline over the entire genome, use `--allReferences` instead of:

```
--references=chr17:41196311:41277499.
```

To run the pipeline on a different read group set:

- Change the `--readGroupSetId` parameter.
- Update the `--references` as appropriate (e.g., add/remove the ‘chr’ prefix on reference names).

To run the pipeline over a different BAM file:

- Change `--BAMFilePath` parameter. Set `--shardBAMReading=false` if no BAM index file is available.
- Update the `--references` as appropriate (e.g., add/remove the ‘chr’ prefix on reference names).

**Additional details**

If the Application Default Credentials are not sufficient, use `--client-secrets PATH/TO/YOUR/client_secrets.json`. If you do not already have this file, see the authentication instructions to obtain it.

Use `--help` to get more information about the command line options. Change the pipeline class name below to match the one you would like to run.

```
java -cp google-genomics-dataflow-runnable.jar \
  com.google.cloud.genomics.dataflow.pipelines.VariantSimilarity --help
```

See the source code for implementation details: [https://github.com/googlegenomics/dataflow-java](https://github.com/googlegenomics/dataflow-java)
This pipeline calculates mean read depth coverage for a given dataset or set of read group sets and writes the results to Annotations in a new AnnotationSet using the Genomics API.

For each “bucket” in the given input references, this computes the average coverage (rounded to six decimal places) across the bucket that 10%, 20%, 30%, etc. of the input read group sets have for each mapping quality of the reads (<10:Low(L), 10-29:Medium(M), >=30:High(H)) as well as these same percentiles of read group sets for all reads regardless of mapping quality (Mapping quality All(A)).

There is also the option to change the number of quantiles accordingly (numQuantiles = 5 would give you the minimum read group set mean coverage for each and across all mapping qualities, the 25th, 50th, and 75th percentiles, and the maximum of these values).

The pipeline is implemented on Google Cloud Dataflow.

**Setup Dataflow**

To launch the job from your local machine: Show/Hide Instructions

Most users launch Dataflow jobs from their local machine. This is unrelated to where the job itself actually runs (which is controlled by the --runner parameter). Either way, Java 8 is needed to run the Jar that kicks off the job.

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart including installing gcloud and running gcloud init.

To launch the job from Google Cloud Shell: Show/Hide Instructions

If you do not have Java on your local machine, the following setup instructions will allow you to launch Dataflow jobs using the Google Cloud Shell:

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart.
3. Use the Cloud Console to activate the Google Cloud Shell.
4. Run the following commands in the Cloud Shell to install Java 8.
sudo apt-get update
sudo apt-get install --assume-yes openjdk-8-jdk maven
sudo update-alternatives --config java
sudo update-alternatives --config javac

Note: Depending on the pipeline, Cloud Shell may not have sufficient memory to run the pipeline locally (e.g., without the --runner command line flag). If you get error java.lang.OutOfMemoryError: Java heap space, follow the instructions to run the pipeline using Compute Engine Dataflow workers instead of locally (e.g. use --runner=DataflowPipelineRunner).

If you want to run a small pipeline on your machine before running it in parallel on Compute Engine, you will need ALPN since many of these pipelines require it. When running locally, this must be provided on the boot classpath but when running on Compute Engine Dataflow workers this is already configured for you. You can download it from here. For example:

```
```

Download the latest GoogleGenomics dataflow runnable jar from the Maven Central Repository. For example:

```
wget -O google-genomics-dataflow-runnable.jar https://search.maven.org/remotecontent?filepath=com/google/cloud/genomics/google-genomics-dataflow/v1-0.1/google-genomics-dataflow-v1-0.1-runnable.jar
```

Create Output Dataset

In order to run this pipeline, you must have a Google Genomics dataset to which the pipeline can output its AnnotationSet and Annotations.

- If you already have a dataset in which you have write access, you may use it. Click here to see your datasets: https://console.cloud.google.com/project/_/genomics/datasets
- If not, you can click on the following link to use the Cloud Platform Console to create one: https://console.cloud.google.com/project/_/genomics/datasets/create.

In either case, the ID of the dataset is the output dataset id you should use when running the pipeline.

Run the pipeline

The following command will calculate the mean coverage as described above for a given genomic region, using a bucket width of 1024 (in this case one bucket output) on the Illumina Platinum Genomes dataset:

```
java -Xbootclasspath/p:alpn-boot.jar -cp google-genomics-dataflow-runnable.jar com.google.cloud.genomics.dataflow.pipelines.CalculateCoverage --references=chr1:552960:553984 --bucketWidth=1024 --inputDatasetId=3049512673186936334 --outputDatasetId=YOUR-OUTPUT-DATASET-ID
```

This can take several minutes to run. You can check your results by using the Genomics API Explorer:

1. First go to the AnnotationSets search request page to determine what your newly created AnnotationSetId is.
1. Put your output dataset id in the datasetIds field.

2. Press the Authorize and Execute button.

2. Then go to the Annotations search request page to be able to see your newly created Annotation.

1. Put the AnnotationSetId you just found in the annotationSetIds field.

2. Select info and position in the fields editor.

3. Press the Authorize and Execute button.

3. Your Annotation should look like this:

```json
{
"annotations": [
{
"position": {
"referenceId": "CNfS6aHAoved2AEQy9ao_KOKwa43",
"referenceName": "chr1",
"start": "552960",
"end": "553984"
},
"info": {
"A": [
"26.623047",
"28.424805",
"35.042969",
"35.083984",
"36.039063",
"39.678711",
"46.819336",
"52.219727",
"52.681641",
"56.575195",
"62.339844"
],
"H": [
"0.196289",
"0.196289",
"0.197266",
"0.393555",
"0.59082",
"0.59082",
"0.788086",
"0.956055",
"1.27832",
"1.345703",
"1.772461"
],
"L": [
"16.304688",
"17.844727",
"21.004883",
"23.180664",
"24.850586",
"24.894531",
"26.427734",
"29.884766",
"29.933594",
"32.101563"
]
}
}
```
The following command will also calculate the mean coverage in the same manner as the previous command, but will use a select number of read group sets from the *Illumina Platinum Genomes* instead of the entire dataset, namely those for NA12883, NA12884, and NA12885. To do this, we must change the number of quantiles we are computing, as we now have fewer read group sets than the default requirement of 11:

```java
java -Xbootclasspath/p:alpn-boot.jar \\
   -cp google-genomics-dataflow runnable.jar \\
   com.google.cloud.genomics.dataflow.pipelines.CalculateCoverage \\
   --references=chr1:552960:553984 \\
   --bucketWidth=1024 \\
   --numQuantiles=3 \\
   --readGroupSetIds=CMvnhpKTFhCAv6TKo6Dq1lgg,CMvnhpKTFhDw8e3V6aCB-Q8,CMvnhpKTFhDo08GNkfe-jxo \\
   --outputDatasetId=YOUR_OUTPUT_DATASET_ID
```

This command should run a bit faster than the above command. You can check your results the same way as described above, except now your Annotation should look like this:

```json
{
   "annotations": [
       {
       "position": {
         "referenceId": "CNfS6aHAoved2AEQy9ao_KOKwa43",
         "referenceName": "chr1",
         "start": "552960",
         "end": "553984"
       },
       "info": {
         "A": [
           "35.042969",
           "51.039063",
           "56.575195"
         ],
         "H": [
           "0.393555",
           "0.956055",
           "1.345703"
         ]
       }
     }
   ]
}
```

4.1. Analyze Reads
The above command line runs the pipeline locally over a small portion of the genome, only taking a few minutes. If modified to run over a larger portion of the genome or the entire genome, it may take a few hours depending upon how many virtual machines are configured to run concurrently via `--numWorkers`. Add the following additional command line parameters to run the pipeline on Google Cloud instead of locally:

```java
--runner=DataflowPipelineRunner \
--project=YOUR-GOOGLE-CLOUD-PLATFORM-PROJECT-ID \
--stagingLocation=gs://YOUR-BUCKET/dataflow-staging \
--numWorkers=
```

Use a comma-separated list to run over multiple disjoint regions. For example to run over BRCA1 and BRCA2:

```
```

To run this pipeline over the entire genome, use `--allReferences` instead of `--references=chr17:41196311:41277499`.

To run the pipeline on a different dataset, change the `--inputDatasetId` parameter.

To run the pipeline on a different group of read group sets, change the `--readGroupSetIds` parameter.

To run the pipeline with a different bucket width, change the `--bucketWidth` parameter.

To run the pipeline with a different number of output quantiles, change the `--numQuantiles` parameter.

**Additional details**

If the Application Default Credentials are not sufficient, use `--client-secrets PATH/TO/YOUR/client_secrets.json`. If you do not already have this file, see the authentication instructions to obtain it.

Use `--help` to get more information about the command line options. Change the pipeline class name below to match the one you would like to run.

```java
java -cp google-genomics-dataflow runnable.jar \n    com.google.cloud.genomics.dataflow.pipelines.VariantSimilarity --help
```

See the source code for implementation details: https://github.com/googlegenomics/dataflow-java

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
The purpose of this code lab is to help you:

- learn how to use the Google BigQuery query tool
- learn valuable BigQuery SQL syntax
- become familiar with the variants table created by a Google Genomics variant export

BigQuery can use thousands of machines in parallel to process your queries. This means you can use it to interact with genomic data in an ad-hoc fashion: Queries that on traditional systems take hours to run (as batch jobs) can instead be processed in seconds with BigQuery.

This code lab focuses on genomic variant data that has been exported from Google Genomics to BigQuery. The dataset used is from the public Illumina Platinum Genomes project data (6 samples). You may run the same queries against other datasets exported from Google Genomics, including:

- the 1000 Genomes project data
- your own data which you can load into Google BigQuery

All output below is for queries against the Platinum Genomes.

Here are some of the questions you’ll answer in this code lab about the variant data:

- How many records are in the variants table
- How many variant calls are in the variants table
- How many variants are called for each sample
- How many samples are in the variants table
- How many variants are there per chromosome
- How many high-quality variants per-sample

Here are some of the technical skills you will learn:

- How to get an overview of the data in your table
- How are non-variant segments represented in the variants table
- How are variant calls represented in the variants table
- How are variant call quality filters represented in the variants table
How to handle hierarchical fields in variants data
• How to count distinct records
• How to group records
• How to write user-defined functions

BigQuery Web Interface

So long as you have created a Google Cloud project, then you will immediately be able to access the BigQuery web interface. Just go to http://bigquery.cloud.google.com.

On the left-hand side of the browser window you should see your Cloud project name displayed like:

If you have multiple projects, be sure that the one you want the queries of this code lab to be billed against is selected. If it is not, then click on the down-arrow icon, select “Switch to Project” and then select the correct project.

Add the Genomics public data project

You could immediately start composing queries against any BigQuery data you have access to, including public datasets. But first let’s add a nice usability shortcut for accessing the Genomics public data.

On the left-hand side of the browser window:

1. Click on the down-arrow icon that is next to your project name
2. Select “Switch to Project”
3. Select “Display Project”
4. Enter “genomics-public-data” in the Add Project dialog
5. Click “OK”

On the left-hand side, you should now see a list of public genomics datasets:
Table Overview

Before issuing any queries against the data, let’s take a look at some meta information about the variants table and get familiar with it.

Table schema

On the left-hand side of the browser window, you should see a list of BigQuery datasets. Opening “Google Genomics Public Data (genomics-public-data)” you should see platinum_genomes. Opening platinum_genomes you should see the variants table.

Selecting the variants table from the drop-down, you should now see the table schema in the right-hand pane:
The key fields of the variants table that will be frequently referenced in this code lab are:

- **reference_name** The reference on which this variant occurs (such as “chr20” or “X”).
- **start** The position at which this variant occurs (0-based). This corresponds to the first base of the string of reference bases.
- **end** The end position (0-based) of this variant. This corresponds to the first base after the last base in the reference allele. So, the length of the reference allele is \(\text{(end - start)}\).
- **reference_bases** The reference bases for this variant.
- **alternate_bases** The bases that appear instead of the reference bases.

and

- **call** The variant calls for this particular variant.

The first set of fields are what makes a variants record unique.

The call field contains a list of the calls for the variants record. The call field is an ARRAY (aka REPEATED) field and is a STRUCT (it contains NESTED fields) ARRAY and STRUCT fields are discussed further below.

The fixed members of the call field are:

- **call.call_set_id** Unique identifier generated by Google Genomics to identify a callset.
- **call.call_set_name** Identifier supplied on input to Google Genomics for a callset. This is also typically known as the sample identifier.
- **call.genotype** Array field containing the numeric genotype encodings for this call. Values:
  - -1: no call
  - 0: reference
  - 1: first alternate_bases value
Google Genomics Documentation, Release v1

• 2: second alternate_bases value
• ...
• n: nth alternate_bases value

call.genotype_likelihood  Array field containing the likelihood value for each corresponding genotype.

More details about other fields can be found at Understanding the BigQuery Variants Table Schema.

<table>
<thead>
<tr>
<th>Data note: 0-based positioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note that both the start field and end fields in the variant table are 0-based. This is consistent with the GA4GH API (which Google Genomics implements), but differs from the VCF specification in which the start column is 1-based and the end column is 0-based.</td>
</tr>
</tbody>
</table>

How was this table created?

The data in the Platinum Genomes variants table was created by:

1. Copying VCFs into Google Cloud Storage
2. Importing the VCFs into Google Genomics
3. Exporting the variants to Google BigQuery

More on the process can be found here on cloud.google.com/genomics.

More on the Google Genomics variant representation can be found here cloud.google.com/genomics.

More on the origin of the data can be found here on googlegenomics.readthedocs.org.

ARRAY and STRUCT fields

BigQuery supports fields of type ARRAY for lists of values and fields of type STRUCT for hierarchical values. These field types are useful for representing rich data without duplication.

<table>
<thead>
<tr>
<th>Legacy SQL Nomenclature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prior to supporting the SQL 2011 standard, BigQuery used its own SQL variant, now called “Legacy SQL”. In Legacy SQL ARRAY and STRUCT fields were referred to as “REPEATED” and “NESTED” fields respectively.</td>
</tr>
<tr>
<td>For more information, see the Legacy SQL Migration Guide.</td>
</tr>
</tbody>
</table>

Two of the variants fields noted above, the alternate_bases and the call field, are ARRAY fields. ARRAY fields are a feature of BigQuery that allow for embedding multiple values of the same type into the same field (similar to a list).

The alternate_bases field is a simple ARRAY field in that it allows for multiple scalar STRING values. For example:

<table>
<thead>
<tr>
<th>reference_name</th>
<th>start</th>
<th>end</th>
<th>alternate_bases</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr4</td>
<td>6214126</td>
<td>6214135</td>
<td>• A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• AACAC</td>
</tr>
<tr>
<td>chr9</td>
<td>16011409</td>
<td>16011412</td>
<td>• C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CT</td>
</tr>
</tbody>
</table>

4.2. Analyze Variants
The call field is a complex ARRAY field in that contains STRUCTs. The Platinum Genomes call field contains 13 fields of its own, such as call_set_name, genotype, and FILTER. Some fields, such as genotype and FILTER, are themselves ARRAY fields. We will see examples of working with these fields below.

Variants vs. non-variants

The Platinum Genomes data is gVCF data, meaning there are records in the variants table for non-variant segments (also known as “reference calls”). Having the reference calls in the variants table, following the gVCF conventions, “makes it straightforward to distinguish variant, reference and no-call states for any site of interest”.

Other variant sources, besides VCFs, can contain non-variant segments, including Complete Genomics masterVar files.

In a variants table exported from Google Genomics, the non-variant segments are commonly represented in one of the following ways (the representation depends on the variant caller that generated the source data):

- With a zero-length alternate_bases value, or
- With the text string <NON_REF> as an alternate_bases value, or
- With the text string <*> as an alternate_bases value

For example:

<table>
<thead>
<tr>
<th>reference_name</th>
<th>start</th>
<th>end</th>
<th>reference_bases</th>
<th>alternate_bases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
<td>1010</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

or

<table>
<thead>
<tr>
<th>reference_name</th>
<th>start</th>
<th>end</th>
<th>reference_bases</th>
<th>alternate_bases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
<td>1010</td>
<td>A</td>
<td>&lt;NON_REF&gt;</td>
</tr>
</tbody>
</table>

In this example we have a reference block of 10 bases on chromosome 1, starting at position 1000. The reference base at position 1000 is an “A” (the reference bases at the other positions of this block are not represented).

In the first case, the alternate_bases ARRAY field contains no values; it is an ARRAY of length 0. In the second case, the alternate_bases ARRAY field is length 1 containing the literal text string <NON_REF>.

See the VCF specification for further discussion of representing non-variant positions in the genome.

The Platinum Genomes data represents non-variant segments with a NULL alternate_bases value, however the queries in this code lab are designed to accommodate each of the above representations.

Table summary data

Click on the “Details” button in the right hand pane of the browser window. This will display information like:
You can immediately see the size of this table at 46.5 GB and over 261 million rows.
Click on the “Preview” button and you see a preview of a few records in the table like:

<table>
<thead>
<tr>
<th>Row</th>
<th>reference_name</th>
<th>start</th>
<th>end</th>
<th>reference_bases</th>
<th>alternate_bases</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>chrX</td>
<td>129853469</td>
<td>129853473</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>chr6</td>
<td>87612383</td>
<td>87612464</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>chr4</td>
<td>171605946</td>
<td>171605968</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>chr4</td>
<td>98935450</td>
<td>98935478</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>chr6</td>
<td>114420377</td>
<td>114420483</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>
Queries

Now that you have an overview of data in the table, we will start issuing queries and progressively add more query techniques and explanations of the `variants` table data.

We will include many documentation references when introducing new concepts, but you may find it useful to open and reference the Standard SQL Query Syntax.

How many records are in the variants table

You saw in the previous section how many variant records are in the table, but to get your feet wet with queries, let’s verify that summary data:

```sql
#standardSQL
SELECT
  COUNT(1) AS number_of_records
FROM
  `genomics-public-data.platinum_genomes.variants`
```

You should see the same result as “Number of Rows” above: 261,285,806.

How many variant calls are in the variants table

Each record in the `variants` table is a genomic position that is a variant or non-variant segment, and each record has within it an ARRAY field, which is a list of calls. Each call includes the `call_set_name` (typically the genomic “sample id”), along with values like the genotype, quality fields, read depth, and other fields typically found in a VCF or Complete Genomics masterVar file.

Let’s now get a summary of total number of calls across all samples. As noted, the `call` field is an ARRAY field, with multiple calls embedded in each `variants` record. We cannot just change what we count above to count the `call` field:

```sql
#standardSQL
SELECT
  COUNT(call) AS number_of_calls
FROM
  `genomics-public-data.platinum_genomes.variants`
```

returns the `number_of_calls` as 261,285,806. Notice that this is the same as the number of variant records. This query did NOT count the array elements, just the number of arrays.

We have a few choices then on how we properly count the calls.

One way is to count the total number of calls by querying over the `variants` records and sum the lengths of each `call` ARRAY.

```sql
#standardSQL
SELECT
  SUM(ARRAY_LENGTH(call)) AS number_of_records
FROM
  `genomics-public-data.platinum_genomes.variants`
```

Another way is to JOIN the `variants` record with the `variants.call` field. This is similar to the Legacy SQL FLATTEN technique, which effectively expands each call record to be a top level result joined with its parent `variants` record fields.
#standardSQL
SELECT COUNT(call) AS number_of_calls 
FROM `genomics-public-data.platinum_genomes.variants` v, v.call 

Note the use of the comma (,) operator, which is a short-hand notation for JOIN. Also note that the join to the call field makes an implicit UNNEST call on the call field.

**Code tip: UNNEST**
The UNNEST function provides a mechanism to query over an ARRAY field as though it is a table. UNNEST returns one record for each element of an ARRAY.

The previous query is equivalent to:

```
#standardSQL
SELECT COUNT(call) AS number_of_calls 
FROM `genomics-public-data.platinum_genomes.variants` v
JOIN UNNEST(v.call)
```

The final example for counting calls extends the previous example to demonstrate accessing one of the call fields. Each call must have a single call_set_name and so to count them:

```
#standardSQL
SELECT COUNT(call.call_set_name) AS number_of_calls 
FROM `genomics-public-data.platinum_genomes.variants` v, v.call call
```

For each of these queries, you should get a result of 309,551,691, which means that there is an average of 1.2 calls per variant record in this dataset.

**Which query is “better”?**
BigQuery pricing is based on the amount of data examined. Query performance also improves when we can reduce the amount of data examined. BigQuery provides empirical data which can be viewed in the web UI; always check the “Query complete (Ns elapsed, M B processed)” displayed. You may make use of the Query Plan Explanation to optimize your queries.

**How many variants and non-variant segments are in the table**
As discussed above, the Platinum Genomes data is gVCF data, and so the variants table contains both real variants as well as non-variant segments.

Let’s now run a query that filters out the non-variant segments:

```
#standardSQL
SELECT COUNT(1) AS number_of_real_variants 
FROM `genomics-public-data.platinum_genomes.variants` v
WHERE EXISTS (SELECT 1 
              FROM UNNEST(v.alternate_bases) AS alt)
```

4.2. Analyze Variants 79
When you issue this command, you’ll observe that the number of variants (including no-calls of variants) is 10,982,549. So the vast majority of records are reference calls, which is to be expected.

What’s the logic of this query? How did it filter out non-variant segments?

As noted above, there are (at least) three different conventions for designating a variant record as a non-variant segment. The WHERE clause here includes variant records where the alternate_bases field contains a value that is a true alternate sequence (it is NOT one of the special marker values).

In the above query, for each record in the variants table, we issue a subquery over the alternate_bases field of that variants record, returning the value 1 for each alternate_bases that is not <NON_REF> or <*>.

If the subquery returns any records, the corresponding variants record is counted.

Let’s turn the previous query around and get a count of the reference segments:

```sql
#standardSQL
SELECT
 COUNT(1) AS number_of_non_variants
FROM
 `genomics-public-data.platinum_genomes.variants` v
WHERE
  NOT EXISTS (SELECT 1
              FROM UNNEST(v.alternate_bases) AS alt
              WHERE
                alt NOT IN ("<NON_REF>", "<*>")
            )
```

This command will return a count of 250,303,257 non-variant records. This is good since:

\[
250,303,257 + 10,982,549 = 261,285,806
\]

The above WHERE clause is a literal negation of the previous query, but the double negation (NOT EXIST ... NOT IN ...) can be a little difficult to follow. A more direct form of this query is:

```sql
#standardSQL
SELECT
 COUNT(1) AS number_of_non_variants
FROM
 `genomics-public-data.platinum_genomes.variants` v
WHERE
  ARRAY_LENGTH(v.alternate_bases) = 0
 OR EXISTS (SELECT 1
             FROM UNNEST(v.alternate_bases) AS alt
             WHERE
                alt IN ("<NON_REF>", "<*>")
            )
```

This query directly counts the variant records which either:

- Have an alternate_bases array length of 0, or
- Contain an alternate_bases value of <NON_REF> or <*>.

This directly maps to the description of the non-variant segment representation noted above. But note that there is a subtle difference between this query and the previous that can produce different results depending on your data.

In many datasets, variants records will be either variants or non-variant segments; such records will either contain alternate_bases values consisting only of genomic sequences OR will contain a single <NON_REF> or <*> value.
It is however very possible for a variant caller to produce a variant record in a VCF with an ALT column value of \( T, \text{<NON\_REF>} \). Of the previous two queries, the first will *exclude* such records from the result, while the second will *include* them.

What this difference makes clear is that the notion of a particular *variants* record being a binary “variant” or “non-variant” segment is dataset-specific. One will typically want to look at more specific criteria (the actual genotype calls of specific variants) during analysis. This is discussed further below.

**How many variants does each sample have called?**

We’ve now had a quick look at the top-level records in the *variants* table. Next let’s look at the child records, namely the individual samples that have had calls made against the variants.

Each variant in the *variants* table will have zero or more `call.call_set_name` values. A given `call.call_set_name` will appear in multiple *variants* records.

To count the number of *variants* records in which each `callset` appears:

```sql
#standardSQL
SELECT
    call.call_set_name AS call_set_name,
    COUNT(call.call_set_name) AS call_count_for_call_set
FROM
    `genomics-public-data.platinum_genomes.variants` v, v.call
GROUP BY
    call_set_name
ORDER BY
    call_set_name
```

You should observe that there are 6 records returned. Each `call_set_name` corresponds to an individual who was sequenced.

<table>
<thead>
<tr>
<th>Row</th>
<th>call_set_name</th>
<th>call_count_for_call_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NA12877</td>
<td>51612762</td>
</tr>
<tr>
<td>2</td>
<td>NA12878</td>
<td>50722005</td>
</tr>
<tr>
<td>3</td>
<td>NA12889</td>
<td>49686680</td>
</tr>
<tr>
<td>4</td>
<td>NA12890</td>
<td>55528744</td>
</tr>
<tr>
<td>5</td>
<td>NA12891</td>
<td>53670694</td>
</tr>
</tbody>
</table>

But humans don’t typically have 50 million variants. Let’s filter out the reference segments and and just look at the “real” variant records:

```sql
#standardSQL
SELECT
    call.call_set_name AS call_set_name,
    COUNT(call.call_set_name) AS call_count_for_call_set
FROM
    `genomics-public-data.platinum_genomes.variants` v, v.call
WHERE
    EXISTS (SELECT 1
        FROM UNNEST(v.alternate_bases) AS alt
    )
WHERE
```

### 4.2. Analyze Variants
alt NOT IN ("<NON_REF>", "<*>")
GROUP BY
call_set_name
ORDER BY
call_set_name

Returns:

<table>
<thead>
<tr>
<th>Row</th>
<th>call_set_name</th>
<th>call_count_for_call_set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NA12877</td>
<td>5782968</td>
</tr>
<tr>
<td>2</td>
<td>NA12878</td>
<td>5809530</td>
</tr>
<tr>
<td>3</td>
<td>NA12889</td>
<td>5775376</td>
</tr>
<tr>
<td>4</td>
<td>NA12890</td>
<td>5790711</td>
</tr>
<tr>
<td>5</td>
<td>NA12891</td>
<td>5787612</td>
</tr>
</tbody>
</table>

5 million variants for a human is on the right scale, but there is one additional filter that we have missed applying to our results.

Filter “true variants” by genotype

Variants loaded into the Platinum Genomes variants table include no-calls. A no-call is represented by a genotype value of -1. These cannot be legitimately called “true variants” for individuals, so let’s filter them out. Many tools filter such calls if at least one of the genotypes is -1, and so we will do the same here.

We can be even more concrete with our variant queries by only including calls with genotypes greater than zero. If a call includes only genotypes that are no-calls (-1) or reference (0), then they are not true variants.

The following query adds the additional filtering by genotype:

```sql
#standardSQL
SELECT
call.call_set_name AS call_set_name,
COUNT(call.call_set_name) AS call_count_for_call_set
FROM
`genomics-public-data.platinum_genomes.variants` v, v.call
WHERE
  EXISTS (SELECT 1 FROM UNNEST(v.alternate_bases) AS alt WHERE
    alt NOT IN ("<NON_REF>", "<*>")
  )
 AND EXISTS (SELECT 1 FROM UNNEST(call.genotype) AS gt WHERE gt > 0)
 AND NOT EXISTS (SELECT 1 FROM UNNEST(call.genotype) AS gt WHERE gt < 0)
GROUP BY
call_set_name
ORDER BY
call_set_name
```

Returns:
Is the non-variant segment filter actually needed here?

The above query filtered out:

- non-variant segments
- calls for which all genotype values are 0 and/or -1

There is some redundancy in this filter. All call.genotype values for non-variant segments in this dataset are either 0, or -1. Thus the above query could safely be rewritten without the filter on alternate_bases.

```sql
#standardSQL
SELECT
    call.call_set_name AS call_set_name,
    COUNT(call.call_set_name) AS call_count_for_call_set
FROM
    `genomics-public-data.platinum_genomes.variants` v, v.call
WHERE
    EXISTS (SELECT 1 FROM UNNEST(call.genotype) AS gt WHERE gt > 0)
    AND NOT EXISTS (SELECT 1 FROM UNNEST(call.genotype) AS gt WHERE gt < 0)
GROUP BY
    call.call_set_name
ORDER BY
    call.call_set_name
```

The previous form of this query may be preferred as it makes the semantic intent of more clear (only query over “true variant” records).

However as queries become larger and more complicated, removing well-known redundancies can make your queries more readable and can also make them less expensive. BigQuery costs are based on the number of bytes processed. The second form of the query does not need to examine the alternate_bases column.

How many samples are in the variants table?

In the previous few queries, we observed that there are 6 distinct call_set_name values in the variants table as each query returned 6 rows. But what if we were interested in specifically returning that count?

One way to do this is to take our existing query and treat it like a table over which we can query. In this example, we take the previous queries and first collapse it down to the minimum results needed - just the list of call set names:

```sql
#standardSQL
SELECT call.call_set_name
FROM `genomics-public-data.platinum_genomes.variants` v, v.call
GROUP BY call.call_set_name
```
then we compose a query using the SQL WITH clause.

```sql
#standardSQL
WITH call_sets AS (
  SELECT call.call_set_name
  FROM `genomics-public-data.platinum_genomes.variants` v, v.call
  GROUP BY call.call_set_name)

SELECT
  COUNT(1) AS number_of_callsets
FROM
  call_sets

This composition query pattern is frequently useful and is shown here as an example.
Composition turns out to be unnecessary for this particular query. We can get the count of distinct `call_set_name` values an easier way:

```sql
#standardSQL
SELECT
  COUNT(DISTINCT call.call_set_name) AS number_of_callsets
FROM
  `genomics-public-data.platinum_genomes.variants` v, v.call
```

**How many variants are there per chromosome**

We’ve had a look at the number of variants per callset. What if we want to look at the number of variants per chromosome. Given our experience with `GROUP BY` and `COUNT` from the previous section, this should be fairly straight-forward. We just need to apply these same tools to the `reference_name` field.

It turns out that there are some wrinkles to contend with. The query that we want is:

- Return all `variants` records in which there is
  - at least one call with
    - at least one genotype greater than 0
- Group the variant records by chromosome and count each group

The first wrinkle is that we need to look into an ARRAY (genotype) within an ARRAY (call) while keeping execution context of the query at the `variants` record level. We are not interested in producing a per-call or per-genotype result. We are interested in producing a per-variant result.

We saw above how to “look into” an ARRAY record, without changing the query context, we can use the `UNNEST` function in an EXISTS subquery in our WHERE clause:

```sql
#standardSQL
SELECT
  reference_name,
  COUNT(reference_name) AS number_of_variant_records
FROM
  `genomics-public-data.platinum_genomes.variants` v
WHERE
  EXISTS (SELECT 1
    FROM UNNEST(v.call) AS call
    WHERE EXISTS (SELECT 1
      FROM UNNEST(call.genotype) AS gt
      WHERE gt > 0))
```
GROUP BY
   reference_name
ORDER BY
   reference_name

Returns:

<table>
<thead>
<tr>
<th>Row</th>
<th>reference_name</th>
<th>number_of_variant_records</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>chr1</td>
<td>813381</td>
</tr>
<tr>
<td>2</td>
<td>chr10</td>
<td>515289</td>
</tr>
<tr>
<td>3</td>
<td>chr11</td>
<td>492978</td>
</tr>
<tr>
<td>4</td>
<td>chr12</td>
<td>487604</td>
</tr>
<tr>
<td>5</td>
<td>chr13</td>
<td>365615</td>
</tr>
<tr>
<td>6</td>
<td>chr14</td>
<td>325148</td>
</tr>
</tbody>
</table>

The above encodes very explicitly our needed logic. We can make this a bit more concise by turning the EXISTS clause into a JOIN of the call field with the call.genotype field:

```sql
#standardSQL
SELECT
   reference_name,
   COUNT(reference_name) AS number_of_variant_records
FROM
   `genomics-public-data.platinum_genomes.variants` v
WHERE
   EXISTS (SELECT 1
      FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
      WHERE gt > 0)
GROUP BY
   reference_name
ORDER BY
   reference_name
```

The above is good and the results are correct, but let’s work on improving our output. Our second wrinkle arises as we’d like to sort the output in chromosome-numeric order but the field we are sorting on is a STRING and the values contain the prefix “chr”.

Let’s walk through a few steps to demonstrate some BigQuery technique.

To sort numerically, we should first trim out the “chr” from the reference_name field:

```sql
#standardSQL
SELECT
   REGEXP_REPLACE(reference_name, '^chr', '') AS chromosome,
   COUNT(reference_name) AS number_of_variant_records
FROM
   `genomics-public-data.platinum_genomes.variants` v
WHERE
   EXISTS (SELECT 1
      FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
      WHERE gt > 0)
GROUP BY
   chromosome
ORDER BY
   chromosome
```

4.2. Analyze Variants
What did we do here? First we used the `REGEXP_REPLACE` function to replace the leading “chr” string with an empty string (and gave the result a column alias of `chromosome`). Then we changed the `GROUP BY` and `ORDER BY` to use the computed `chromosome` field. But the ordering isn’t quite what we wanted:

```sql
#standardSQL
SELECT
    CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) AS chromosome,
    COUNT(reference_name) AS number_of_variant_records
FROM
    `genomics-public-data.platinum_genomes.variants` v
WHERE
    EXISTS (SELECT 1
        FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
        WHERE gt > 0)
GROUP BY
    chromosome
ORDER BY
    chromosome
```

Unfortunately this generates an error:

```
Error: Bad int64 value: X
```

Not all chromosome names are numeric, namely X, Y, and M. This makes it challenging to order as desired. Let’s approach this slightly differently and use string sorting. To get the desired order, we will prepend a “0” to chromosomes 1-9:

```sql
#standardSQL
SELECT
    CASE
        WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10
            THEN CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', ''))
        ELSE REGEXP_REPLACE(reference_name, '^chr', '')
    END AS chromosome,
    COUNT(reference_name) AS number_of_variant_records
FROM
    `genomics-public-data.platinum_genomes.variants` v
WHERE
    EXISTS (SELECT 1
        FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
        WHERE gt > 0)
GROUP BY
    chromosome
ORDER BY
    chromosome
```
GROUP BY
   chromosome
ORDER BY
   chromosome

This looks better:

<table>
<thead>
<tr>
<th>Row</th>
<th>chromosome</th>
<th>number_of_variant_records</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>813381</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>828795</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>676061</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>731912</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>607471</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What did we do? We used the highly flexible **CASE function** to prepend a “0” to all chromosomes numbered less than 10, and only removed the “chr” from the remaining **reference_name** values.

Also notice the use of the **SAFE_CAST** function. This will return NULL for X, Y, and M instead of raising an error.

As a final improvement on the output of the above query, let’s display the **reference_name** unchanged while still getting the sort ordering we want. All we need to do is move our **CASE** clause to the **ORDER BY**:

```sql
#standardSQL
SELECT
   reference_name,
   COUNT(reference_name) AS number_of_variant_records
FROM
   `genomics-public-data.platinum_genomes.variants` v
WHERE
   EXISTS (SELECT 1
      FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
      WHERE gt > 0)
GROUP BY
   reference_name
ORDER BY
   CASE
       WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10
           THEN CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', ''))
       ELSE REGEXP_REPLACE(reference_name, '^chr', '')
   END

Result:

4.2. Analyze Variants
User Defined Functions

We were able to embed some fairly interesting logic into our query with the CASE statement. But doing so made the query more verbose. As you build more complex queries, keeping the queries concise becomes more and more important to make it easier to ensure their logic is correct.

Let’s use one last bit of BigQuery technique to improve on our query: User Defined Functions. UDFs can be defined as SQL expressions or as JavaScript.

In our first example, we will simply move the CASE logic from our previous query into a function:

```
#standardSQL
CREATE TEMPORARY FUNCTION SortableChromosome(reference_name STRING) 
  RETURNS STRING AS ( 
    -- Remove the leading "chr" (if any) in the reference_name 
    -- If the chromosome is 1 - 9, prepend a "0" since 
    -- "2" sorts after "10", but "02" sorts before "10". 
    CASE 
      WHEN SAFE_CAST(REGEXP_REPLACE(reference_name, '^chr', '') AS INT64) < 10 
      THEN CONCAT('0', REGEXP_REPLACE(reference_name, '^chr', '')) 
      ELSE REGEXP_REPLACE(reference_name, '^chr', '') 
    END 
  );

SELECT 
  reference_name, 
  COUNT(reference_name) AS number_of_variant_records 
FROM 
  `genomics-public-data.platinum_genomes.variants` v 
WHERE 
  EXISTS (SELECT 1 
    FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt 
    WHERE gt > 0) 
GROUP BY 
  reference_name 
ORDER BY SortableChromosome(reference_name)
```

In the second example, we use a function defined in JavaScript:

```
#standardSQL
CREATE TEMPORARY FUNCTION SortableChromosome(reference_name STRING) 
  RETURNS STRING LANGUAGE js AS ' 
  // Remove the leading "chr" (if any) in the reference_name 
  var chr = reference_name.replace(/^chr/, ''); 
  
  // Remove the leading "chr" (if any) in the reference_name 
  return chr; 
  
SELECT 
  reference_name, 
  COUNT(reference_name) AS number_of_variant_records 
FROM 
  `genomics-public-data.platinum_genomes.variants` v 
WHERE 
  EXISTS (SELECT 1 
    FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt 
    WHERE gt > 0) 
GROUP BY 
  reference_name 
ORDER BY SortableChromosome(reference_name)
```
// If the chromosome is 1 - 9, prepend a "0" since
// "2" sorts after "10", but "02" sorts before "10".
if (chr.length == 1 && '123456789'.indexOf(chr) >= 0) {
  return '0' + chr;
}
return chr;

```

SELECT reference_name, COUNT(reference_name) AS number_of_variant_records
FROM `genomics-public-data.platinum_genomes.variants` v
WHERE EXISTS (SELECT 1
  FROM UNNEST(v.call) AS call, UNNEST(call.genotype) AS gt
  WHERE gt > 0)
GROUP BY reference_name
ORDER BY SortableChromosome(reference_name)

```

Each of these two queries returns the same as our previous query, but the logic of the query is more concise.

**How many high-quality variants per-sample**

The VCF specification describes the FILTER field which can be used to label variant calls of different qualities. Let’s take a look at the per-call FILTER values for the Platinum Genomes dataset:

```
#standardSQL
SELECT call_filter, COUNT(call_filter) AS number_of_calls
FROM `genomics-public-data.platinum_genomes.variants` v,
  v.call, UNNEST(call.FILTER) AS call_filter
GROUP BY call_filter
ORDER BY number_of_calls

```

Returns:
Calls with multiple FILTER values

The values for the `number_of_calls` seem high based on the total number of calls. Let’s sum up all of the FILTER values:

```sql
#standardSQL
SELECT COUNT(call_filter) AS number_of_filters
FROM `genomics-public-data.platinum_genomes.variants` v,
  v.call,
  call.FILTER AS call_filter
```

The returned result is 327,580,807, which is higher than the total number of calls we computed earlier (309,551,691). So what is going on here? Is our query faulty?

No. The FILTER field is an ARRAY field within each call field, so some call fields have multiple FILTER values. Let’s concatenate the FILTER field values while looking at a few variant calls.

```sql
#standardSQL
SELECT
  reference_name,
  start, 'end',
  reference_bases,
  call.call_set_name AS call_set_name,
  (SELECT STRING_AGG(call_filter) FROM UNNEST(call.FILTER) AS call_filter) AS filters,
  ARRAY_LENGTH(call.FILTER) AS filter_count
FROM `genomics-public-data.platinum_genomes.variants` v, v.call
WHERE
  ARRAY_LENGTH(call.FILTER) > 1
ORDER BY
  filter_count DESC, reference_name, start, 'end', reference_bases, call_set_name
LIMIT 10
```

Returns:
So we can see that some variant calls of low quality will fail to pass multiple filters.

**FILTERing for high quality variant records**

From the count of **FILTER** values above, we can see that the vast majority of variant calls have been marked with the **PASS** label, indicating that they are high quality calls that have passed all variant calling filters.

When analyzing variants, you will often want to filter out lower quality variants. It is expected that if the **FILTER** field contains the value **PASS**, it will contain no other values. Let’s verify that by adding one new condition to the **WHERE** clause of the previous query:

```sql
#standardSQL
SELECT
  reference_name,
  start,
  `end`,
  reference_bases,
  call.call_set_name AS call_set_name,
  (SELECT STRING_AGG(call_filter) FROM UNNEST(call.FILTER) AS call_filter) AS filters,
  ARRAY_LENGTH(call.FILTER) AS filter_count
FROM
  `genomics-public-data.platinum_genomes.variants` v, v.call
WHERE
  EXISTS (SELECT 1 FROM UNNEST(call.FILTER) AS call_filter WHERE call_filter = 'PASS')
  AND ARRAY_LENGTH(call.FILTER) > 1
ORDER BY
  filter_count DESC, reference_name, start, `end`, reference_bases, call_set_name
LIMIT 10
```

The result is:

```
Query returned zero records.
```

This query omitted any call that did not contain a **PASS** value for **FILTER**, and only returned calls for which there was more than 1 **FILTER** value.

**Count high quality calls for samples**

**All high quality calls for each sample**

The following counts all calls (variants and non-variants) for each call set omitting any call with a non-**PASS** filter.

---

4.2. Analyze Variants 91
#standardSQL
SELECT  
call.call_set_name AS call_set_name,  
COUNT(1) AS number_of_calls  
FROM  
`genomics-public-data.platinum_genomes.variants` v, v.call  
WHERE  
NOT EXISTS (SELECT 1 FROM UNNEST(call.FILTER) AS call_filter WHERE call_filter != 'PASS')  
GROUP BY  
call_set_name  
ORDER BY  
call_set_name

Returns:

<table>
<thead>
<tr>
<th>Row</th>
<th>call_set_name</th>
<th>number_of_calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NA12877</td>
<td>44883669</td>
</tr>
<tr>
<td>2</td>
<td>NA12878</td>
<td>44174985</td>
</tr>
<tr>
<td>3</td>
<td>NA12889</td>
<td>42826125</td>
</tr>
<tr>
<td>4</td>
<td>NA12890</td>
<td>48815828</td>
</tr>
<tr>
<td>5</td>
<td>NA12891</td>
<td>46700963</td>
</tr>
<tr>
<td>6</td>
<td>NA12892</td>
<td>41817518</td>
</tr>
</tbody>
</table>

All high quality true variant calls for each sample

The following counts all calls (variants and non-variants) for each call set omitting any call with a non-PASS filter and including only calls with at least one true variant (genotype > 0).

#standardSQL
SELECT  
call.call_set_name AS call_set_name,  
COUNT(1) AS number_of_calls  
FROM  
`genomics-public-data.platinum_genomes.variants` v, v.call  
WHERE  
NOT EXISTS (SELECT 1 FROM UNNEST(call.FILTER) AS call_filter WHERE call_filter != 'PASS')  
AND EXISTS (SELECT 1 FROM UNNEST(call.genotype) as gt WHERE gt > 0)  
GROUP BY  
call_set_name  
ORDER BY  
call_set_name
Where to go next

The Google Genomics team and the community have contributed many data analysis examples and tools that build on the concepts you have learned here.

To find more sample queries and methods of accessing a `variants` table in BigQuery see:

- https://github.com/googlegenomics/getting-started-bigquery
- https://github.com/googlegenomics/bigquery-examples
- https://github.com/googlegenomics/codelabs
- Tute Genomics Annotation
- Analyze Variants

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Data Analysis Codelab

The properly rendered version of this document can be found at Read The Docs.
If you are reading this on github, you should instead click here.

There are a collection of analyses upon variants documented in codelab Data Analysis using Google Genomics.

In this codelab, you will use Google Genomics, Google BigQuery, Apache Spark, and R to explore the 1,000 Genomes dataset. Specifically, you will:

- run a principal component analysis (either from scratch or using pre-computed results)
- use BigQuery to explore population variation
- zoom in to specific genome regions, including using the Genomics API to look all the way down to raw reads
- run a GWAS over the variants within BRCA1
- visualize and annotate results using various R packages, including Bioconductor
To make use of this upon your own data:

1. First, load your data into Google Genomics and export your variants to BigQuery. See \use_cases/load_data/index for more detail as to how to do this.
2. Update the BigQuery table name, variant set id, and read group set in the example to match those of your data.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

**Transition/Transversion Ratio**

| The properly rendered version of this document can be found at Read The Docs. |
| If you are reading this on github, you should instead click here. |

There are several transition/transversion ratio examples in GitHub:

- Ti/Tv by Genomic Window query and plot.
- Ti/Tv by Alternate Allele Counts query and plot.
- Ti/Tv for an entire cohort query.
- A comparison of vcfstats Ti/Tv results to results from BigQuery.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

**Hardy-Weinberg Equilibrium**

| The properly rendered version of this document can be found at Read The Docs. |
| If you are reading this on github, you should instead click here. |

There are several Hardy-Weinberg Equilibrium examples in GitHub:

- Hardy-Weinberg Equilibrium query and example.
- A comparison of vcfstats Hardy-Weinberg Equilibrium results to results from BigQuery.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

**Compute Principal Coordinate Analysis**

| The properly rendered version of this document can be found at Read The Docs. |
| If you are reading this on github, you should instead click here. |
Compute Principal Coordinate Analysis

Principal Coordinate Analysis counts the number of variants two samples have in common. These counts are then placed into an \( N \times N \) matrix where \( N \) is the number of samples in the variant set. The matrix is centered, scaled, and then the first two principal components are computed for each individual.

See the Data Analysis using Google Genomics codelab for an example that makes use of the results of this analysis run upon 1,000 Genomes.

Both Google Cloud Dataflow and Apache Spark implementations are available.

Dataflow

Setup

To launch the job from your local machine: Show/Hide Instructions

Most users launch Dataflow jobs from their local machine. This is unrelated to where the job itself actually runs (which is controlled by the --runner parameter). Either way, Java 8 is needed to run the Jar that kicks off the job.

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart including installing gcloud and running gcloud init.

To launch the job from Google Cloud Shell: Show/Hide Instructions

If you do not have Java on your local machine, the following setup instructions will allow you to launch Dataflow jobs using the Google Cloud Shell:

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart.
3. Use the Cloud Console to activate the Google Cloud Shell.
4. Run the following commands in the Cloud Shell to install Java 8.
sudo apt-get update
sudo apt-get install --assume-yes openjdk-8-jdk maven
sudo update-alternatives --config java
sudo update-alternatives --config javac

Note: Depending on the pipeline, Cloud Shell may not have sufficient memory to run the pipeline locally (e.g., without the --runner command line flag). If you get error java.lang.OutOfMemoryError: Java heap space, follow the instructions to run the pipeline using Compute Engine Dataflow workers instead of locally (e.g. use --runner=DataflowPipelineRunner).

If you want to run a small pipeline on your machine before running it in parallel on Compute Engine, you will need ALPN since many of these pipelines require it. When running locally, this must be provided on the boot classpath but when running on Compute Engine Dataflow workers this is already configured for you. You can download it from here. For example:

```
wget -O alpn-boot.jar \\
   alpn-boot-8.1.8.v20160420.jar
```

Download the latest GoogleGenomics dataflow runnable jar from the Maven Central Repository. For example:

```
wget -O google-genomics-dataflow-runnable.jar \\
   https://search.maven.org/remotecontent?filepath=com/google/cloud/genomics/google-genomics-dataflow/v1-0.1/google-genomics-dataflow-v1-0.1-runnable.jar
```

Run the pipeline

The following command will run PCA over the BRCA1 region within the Illumina Platinum Genomes variant set.

```
java -XXbootclasspath/p:alpn-boot.jar \\
   -cp google-genomics-dataflow-runnable.jar \\
   com.google.cloud.genomics.dataflow.pipelines.VariantSimilarity \\
   --variantSetId=3049512673186936334 \\
   --references=chr17:41196311:41277499 \\
   --output=gs://YOUR-BUCKET/dataflow-output/platinum-genomes-brca1-pca.tsv
```

The above command line runs the pipeline locally over a small portion of the genome, only taking a few minutes. If modified to run over a larger portion of the genome or the entire genome, it may take a few hours depending upon how many virtual machines are configured to run concurrently via --numWorkers. Add the following additional command line parameters to run the pipeline on Google Cloud instead of locally:

```
--runner=DataflowPipelineRunner \\
--project=YOUR-GOOGLE-CLOUD-PLATFORM-PROJECT-ID \\
--stagingLocation=gs://YOUR-BUCKET/dataflow-staging \\
--numWorkers=#
```

Use a comma-separated list to run over multiple disjoint regions. For example to run over BRCA1 and BRCA2 --references=chr13:32889610:32973808,chr17:41196311:41277499.

To run this pipeline over the entire genome, use --allReferences instead of --references=chr17:41196311:41277499.

To run the pipeline on a different variant set:
• Change the variant set id for the --variantSetId id parameter.
• Update the --references as appropriate (e.g., add/remove the ‘chr’ prefix on reference names).

Additional details

If the Application Default Credentials are not sufficient, use --client-secrets PATH/TO/YOUR/client_secrets.json. If you do not already have this file, see the authentication instructions to obtain it.

Use --help to get more information about the command line options. Change the pipeline class name below to match the one you would like to run.

```
java -cp google-genomics-dataflow*runnable.jar \n  com.google.cloud.genomics.dataflow.pipelines.VariantSimilarity --help
```

See the source code for implementation details: https://github.com/googlegenomics/dataflow-java

Spark

Setup

• Deploy your Spark cluster using Google Cloud Dataproc. This can be done using the Cloud Platform Console or the following gcloud command:

```
gcloud beta dataproc clusters create example-cluster --scopes cloud-platform
```

• ssh to the master.

```
gcloud compute ssh example-cluster-m
```

• Compile and build the pipeline jar. You can build locally or build on the Spark master Google Compute Engine virtual machine.

To compile and build on Compute Engine: Show/Hide Instructions

1. Install sbt.

```
echo "deb https://dl.bintray.com/sbt/debian /*" | sudo tee -a /etc/apt/ ˓→sources.list.d/sbt.list
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 642AC823
sudo apt-get install apt-transport-https
sudo apt-get update
sudo apt-get install sbt
```

2. Clone the github repository.

```
sudo apt-get install git
git clone https://github.com/googlegenomics/spark-examples.git
```

3. Compile the Jar.

```
cd spark-examples
sbt assembly
cp target/scala-2.*/googlegenomics-spark-examples-assembly-*.jar ~/  cd ~/
```

4.2. Analyze Variants
Run the job

The following command will run PCA over the BRCA1 region within the *Illumina Platinum Genomes* variant set.

```
spark-submit \
--class com.google.cloud.genomics.spark.examples.VariantsPcaDriver \
--conf spark.shuffle.spill=true \n  googlegenomics-spark-examples-assembly-1.0.jar \n  --variant-set-id 3049512673186936334 \n  --references chr17:41196311:41277499 \n  --output-path gs://YOUR-BUCKET/output/platinum-genomes-brca1-pca.tsv
```

The above command line runs the job over a small portion of the genome, only taking a couple minutes. If modified to run over a larger portion of the genome or the entire genome, it may take a few hours depending upon how many machines are in the Spark cluster.

To run this job over a large portion of the genome or the entire genome:

- Create a larger cluster: `gcloud beta dataproc clusters create cluster-2 --scopes cloud-platform --num-workers #`
- Add `--num-reduce-partitions #` to be equal to the number of cores in your cluster.
- Use a comma-separated list to run over multiple disjoint regions. For example to run over BRCA1 and BRCA2
- Use `--all-references` instead of `--references chr17:41196311:41277499` to run over the entire genome.

To run the job on a different variant set:

- Change the variant set id for the `--variant-set-id` parameter.
- Update the `--references` as appropriate (e.g., add/remove the ‘chr’ prefix on reference names).

Additional details

If the Application Default Credentials are not sufficient, use `--client-secrets=PATH/TO/YOUR/client_secrets.json`. If you do not already have this file, see the authentication instructions to obtain it.

Use `--help` to get more information about the job-specific command line options. Change the job class name below to match the one you would like to run.

```
spark-submit --class com.google.cloud.genomics.spark.examples.VariantsPcaDriver \
  googlegenomics-spark-examples-assembly-1.0.jar --help
```

See the source code for implementation details: [https://github.com/googlegenomics/spark-examples](https://github.com/googlegenomics/spark-examples)

Gather the results into a single file

```
gsutil cat gs://YOUR-BUCKET/output/platinum-genomes-brca1-pca.tsv* \
  | sort > platinum-genomes-brca1-pca.tsv
```

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see [https://cloud.google.com/genomics/support](https://cloud.google.com/genomics/support).
Compute Principal Coordinate Analysis on the Intersection of Two Variant Sets

Principal Coordinate Analysis counts the number of variants two samples have in common. These counts are then placed into an \((N+M) \times (N+M)\) matrix where \(N\) is the number of samples in the control variant set (e.g., 1,000 Genomes) and \(M\) is the number of samples in the case variant set. The matrix is centered, scaled, and then the first two principal components are computed for each individual.

In the two-way version, the variants shared between two variant sets are used to compute PCA among the individuals in both variant sets. This can be useful, for example, as an ethnicity check when comparing a variant set to 1,000 Genomes. See codelab Quality Control using Google Genomics for an example of this.

An Apache Spark implementation is available.

Setup

- Deploy your Spark cluster using Google Cloud Dataproc. This can be done using the Cloud Platform Console or the following `gcloud` command:

  ```bash
  gcloud beta dataproc clusters create example-cluster --scopes cloud-platform
  ```

- ssh to the master.

  ```bash
  gcloud compute ssh example-cluster-m
  ```

- Compile and build the pipeline jar. You can build locally or build on the Spark master Google Compute Engine virtual machine.

  To compile and build on Compute Engine: Show/Hide Instructions

  1. Install sbt.

     ```bash
     echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list
dsudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 642AC823
sudo apt-get install apt-transport-https
sudo apt-get install apt-get update
sudo apt-get install sbt
     ```

  2. Clone the github repository.

     ```bash
     sudo apt-get install git
git clone https://github.com/googlegenomics/spark-examples.git
     ```
3. Compile the Jar.

```bash
cd spark-examples
sbt assembly
cp target/scala-2.*/googlegenomics-spark-examples-assembly-*.jar ~/
cd ~/
```

Run the job

The following command will run a two-way PCA over the BRCA1 region within the *Illumina Platinum Genomes* variant set and the *1,000 Genomes* phase 1 variants.

```bash
spark-submit
  --class com.google.cloud.genomics.spark.examples.VariantsPcaDriver
  --conf spark.shuffle.spill=true
  googlegenomics-spark-examples-assembly-1.0.jar
  --variant-set-id 10473108253681171589 3049512673186936334
  --references 17:41196311:41277499 chr17:41196311:41277499
  --output-path gs://YOUR-BUCKET/output/two-way-brca1-pca.tsv
```

The above command line runs the job over a small portion of the genome, only taking a couple minutes. If modified to run over a larger portion of the genome or the entire genome, it may take a few hours depending upon how many machines are in the Spark cluster.

To run this job over the entire genome:

- Create a larger cluster: `gcloud beta dataproc clusters create cluster-2 --scopes cloud-platform --num-workers #`
- Add `--num-reduce-partitions #` to be somewhere between 10-20 this will be the level of parallelism when computing the reference call similarity, keep it bounded to a small number, otherwise the shuffle will need to move a full similarity matrix for each reducer.
- Use `--all-references` instead of `--references 17:41196311:41277499 chr17:41196311:41277499` to run over the entire genome.

To run the job on a different variant set:

- Change the second variant set id for the `--variant-set-id` id parameter.
- Update the second value in `--references` as appropriate (e.g., add/remove the ‘chr’ prefix on reference names).
- Increase the memory used when running with a large $M$ via `--conf spark.driver.maxResultSize=10G` and `--driver-memory 20G`

Additional details

If the Application Default Credentials are not sufficient, use `--client-secrets=PATH/TO/YOUR/client_secrets.json`. If you do not already have this file, see the authentication instructions to obtain it.

Use `--help` to get more information about the job-specific command line options. Change the job class name below to match the one you would like to run.

```bash
spark-submit --class com.google.cloud.genomics.spark.examples.VariantsPcaDriver
  googlegenomics-spark-examples-assembly-1.0.jar --help
```

See the source code for implementation details: https://github.com/googlegenomics/spark-examples
Compute Identity By State

Identity-by-State is a simple similarity measure that describes the alleles shared by two individuals as a single number. See the Quality Control using Google Genomics codelab for an example that makes use of the results of this analysis run upon Illumina Platinum Genomes.

A Google Cloud Dataflow implementation is available.

Setup Dataflow

To launch the job from your local machine: Show/Hide Instructions

Most users launch Dataflow jobs from their local machine. This is unrelated to where the job itself actually runs (which is controlled by the --runner parameter). Either way, Java 8 is needed to run the Jar that kicks off the job.

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart including installing gcloud and running gcloud init.

To launch the job from Google Cloud Shell: Show/Hide Instructions

If you do not have Java on your local machine, the following setup instructions will allow you to launch Dataflow jobs using the Google Cloud Shell:

Gather the results into a single file

```
gsutil cat gs://YOUR-BUCKET/output/two-way-brca1-pca.tsv* \  | sort > two-way-brca1-pca.tsv
```

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Gather the results into a single file

```
gsutil cat gs://YOUR-BUCKET/output/two-way-brca1-pca.tsv* \  | sort > two-way-brca1-pca.tsv
```

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Compute Identity By State

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

Contents

- Compute Identity By State
  - Setup Dataflow
  - Run the pipeline
  - Gather the results into a single file
  - Additional details

Identity-by-State is a simple similarity measure that describes the alleles shared by two individuals as a single number. See the Quality Control using Google Genomics codelab for an example that makes use of the results of this analysis run upon Illumina Platinum Genomes.

A Google Cloud Dataflow implementation is available.

Setup Dataflow

To launch the job from your local machine: Show/Hide Instructions

Most users launch Dataflow jobs from their local machine. This is unrelated to where the job itself actually runs (which is controlled by the --runner parameter). Either way, Java 8 is needed to run the Jar that kicks off the job.

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart including installing gcloud and running gcloud init.

To launch the job from Google Cloud Shell: Show/Hide Instructions

If you do not have Java on your local machine, the following setup instructions will allow you to launch Dataflow jobs using the Google Cloud Shell:
1. If you have not already done so, follow the Genomics Quickstart.

2. If you have not already done so, follow the Dataflow Quickstart.

3. Use the Cloud Console to activate the Google Cloud Shell.

4. Run the following commands in the Cloud Shell to install Java 8.

   ```bash
   sudo apt-get update
   sudo apt-get install --assume-yes openjdk-8-jdk maven
   sudo update-alternatives --config java
   sudo update-alternatives --config javac
   ```

Note: Depending on the pipeline, Cloud Shell may not not have sufficient memory to run the pipeline locally (e.g., without the `--runner` command line flag). If you get error `java.lang.OutOfMemoryError: Java heap space`, follow the instructions to run the pipeline using Compute Engine Dataflow workers instead of locally (e.g. use `--runner=DataflowPipelineRunner`).

If you want to run a small pipeline on your machine before running it in parallel on Compute Engine, you will need ALPN since many of these pipelines require it. When running locally, this must be provided on the boot classpath but when running on Compute Engine Dataflow workers this is already configured for you. You can download it from here. For example:

   ```bash
   wget -O alpn-boot.jar \
   ```

Download the latest GoogleGenomics dataflow runnable jar from the Maven Central Repository. For example:

   ```bash
   wget -O google-genomics-dataflow-runnable.jar \
   https://search.maven.org/remotecontent?filepath=com/google/cloud/genomics/google-\n   → genomics-dataflow/vl-0.1/google-genomics-dataflow-v1-0.1-runnable.jar
   ```

Run the pipeline

The following command will run Identity-by-State over the BRCA1 region within the Illumina Platinum Genomes variant set.

   ```bash
   java -Xbootclasspath/p:alpn-boot.jar \n   -cp google-genomics-dataflow-runnable.jar \n   com.google.cloud.genomics.dataflow.pipelines.IdentityByState \n   --variantSetId=3049512673186936334 \n   --references=chr17:41196311:41277499 \n   --hasNonVariantSegments \n   --output=gs://YOUR-BUCKET/dataflow-output/platinum-genomes-brca1-ibs.tsv
   ```

Note that there are several IBS calculators from which to choose. Use the `--callSimilarityCalculatorFactory` to switch between them.

Also notice use of the `--hasNonVariantSegments` parameter when running this pipeline on the Illumina Platinum Genomes variant set.

- For data with non-variant segments (such as Complete Genomics data or data in Genome VCF (gVCF) format), specify this flag so that the pipeline correctly takes into account non-variant segment records that overlap variants within the variant set.
- The source Illumina Platinum Genomes data imported into Google Genomics was in gVCF format.
The above command line runs the pipeline locally over a small portion of the genome, only taking a few minutes. If modified to run over a larger portion of the genome or the entire genome, it may take a few hours depending upon how many virtual machines are configured to run concurrently via --numWorkers. Add the following additional command line parameters to run the pipeline on Google Cloud instead of locally:

```
--runner=DataflowPipelineRunner \ 
--project=YOUR-GOOGLE-CLOUD-PLATFORM-PROJECT-ID \ 
--stagingLocation=gs://YOUR-BUCKET/dataflow-staging \ 
--numWorkers=#
```

Use a comma-separated list to run over multiple disjoint regions. For example to run over BRCA1 and BRCA2
```
```

To run this pipeline over the entire genome, use --allReferences instead of
```
--references=chr17:41196311:41277499.
```

To run the pipeline on a different variant set:
- Change the variant set id for the --variantSetId id parameter.
- Update the --references as appropriate (e.g., add/remove the ‘chr’ prefix on reference names).
- Remove the --nonVariantSegments parameter if it is not applicable.

Gather the results into a single file

```
gsutil cat gs://YOUR-BUCKET/output/platinum-genomes-brca1-ibs.tsv* \ 
   | sort > platinum-genomes-brca1-ibs.tsv
```

Additional details

If the Application Default Credentials are not sufficient, use --client-secrets PATH/TO/YOUR/client_secrets.json. If you do not already have this file, see the authentication instructions to obtain it.

Use --help to get more information about the command line options. Change the pipeline class name below to match the one you would like to run.

```
java -cp google-genomics-dataflow*runnable.jar \
   com.google.cloud.genomics.dataflow.pipelines.VariantSimilarity --help
```

See the source code for implementation details: https://github.com/googlegenomics/dataflow-java

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Genome-Wide Association Study (GWAS)

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

Google BigQuery can be used to perform a GWAS. Here are several examples:
• Chi-squared tests on \textit{1,000 Genomes} dataset with members of EAS super population as case and control all other populations:
  – iPythonNotebook \texttt{Genome-wide association study (GWAS).ipynb}
  – SQL \texttt{gwas-pattern-chi-squared-test.sql}
• Two-proportion Z test on \textit{1,000 Genomes} dataset with members of EAS super population as case and control all other populations:
  – SQL \texttt{gwas-pattern-two-proportion-z-test.sql}
• Chi-squared test on \textit{1,000 Genomes} dataset with case and control determined by clustering from a PCA:
  – R package vignette \texttt{AllModalitiesDemo.md}
  – written as a codelab \texttt{AllModalitiesDemo.md}

To run this on your own data:

1. First, load your data into Google Genomics and export your variants to BigQuery. See Load Genomic Variants for more detail as to how to do this.
2. For data with non-variant segments (e.g, gVCF or Complete Genomics data), reshape the data into multi-sample variants format via Multi-Sample Variants Format

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Annotate Variants

There are many ways to annotate cloud-resident genomic variants.

Tute Genomics Annotation

Tute Genomics has made available to the community annotations for all hg19 SNPs as a BigQuery table.

• For more details about the annotation databases included, see Tute’s blog post.
• For sample queries on public data, see https://github.com/googlegenomics/bigquery-examples/tree/master/platinumGenomes
Google Cloud Platform data locations

- Google Cloud Storage folder gs://tute_db
- Google BigQuery Dataset ID silver-wall-555:TuteTable.hg19

To make use of this on your own data:

1. First, load your data into Google Genomics and export your variants to BigQuery. See Load Genomic Variants for more detail as to how to do this.

2. Copy and modify one of the queries in https://github.com/googlegenomics/bigquery-examples/tree/master/platinumGenomes so that it will perform a JOIN command against your table.

3. Run the revised query with BigQuery to join the Tute table with your variants and materialize the result to a new table. Notice in the screenshot below the destination table and ‘Allow Large Results’ is checked.
-- Annotate all SNPs from the Platinum Genomes cohort.

WITH cohort_variants AS (
    SELECT
      REGEXP_EXTRACT(reference_name, r'chr\d+') AS chr,
      start AS start,
      reference_bases,
      alt
    FROM "genomics-public-data.platinum_genomes.variants" v,
    v.alternate_bases alt WITH OFFSET alt_offset
    WHERE
      -- Require that at least one sample in the cohort has this variant.
      EXISTS(SELECT gt FROM UNNEST(v.call) call, UNNEST(call.genotype) gt WHERE gt = alt_offset+1)
),

--

SELECT
  annots.Chr,
  annots.Start,
  Ref,
  annots.Alt,
  Func,
  Gene,
  PopFreqMax,
  ExonicFunc,
  ClinVar_SIG,
  ClinVar_DIS
FROM "silver-wall-555.TutoTable.hgl9" AS annots
JOIN cohort_variants AS vars
ON vars.chr = annots.Chr
AND vars.start = annots.Start
AND vars.reference_bases = annots.Ref
AND vars.alt = annots.Alt

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Bioconductor Annotation

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.
Bioconductor provides a convenient way to annotate small regions of the genome.

```r
require(GoogleGenomics)
require(VariantAnnotation)
require(BSgenome.Hsapiens.UCSC.hg19)
require(TxDb.Hsapiens.UCSC.hg19.knownGene)

GoogleGenomics::authenticate("/PATH/TO/YOUR/client_secrets.json")

variants <- getVariants(datasetId="10473108253681171589", chromosome="17",
                        start=41196311, end=41277499)
granges <- variantsToGRanges(variants)

granges$ref <- lapply(granges$ref, function(x) substr(x, 1, 1))

granges$ALT <- lapply(granges$ALT, function(x) substr(x, 2))

granges$END <- granges$END - 1

granges <- granges[order(granges$start),]

granges$len <- sapply(granges$end, function(x) x - granges$start)

taxdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

codingVariants <- locateVariants(granges, taxdb, CodingVariants())
codingVariants

coding <- predictCoding(rep(granges$len, elementLengths(granges$ALT)),
                          taxdb, seqSource=Hsapiens,
                          varAllele=unlist(granges$ALT, use.names=FALSE))
coding
```

A more extensive example of variant annotation with Bioconductor is documented towards the end of codelab Data Analysis using Google Genomics.

To make use of this upon your own data:

1. First, load your data into Google Genomics. See `/use_cases/load_data/index` for more detail as to how to do this.
2. If you do not have them already, install the necessary Bioconductor packages. See Using Bioconductor for more detail as to how to do this.
3. Update the parameters to the `getVariants` call the example above to match that of your data and desired genomic region to annotate.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

**Interval JOINs**

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

If you want to use BigQuery to JOIN variants with other data described by genomic region intervals (overlaps), this page demonstrates the use of a complex JOIN predicate.

**Example**

Let’s use a concrete example: Suppose you have a list of gene names and you want to find all the rare SNPs overlapping those genes and also 100,000 bp on either side of the gene for all of your whole genome samples.

1. The first thing we need to do is load or create our table of intervals.
   - If you have a BED file containing your intervals of interest, you can upload that to BigQuery and use it directly.

4.3. Annotate Variants
• Alternatively, the *Tute Genomics Annotation* table has the gene positions for hg19 which we can use to create our interval table. For example:

```sql
SELECT
  Gene,
  Chr,
  MIN(Start) AS gene_start,
  MAX(`End`) AS gene_end,
  MIN(`End`) - 100000 AS region_start,
  MAX(`End`) + 100000 AS region_end
FROM
  `silver-wall-555.TuteTable.hg19` WHERE
  Gene IN ('APC', 'ATM', 'BMPR1A', 'BRCA1', 'BRCA2', 'CDK4',
    'CDKN2A', 'CREBBP', 'EGFR', 'EP300', 'ETV6', 'FHIT', 'FLT3',
    'HRAS', 'KIT', 'MET', 'MLH1', 'NTRK1', 'PAX8', 'PDGFRA',
    'PPARG', 'PRCC', 'PRKAR1A', 'PTEN', 'RET', 'STK11',
    'TFE3', 'TGFB1', 'TGFB2', 'TP53', 'WWOX')
GROUP BY
  Chr,
  Gene
```

2. Suppose we have materialized our interval table to `test.myIntervalTable` and at a minimum it contains columns `region_start` and `region_end`. Now we can run the following query to identify rare variants within our cohort that overlap the regions of interest.

```sql
WITH
  -- Retrieve the variants in this cohort, flattening by alternate bases and
  -- counting affected alleles.
  variants AS ( SELECT
    reference_name,
    start,
    `end`,
    reference_bases,
    alt,
    (SELECT COUNTIF(gt = alt_offset+1) FROM v.call call, call.genotype gt) AS num_variant_alleles,
    (SELECT COUNTIF(gt >= 0) FROM v.call call, call.genotype gt) AS total_num_alleles
  FROM
    `genomics-public-data.1000_genomes_phase_3.variants_20150220_release` v, v.alternate_bases alt WITH OFFSET alt_offset ),
  -- JOIN the variants with the genomic intervals overlapping
  -- the genes of interest.
  -- The JOIN criteria is complicated since we are trying to see if a SNP
  -- overlaps an interval. With standard SQL we can use complex JOIN
  -- predicates, including arbitrary expressions.
  gene_variants AS ( SELECT
    reference_name,
    start,
    reference_bases,
    alt,
    num_variant_alleles,
    total_num_alleles
```
FROM variants
JOIN test.myIntervalTable AS intervals ON
  variants.reference_name = intervals.Chr
  AND intervals.region_start <= variants.start
  AND intervals.region_end >= variants.`end`),
--
-- Retrieve annotations for rare variants only.
select annotations AS (SELECT
  Chr,
  Start,
  Ref,
  Alt,
  Func,
  Gene,
  PopFreqMax,
  ExonicFunc
FROM `silver-wall-555.TuteTable.hg19`
WHERE
  PopFreqMax <= 0.01 )
--
-- And finally JOIN the variants in the regions of interest
-- with annotations for rare variants.
SELECT
  Chr,
  annots.Start AS Start,
  Ref,
  annots.Alt,
  Func,
  Gene,
  PopFreqMax,
  ExonicFunc,
  num_variant_alleles,
  total_num_alleles
FROM rare_variant_annotations AS annots
JOIN gene_variants AS vars ON
  vars.reference_name = annots.Chr
  AND vars.start = annots.Start
  AND vars.reference_bases = annots.Ref
  AND vars.alt = annots.Alt
ORDER BY
  Chr,
  Start

**Results**

A specific run of the above interval JOIN took

*Query complete (92.1s elapsed, 3.38 TB processed)*

on:

4.3. **Annotate Variants**
• 2,504 samples for 84,801,867 phase 3 variants from 1,000 Genomes
• the nearly 9 billion row Tute Genomics Annotation table
• and a gene list containing 250 randomly chosen genes via the following query

```
SELECT
  Gene,
  Chr,
  MIN(Start) AS gene_start,
  MAX(`End`) AS gene_end,
  MIN(Start) - 100000 AS region_start,
  MAX(`End`) + 100000 AS region_end
FROM
  `silver-wall-555.TuteTable.hg19`
WHERE
  Gene IN (SELECT Gene FROM `silver-wall-555.TuteTable.hg19` GROUP BY Gene
              LIMIT 250)
GROUP BY
  Chr,
  Gene
```

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

**Annovar Annotation**

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

If your source data is single-sample VCF, gVCF, or Complete Genomics masterVar format, this page offers some solutions to annotate all variants found within the cohort using Annovar or similar tools.

1. First, load your data into Google Genomics and export your variants to BigQuery. See Load Genomic Variants for more detail as to how to do this.

2. Note that merging has occurred during the import process, so each unique variant within the cohort will be a separate record within the variant set, with all calls for that variant nested within the record. For more information see Variant Import merge logic details.

3. To create an export file similar to a VCF, run a query like the following and materialize the results to a new table. https://github.com/StanfordBioinformatics/mvp_aaa_codelabs/blob/master/sql/multisample-vcf.sql

4. Export the table to Cloud Storage and then download it to a Compute Engine instance with sufficient disk space.

5. Use sed or another file editing tool to finish the transformation needed. See also https://github.com/StanfordBioinformatics/mvp_aaa_codelabs/blob/master/bin/bq-to-vcf.py For example:

   • Add the #CHROM POS ID REF ALT QUAL FILTER INFO FORMAT header line.

   • Convert commas to tabs.

6. Then run Annovar or similar tools on the file(s).

7. Lastly, import the result of the annotation back into BigQuery for use in your analyses.
Annotate Variants with Google Genomics

Variant annotation is a mechanism for finding and filtering interesting variants in a given variant set. An annotated variant set might be used to identify variants which affect a gene of interest, or to highlight potential rare variants in an individual.

A Google Cloud Dataflow implementation is available.

Setup Dataflow

To launch the job from your local machine: Show/Hide Instructions

Most users launch Dataflow jobs from their local machine. This is unrelated to where the job itself actually runs (which is controlled by the --runner parameter). Either way, Java 8 is needed to run the Jar that kicks off the job.

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart including installing gcloud and running gcloud init.

To launch the job from Google Cloud Shell: Show/Hide Instructions

If you do not have Java on your local machine, the following setup instructions will allow you to launch Dataflow jobs using the Google Cloud Shell:

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart.
3. Use the Cloud Console to activate the Google Cloud Shell.
4. Run the following commands in the Cloud Shell to install Java 8.

```
sudo apt-get update
dsudo apt-get install --assume-yes openjdk-8-jdk maven
dsudo update-alternatives --config java
dsudo update-alternatives --config javac
```
**Note:** Depending on the pipeline, Cloud Shell may not have sufficient memory to run the pipeline locally (e.g., without the `--runner` command line flag). If you get error `java.lang.OutOfMemoryError: Java heap space`, follow the instructions to run the pipeline using Compute Engine Dataflow workers instead of locally (e.g., use `--runner=DataflowPipelineRunner`).

If you want to run a small pipeline on your machine before running it in parallel on Compute Engine, you will need ALPN since many of these pipelines require it. When running locally, this must be provided on the boot classpath but when running on Compute Engine Dataflow workers this is already configured for you. You can download it from here. For example:

```
wget -O alpn-boot.jar \
```

Download the latest GoogleGenomics dataflow **runnable** jar from the [Maven Central Repository](https://search.maven.org/remotecontent?filepath=com/google/cloud/genomics/google-genomics-dataflow/v1-0.1/google-genomics-dataflow-v1-0.1-runnable.jar).

Run the pipeline

The following command will use ClinVar to annotate variants on a portion of chromosome 17 within the *Illumina Platinum Genomes* dataset for individual NA12877.

**Disclaimer:** This program is currently intended to be a sample, and provides only a small subset of functionality found in most variant annotation programs. Pull requests are welcome!

```
java -Xbootclasspath/p:alpn-boot.jar \ 
    -cp google-genomics-dataflow-runnable.jar \ 
    com.google.cloud.genomics.dataflow.pipelines.AnnotateVariants \ 
    --variantSetId=3049512673186936334 \ 
    --references=chr17:40700000:40800000 \ 
    --transcriptSetIds=CIjfoPXj9LqP1AEQ5vnq14KewYuSAQ \ 
    --variantAnnotationSetIds=CILSqfjtlY6tHxC0nNH-4cu-x1Q \ 
    --callSetIds=3049512673186936334-0 \ 
    --output=gs://YOUR-BUCKET/dataflow-output/platinum-genomes-chr17region-annotation.\tsv
```

You can check your results by ensuring that the following three results are contained in the output files:

```
chr17:40714803:A:CI7s77ro84KpKhIFY2hyMTcYa4S1EyDwuoPBI1PDR19AB: [{alternateBases=A, \n    effect=NONSYNONYMOUS_SNP, geneId=ChYIiN-g9eP0uo-UARDi_aPt7qzv9tweB(glx)nxjxjr_ \n    rQT1JrU8My-4_2U(d)a, transcriptIds=[ChYIiN-g9eP0uo-UARDm-eqXgp7Bi5IB(gl)xnxjxjr_rQTII_ \n    3bW3_PSH6AE, type=SNP}]
```

```
chr17:40722028:G:CI7s77ro84KpKhIFY2hyMTcY7Ly1EydvgexCryb2xrQw: [{alternateBases=G, \n    effect=NONSYNONYMOUS_SNP, geneId=ChYIiN-g9eP0uo-UARDi_ \n    aPt7qzv9tweB(gl)xnxjxjrUTIL3v58kGBMzFjw, transcriptIds=[ChYIiN-g9eP0uo-UARDm- \n    eqXgp7Bi5IB(g)lxnxjxjrUTIMx96zMyV0gE], type=SNP}]
```

```
chr17:4076905:A:CI7s77ro84KpKhIFY2hyMTcY2caOeCy4W4NnN8qzS8S0: [{alternateBases=A, \n    effect=NONSYNONYMOUS_SNP, geneId=ChYIiN-g9eP0uo-UARDi_ \n    aPt7qzv9tweB(gl)xnxjxjr7QT1I726M7yo08Cna, transcriptIds=[ChYIiN-g9eP0uo-UARDm- \n    eqXgp7Bi5IB(g)xnxjxjr7QTINX5KoLHyHYkwE], type=SNP}]
```
The above command line runs the pipeline locally over a small portion of the genome, only taking a few minutes. If modified to run over a larger portion of the genome or the entire genome, it may take a few hours depending upon how many virtual machines are configured to run concurrently via `--numWorkers`. Add the following additional command line parameters to run the pipeline on Google Cloud instead of locally:

```
--runner=DataflowPipelineRunner \
--project=YOUR-GOOGLE-CLOUD-PLATFORM-PROJECT-ID \
--stagingLocation=gs://YOUR-BUCKET/dataflow-staging \
--numWorkers=
```

Use a comma-separated list to run over multiple disjoint regions. For example to run over BRCA1 and BRCA2:

```
```

To run this pipeline over the entire genome, use `--allReferences` instead of `--references=chr17:41196311:41277499`.

Note that this program accepts `VariantSets` and `AnnotationSets` as input. The analogous inputs to traditional variant annotation programs are `VCF` and `.csv` files, respectively. To run the pipeline on a different `VariantSet` change the `--variantSetId` id parameter and:

- update the `--callSetIds` parameter accordingly
- see `ClinVar Annotations` if the variants were aligned to GRCh38 instead of GRCh37/hg19 and update `--variantAnnotationSetIds`
- see `UCSC Annotations` if the variants were aligned to GRCh38 instead of GRCh37/hg19 and update `--transcriptSetIds`

### Gather the results into a single file

```
gsutil cat gs://YOUR-BUCKET/output/platinum-genomes-brca1-clinvar-annotation.tsv* \  
| sort > platinum-genomes-brca1-clinvar-annotation.tsv
```

### Additional details

If the Application Default Credentials are not sufficient, use `--client-secrets PATH/TO/YOUR/client_secrets.json`. If you do not already have this file, see the [authentication instructions](https://cloud.google.com/genomics/support) to obtain it.

Use `--help` to get more information about the command line options. Change the pipeline class name below to match the one you would like to run.

```
java -cp google-genomics-dataflow*runnable.jar \  
com.google.cloud.genomics.dataflow.pipelines.VariantSimilarity --help
```

See the source code for implementation details: [https://github.com/googlegenomics/dataflow-java](https://github.com/googlegenomics/dataflow-java)

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see [https://cloud.google.com/genomics/support](https://cloud.google.com/genomics/support).
Perform Quality Control Checks on Reads

This pipeline tests a set of reads for contamination. It takes as input:

- a set of ReadGroupSets to test
- statistics on reference allele frequencies for SNPs with a single alternative from a set of VariantSets

and combines these to produce an estimate of the amount of contamination.

Uses the sequence data alone approach described in:

Detecting and Estimating Contamination of Human DNA Samples in Sequencing and Array-Based Genotype Data
Jun, Goo et al.
The American Journal of Human Genetics, Volume 91, Issue 5, 839 - 848
DOI: http://dx.doi.org/10.1016/j.ajhg.2012.09.004

The pipeline is implemented on Google Cloud Dataflow.

Setup Dataflow

Note: this pipeline is new and still undergoing testing. We recommend that you follow the instructions here to build the latest version of the source code.

To launch the job from your local machine: Show/Hide Instructions

Most users launch Dataflow jobs from their local machine. This is unrelated to where the job itself actually runs (which is controlled by the --runner parameter). Either way, Java 8 is needed to run the Jar that kicks off the job.

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart including installing gcloud and running gcloud init.
To launch the job from Google Cloud Shell: **Show/Hide Instructions**

If you do not have Java on your local machine, the following setup instructions will allow you to launch Dataflow jobs using the Google Cloud Shell:

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart.
3. Use the Cloud Console to activate the Google Cloud Shell.
4. Run the following commands in the Cloud Shell to install Java 8.

```bash
sudo apt-get update
ds
sudo apt-get install --assume-yes openjdk-8-jdk maven
ds
sudo update-alternatives --config java
ds
sudo update-alternatives --config javac
```

**Note:** Depending on the pipeline, Cloud Shell may not have sufficient memory to run the pipeline locally (e.g., without the **--runner** command line flag). If you get error `java.lang.OutOfMemoryError: Java heap space`, follow the instructions to run the pipeline using Compute Engine Dataflow workers instead of locally (e.g. use **--runner=DataflowPipelineRunner**).

If you want to run a small pipeline on your machine before running it in parallel on Compute Engine, you will need **ALPN** since many of these pipelines require it. When running locally, this must be provided on the boot classpath but when running on Compute Engine Dataflow workers this is already configured for you. You can download it from here. For example:

```bash
wget -O alpn-boot.jar 
  alpn-boot-8.1.8.v20160420.jar
```

Download the latest GoogleGenomics dataflow runnable jar from the Maven Central Repository. For example:

```bash
wget -O google-genomics-dataflow-runnable.jar 
  https://search.maven.org/remotecontent?filepath=com/google/cloud/genomics/google-genomics-dataflow/v1-0.1/google-genomics-dataflow-v1-0.1-runnable.jar
```

### Run the pipeline

The following command will calculate the contamination estimate for a given ReadGroupSet and specific region in the **1,000 Genomes** dataset. It also uses the VariantSet within **1,000 Genomes** for retrieving the allele frequencies.

```bash
java -Xbootclasspath/p:alpn-boot.jar 
  -cp google-genomics-dataflow-runnable.jar 
  com.google.cloud.genomics.dataflow.pipelines.VerifyBamId 
  --references=17:41196311:41277499 
  --readGroupSetIds=CMvnhpKTFhDq9e2Yy9G-Bg 
  --variantSetId=10473108253681171589 
  --output=gs://YOUR-BUCKET/dataflow-output/verifyBamId-platinumGenomes-BRCA1-
  readGroupSet-CMvnhpKTFhCAv6TKo6Dglgg.txt
```

The above command line runs the pipeline locally over a small portion of the genome, only taking a few minutes. If modified to run over a larger portion of the genome or the entire genome, it may take a few hours depending upon how many virtual machines are configured to run concurrently via **--numWorkers**. Add the following additional command line parameters to run the pipeline on Google Cloud instead of locally:

### 4.4. Perform Quality Control Checks
Use a comma-separated list to run over multiple disjoint regions. For example to run over BRCA1 and BRCA2

```
```

To run this pipeline over the entire genome, use `--allReferences` instead of

```
--references=chr17:41196311:41277499.
```

To run the pipeline on a different group of read group sets: * Change the `--readGroupSetIds` or the `--inputDatasetId` parameter. * Update the `--references` as appropriate (e.g., add/remove the ‘chr’ prefix on reference names).

To configure the pipeline more to fit your needs in terms of the minimum allele frequency to use or the fraction of positions to check, change the `--minFrequency` and `--samplingFraction` parameters.

**Additional details**

If the Application Default Credentials are not sufficient, use `--client-secrets PATH/TO/YOUR/client_secrets.json`. If you do not already have this file, see the authentication instructions to obtain it.

Use `--help` to get more information about the command line options. Change the pipeline class name below to match the one you would like to run.

```
java -cp google-genomics-dataflow*runnable.jar \
    com.google.cloud.genomics.dataflow.pipelines.VariantSimilarity --help
```

See the source code for implementation details: https://github.com/googlegenomics/dataflow-java

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

**Perform Quality Control on Variants**

The properly rendered version of this document can be found at Read The Docs.

If you are reading this on github, you should instead click here.

There are a collection of quality control checks for variants documented in codelab Quality Control using Google Genomics. The methods include:

- Sample Level
  - Genome Call Rate
  - Missingness Rate
  - Singleton Rate
  - Heterozygosity Rate
  - Homozygosity Rate
  - Inbreeding Coefficient
Sex Inference
- Ethnicity Inference
- Genome Similarity

- Variant Level
  - Ti/Tv by Genomic Window
  - Ti/Tv by Alternate Allele Counts
  - Ti/Tv by Depth
  - Missingness Rate
  - Hardy-Weinberg Equilibrium
  - Heterozygous Haplotype

These methods were co-developed with researchers working on the Million Veterans Program data. For more detail, please see the paper and diagram of their full pipeline with some additional quality control checks on github.

To make use of this codelab upon your own data:

1. First, load your data into Google Genomics and export your variants to BigQuery. See Load Genomic Variants for more detail as to how to do this.

2. Each section of the codelab discusses how to run that part on your own data. For example, update the BigQuery table name in Part 1: Data Overview

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

**Compute and Analyze Linkage Disequilibrium**

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

**Compute Linkage Disequilibrium on a Variant Set**

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

Contents

- Compute Linkage Disequilibrium on a Variant Set
  - Setup Dataflow
This pipeline calculates linkage disequilibrium between pairs of variants in a Global Alliance VariantSet (which you can create from a VCF file). It takes as input a VariantSet for which the linkage disequilibrium values will be calculated and calculates the D' and allelic correlation measures of linkage disequilibrium, defined in Box 1 of:

Linkage disequilibrium - understanding the evolutionary past and mapping the medical future
Slatkin, Montgomery
Nature Reviews Genetics, Volume 9, Issue 6, 477 - 485
DOI: http://dx.doi.org/10.1038/nrg2361

The pipeline is implemented on Google Cloud Dataflow.

Setup Dataflow

To launch the job from your local machine: Show/Hide Instructions

Most users launch Dataflow jobs from their local machine. This is unrelated to where the job itself actually runs (which is controlled by the --runner parameter). Either way, Java 8 is needed to run the Jar that kicks off the job.

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart including installing gcloud and running gcloud init.

To launch the job from Google Cloud Shell: Show/Hide Instructions

If you do not have Java on your local machine, the following setup instructions will allow you to launch Dataflow jobs using the Google Cloud Shell:

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart.
3. Use the Cloud Console to activate the Google Cloud Shell.
4. Run the following commands in the Cloud Shell to install Java 8.

```
sudo apt-get update
sudo apt-get install --assume-yes openjdk-8-jdk maven
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

Note: Depending on the pipeline, Cloud Shell may not have sufficient memory to run the pipeline locally (e.g., without the --runner command line flag). If you get error java.lang.OutOfMemoryError: Java heap space, follow the instructions to run the pipeline using Compute Engine Dataflow workers instead of locally (e.g. use --runner=DataflowPipelineRunner).

If you want to run a small pipeline on your machine before running it in parallel on Compute Engine, you will need ALPN since many of these pipelines require it. When running locally, this must be provided on the boot classpath but
when running on Compute Engine Dataflow workers this is already configured for you. You can download it from here. For example:

```bash
```

Build the Linkage Disequilibrium jar:

```bash
git clone https://github.com/googlegenomics/linkage-disequilibrium.git
cd linkage-disequilibrium
mvn package
```

**Run the pipeline**

The following command will calculate linkage disequilibrium between all pairs of variants within 50,000 base pairs of each other for a specific region in the *1,000 Genomes* Phase 3 VariantSet, and retain results for all pairs that have an absolute value of their allelic correlation of at least 0.4.

```java
java -Xbootclasspath/p:alpn-boot.jar -cp target/linkage-disequilibrium*runnable.jar com.google.cloud.genomics.dataflow.pipelines.LinkageDisequilibrium --variantSetId=11027761582969783635 --references=17:41196311:41277499 --window=50000 --ldCutoff=0.4 --output=gs://YOUR-BUCKET/dataflow-output/linkage-disequilibrium-1000G_Phase_3-BRCA1.txt
```

The above command line runs the pipeline locally over a small portion of the genome, only taking a few minutes. If modified to run over a larger portion of the genome or the entire genome, it may take a few hours depending upon how many virtual machines are configured to run concurrently via `--numWorkers`. Add the following additional command line parameters to run the pipeline on Google Cloud instead of locally:

```bash
--runner=DataflowPipelineRunner --project=YOUR-GOOGLE-CLOUD-PLATFORM-PROJECT-ID --stagingLocation=gs://YOUR-BUCKET/dataflow-staging --numWorkers=
```

Use a comma-separated list to run over multiple disjoint regions. For example to run over BRCA1 and BRCA2:

```bash
```

To run this pipeline over the entire genome, use `--allReferences` instead of:

```bash
--references=chr17:41196311:41277499.
```

To run the pipeline on a subset of individuals in a VariantSet:

- Add a `--callSetsToUse` flag that has a comma-delimited list of call sets to include.

**Additional details**

Use `--help` to get more information about the command line options. Change the pipeline class name below to match the one you would like to run.
These pipelines take linkage disequilibrium (LD) results generated by the `Compute Linkage Disequilibrium on a Variant Set` pipeline and transforms them into Cloud BigQuery and Cloud BigTable datasets that can be efficiently analyzed. Each pipeline takes as input a set of LD results and exports, transforms, and loads the results into the appropriate target data store.

The pipelines are implemented on Google Cloud Dataflow.

**Load Linkage Disequilibrium Data into Cloud BigQuery**

The output of the LinkageDisequilibrium pipeline is a file of comma-separated values. This is a standard file format for BigQuery ingestion. General instructions for loading data into BigQuery are available [here](https://cloud.google.com/bigquery/docs/how-to-load-data). An example script for loading data from a CSV is available at `load_data_from_csv.py`.

When using that script, the `schema` for the table of LD results is available in the linkage disequilibrium repository in the `ld_bigquery_schema_fields.txt` file.

Consequently, LD data can be loaded into a BigQuery table with the following code snippet:

```
PROJECTID=<your-project-id>
DATASETID=<your-bigquery-dataset-id>
TABLE=<your-desired-bigquery-table-name>
DATA=<path-to-linkage-disequilibrium-result-data>

python path/to/load_data_from_csv.py \
    $PROJECTID $DATASETID $TABLE schema/ld_bigquery_schema_fields.txt $DATA
```
**Setup Dataflow**

To *launch* the job from your local machine: **Show/Hide Instructions**

Most users *launch* Dataflow jobs from their local machine. This is unrelated to where the job itself actually runs (which is controlled by the `--runner` parameter). Either way, Java 8 is needed to run the Jar that kicks off the job.

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart including installing `gcloud` and running `gcloud init`.

To *launch* the job from Google Cloud Shell: **Show/Hide Instructions**

If you do not have Java on your local machine, the following setup instructions will allow you to *launch* Dataflow jobs using the Google Cloud Shell:

1. If you have not already done so, follow the Genomics Quickstart.
2. If you have not already done so, follow the Dataflow Quickstart.
3. Use the Cloud Console to activate the Google Cloud Shell.
4. Run the following commands in the Cloud Shell to install Java 8.

```
sudo apt-get update
sudo apt-get install --assume-yes openjdk-8-jdk maven
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

**Note:** Depending on the pipeline, Cloud Shell may not have sufficient memory to run the pipeline locally (e.g., without the `--runner` command line flag). If you get error `java.lang.OutOfMemoryError`, follow the instructions to run the pipeline using Compute Engine Dataflow workers instead of locally (e.g., use `--runner=DataflowPipelineRunner`).

If you want to run a small pipeline on your machine before running it in parallel on Compute Engine, you will need ALPN since many of these pipelines require it. When running locally, this must be provided on the boot classpath but when running on Compute Engine Dataflow workers this is already configured for you. You can download it from here.

```
```

Build the Linkage Disequilibrium jar:

```
git clone https://github.com/googlegenomics/linkage-disequilibrium.git
cd linkage-disequilibrium
mvn package
```

**Load Linkage Disequilibrium Data into Cloud BigTable**

Because BigTable allows efficient access to extremely large datasets indexed by a single key, it is a natural choice for representation of LD data. The `WriteLdBigtable` pipeline converts data generated by the `Compute Linkage Disequilibrium on a Variant Set` pipeline and writes the results into a BigTable using Dataflow. The key for each BigTable row is designed so that all LD results for a single query variant appear in a contiguous block of the table, sorted by
the location of the target variants, and results for query variants are sorted by the location of query variants. This key design allows efficient access to all LD results for a single variant or a single region of the genome.

The following command will load LD results into an existing BigTable:

```
java -Xbootclasspath/p:alpn-boot.jar \
   -cp target/linkage-disequilibrium*runnable.jar \
   com.google.cloud.genomics.dataflow.pipelines.WriteLdBigtable \
   --bigtableProjectId=YOUR_BIGTABLE_PROJECT_ID \
   --bigtableClusterId=YOUR_BIGTABLE_CLUSTER_ID \
   --bigtableZoneId=YOUR_BIGTABLE_ZONE_ID \
   --bigtableTableId=YOUR_BIGTABLE_TABLE_ID \
   --ldInput="gs://YOUR-BUCKET/PATH-TO-DIRECTORY-WITH-LD-RESULTS/*"
```

The above command line runs the pipeline locally over a small portion of the genome, only taking a few minutes. If modified to run over a larger portion of the genome or the entire genome, it may take a few hours depending upon how many virtual machines are configured to run concurrently via `--numWorkers`. Add the following additional command line parameters to run the pipeline on Google Cloud instead of locally:

```
--runner=DataflowPipelineRunner \
--project=YOUR-GOOGLE-CLOUD-PLATFORM-PROJECT-ID \
--stagingLocation=gs://YOUR-BUCKET/dataflow-staging \
--numWorkers=#
```

## Retrieve Linkage Disequilibrium Data from Cloud BigTable

Once a BigTable storing LD data has been created, a mechanism for accessing the results must be created. The `QueryLdBigtable` pipeline provides an example in which Dataflow is used to read a subset of data from an LD BigTable and write the results to GCS in the same format as it was originally written by the `Compute Linkage Disequilibrium on a Variant Set` pipeline.

The following command will query LD results for a specific region of the genome and write results to a Cloud bucket:

```
java -Xbootclasspath/p:alpn-boot.jar \
   -cp target/linkage-disequilibrium*runnable.jar \
   com.google.cloud.genomics.dataflow.pipelines.QueryLdBigtable \
   --bigtableProjectId=YOUR_BIGTABLE_PROJECT_ID \
   --bigtableClusterId=YOUR_BIGTABLE_CLUSTER_ID \
   --bigtableZoneId=YOUR_BIGTABLE_ZONE_ID \
   --bigtableTableId=YOUR_BIGTABLE_TABLE_ID \
   --queryRange="17:41196311-41277499" \
   --resultLocation="gs://YOUR-BUCKET/PATH-TO-OUTPUT-FILE"
```

Use a comma-separated list to run over multiple disjoint regions. For example to run over BRCA1 and BRCA2:

```
```

To run this pipeline over the entire genome, use `--allReferences` instead of `--references=chr17:41196311:41277499`.

## Additional details

Use `--help` to get more information about the command line options. Change the pipeline class name below to match the one you would like to run.
```
java -cp target/linkage-disequilibrium*runnable.jar \
    com.google.cloud.genomics.dataflow.pipelines.LinkageDisequilibrium \
    --help=com.google.cloud.genomics.dataflow.pipelines.LinkageDisequilibrium\n    "$LinkageDisequilibriumOptions"
```

See the source code for implementation details: https://github.com/googlegenomics/linkage-disequilibrium

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

### Analyze Linkage Disequilibrium Results

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

There are several examples of interacting with LD results stored in BigQuery using Datalab in GitHub. The examples are all part of the linkage disequilibrium project.

- Exploring summary statistics of LD data.
- Visualizing LD patterns in specific genomic regions.
- Examining the rate of LD decay as a function of distance.
- Selecting “tag variants” and visualizing tag variant distributions.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

### Advanced BigQuery Topics

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

**Multi-Sample Variants Format**

The properly rendered version of this document can be found at Read The Docs. If you are reading this on github, you should instead click here.

If your source data is jointly-called (e.g., like 1,000 Genomes) it will already be in “multi-sample variants” format when it is exported from the Variants API to Google BigQuery.

If your source data is single-sample gVCF or Complete Genomics masterVar format, this page offers some solutions to convert it to multi-sample variants format.

4.6. Advanced BigQuery Topics
Overview

Suppose you have imported your single-sample files to the Variants API and exported them to BigQuery. Let’s refer to this original table as the “genome calls” table. It contains all reference calls and variant calls.

To facilitate variant-centric analysis like we see in the BigQuery 1,000 Genomes samples, we can generate a second table, the “multi-sample variants” table. The multi-sample variants table resembles a multi-sample VCF file. In this table:

- every variant record includes calls for all callsets
- variants which contained only reference calls for all callsets are omitted

Motivation

Data from source files in genome VCF (gVCF) format or in Complete Genomics format can be challenging to query due to the presence of non-variant segment records.

For example to lookup rs9536314 in the Klotho gene, the WHERE clause

```sql
WHERE
    reference_name = 'chr13'
    AND start = 33628137
```

becomes

```sql
WHERE
    reference_name = 'chr13'
    AND start <= 33628137
    AND end >= 33628138
```

to capture not only that variant, but any other records that overlap that genomic position.

Suppose we want to calculate an aggregate for a particular variant, such as the number of samples with the variant on one or both alleles and of samples that match the reference? The WHERE clause above will do the trick. But then suppose we want to do this for all SNPs in our dataset?

Examples

There are a few ways to generate the multi-sample variants table for use in variant-centric analyses such as Genome-Wide Association Study (GWAS):

- Use a cluster computing job to transform data with non-variant segments to variant-only data with calls from non-variant-segments merged into the variants with which they overlap. This is currently done only for SNP variants. Indels and structural variants are left as-is.
  - see the Dataflow example
  - see the Hadoop Streaming example
- Use BigQuery to transform the data, materialize the result to a new table

A note about scaling: as the number of samples increases, so does the number of private and rare variants. At a certain point there are many, many rows with mostly 0/0 genotypes. We are experimenting with alternate transformations. Comment on this issue if you want a pointer to the most recent prototype.
Migrating from Genomics API v1beta2 to v1

The v1beta2 version of the Google Genomics API is deprecated.
If you use the the Google Genomics v1beta2 API, you will need to update your source code to use the v1 API.
Most changes are minor and updating your code will simply involve changing the name of objects or methods.

API Documentation

- v1beta2: https://cloud.google.com/genomics/v1beta2/reference/
- v1: https://cloud.google.com/genomics/reference/rest/

New client libraries

Python, Java, Go, and other language-specific client libraries for the v1 API can be found at:
- https://cloud.google.com/genomics/v1/libraries

API Changes

New endpoint:

- v1beta2: https://www.googleapis.com/genomics/v1beta2
- v1: https://genomics.googleapis.com/v1
When reaching the end of paged-responses, the v1beta2 API would omit the `nextPageToken` field from the response. The v1 API will always return the `nextPageToken` field. When end of paging is reached, the value will be an empty string.

### New names

<table>
<thead>
<tr>
<th>v1beta2</th>
<th>v1</th>
</tr>
</thead>
<tbody>
<tr>
<td>variantsets.importVariants</td>
<td>variants.import</td>
</tr>
<tr>
<td>variantsets.mergeVariants</td>
<td>variant.merge</td>
</tr>
<tr>
<td>jobs.cancel</td>
<td>operations.cancel</td>
</tr>
<tr>
<td>jobs.get</td>
<td>operations.get</td>
</tr>
<tr>
<td>jobs.search</td>
<td>operations.list</td>
</tr>
</tbody>
</table>

### Changed parameters

For Google Cloud project references, the v1beta2 API used the Cloud project number, while the v1 API uses the Cloud project id.

This change impacts how you call APIs. The following APIs have changed from accepting an input project number to a project id:

- `datasets.list`
- `jobs.search` (now `operations.list`)
- `readgroupsets.export`
- `variantsets.export`

This change impacts objects returned from APIs. The following objects have changed to contain a project id instead of a project number:

- `datasets`
- `jobs` (now `operations`)

### New datatypes

<table>
<thead>
<tr>
<th>Field</th>
<th>v1beta2</th>
<th>v1</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset.create_time</td>
<td>long</td>
<td>A timestamp in RFC3339 UTC “Zulu” format</td>
</tr>
<tr>
<td>read.info</td>
<td>map&lt;string, array&lt;string&gt;&gt; &gt;</td>
<td>map&lt;string, array&lt;object&gt;&gt; &gt;</td>
</tr>
<tr>
<td>annotations.info</td>
<td>map&lt;string, array&lt;string&gt;&gt; &gt;</td>
<td>map&lt;string, array&lt;object&gt;&gt; &gt;</td>
</tr>
</tbody>
</table>

### New objects

**Operations replaces jobs**

The `jobs` object has been replaced with `operations`. 
Job Status

With v1beta2, the status was encoded in the `status` field with detailed information in the `detailedStatus`.

With v1, the status is a combination of `status` field plus the existence of the either the `errors` object (failure) or the `response` object (success).

ID

With v1beta2, the job was identified with the `id` field. With v1, the operation is identified by the `name` field.

importedIds

After a successful import, the job object would be populated with a list of `importedIds`. In the operation, this data is a child field of the `response`, either the `callSetIds` or `readGroupSetIds`.

created

The job `created` time is now in the operation `metadata.createTime` field.

request

The job `request` values are now in the operation `metadata.request` field.

Field Masks

All Genomics APIs accept an optional list of fields to return. This is sometimes referred to as a “field mask”. The `patch` and `update` APIs accept an `updateMask` indicating the specific fields to change.

In v1beta2, the “/” character could be used to separate a parent object from a child field, for example:

- `readGroupSets/id`

In v1, the “/” character is not accepted, and parentheses must wrap the child field:

- `readGroupSets(id)`

You can test building a proper field mask by using the “fields editor” available in the documentation for the Google Genomics API, or from the Google APIs Explorer.

For example, for `genomics.datasets.list` see:

- API Documentation
- APIs Explorer

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
Build your own Google Genomics API Client

The tools for working with the Google Genomics API are all open source and available on GitHub. This documentation covers how to get started with the available tools as well as how you might build your own code which uses the API.

Important constants and links

Google’s base API url is: https://genomics.googleapis.com/v1
More information on the API can be found at: https://cloud.google.com/genomics and http://ga4gh.org
To test Google’s compliance with the GA4GH API, you can use the compliance tests: https://github.com/ga4gh/compliance
To get a list of public datasets that can be used with Google’s API calls, you can use the APIs explorer or Discover Published Data.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Common API workflows

There are many genomics-related APIs documented at cloud.google.com/genomics/reference/rest/.
Of the available calls, there are some very common patterns that can be useful when developing your own code.
The following sections describe these workflows using plain URLs and simplified request bodies. Each step should map 1-1 with all of the auto-generated client libraries.

Browsing read data

- GET /datasets?projectId=YOUR-PROJECT-ID
  List all available datasets in a particular project.
- POST /readgroupsets/search {datasetIds: [<datasetId>]}  
  Search for read group sets in a particular dataset. Choose one readGroupSetId from the result.
  Note: This is a good place to use a partial request to only ask for the id and name fields on a read group set. Then you can follow up with a GET /readgroupsets/<readGroupSetId> call to get the complete read group set data.
- GET /readgroupsets/<readGroupSetId>/coveragebuckets
  Get coverage information for a particular read group set. This will tell you where the read data is located, and which referenceNames should be used in the next step.
**POST /reads/search {readGroupSetIds: [<readGroupSetId>]}

Get reads for a particular read group set.

The referenceName can be chosen from the coverage buckets by the user, along with the start and end coordinates they wish to view. The API uses 0-based coordinates.

### Map reducing over read data within a read group set

**GET /readgroupsets/<readGroupSetId>/coveragebuckets**

First get coverage information for the read group set you are working with.

Iterate over the `coveragebuckets` array. For each bucket, there is a field `range.end`. Using this field, and the number of shards you wish to have, you can calculate sharding bounds.

Let’s say there are 23 references, and you want 115 shards. The easiest math would have us creating 5 shards per reference, each with a `start` of `i * range.end/5` and an `end` of `min(range.end, start + range.end/5)`.

**POST /reads/search {readGroupSetId: x, referenceName: shard.refName, start: shard.start, end: shard.end}**

Once you have your shard bounds, each shard will then do a reads search to get data. (Don’t forget to use a use a partial request)

### Map reducing over variant data

**GET /variantsets/<datasetId>**

First get a summary of the variants you are working with. This includes the references that have data, as well as their upper bounds.

Iterate over the `referenceBounds` array. For each reference, there is a field `upperBound`. Using this field, and the number of shards you wish to have, you can calculate sharding bounds.

Let’s say there are 23 references, and you want 115 shards. The easiest math would have us creating 5 shards per reference, each with a `start` of `i * referenceBounds.upperBound/5` and an `end` of `min(referenceBound.upperBound, start + referenceBounds.upperBound/5)`.

**POST /variants/search {variantSetIds: [x], referenceName: shard.refName, start: shard.start, end: shard.end}**

Once you have your shard bounds, each shard will then do a variants search to get data. (Don’t forget to use a use a partial request)

If you only want to look at certain call sets, you can include the `callSetIds: ["id1", "id2"]` field on the search request. Only call information for those call sets will be returned. Variants without any of the requested call sets will not be included at all.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.
API authorization requirements

Calls to the Google Genomics API can be made with OAuth or with an API key.

- To access private data or to make any write calls, an API request needs to be authenticated with OAuth.
- Read-only calls to public data only require an API key to identify the calling project. (OAuth will also work)

Some APIs are still in the testing phase. The following lays out where each API call stands and also indicates whether a call supports requests without OAuth.

### Available APIs

<table>
<thead>
<tr>
<th>API method</th>
<th>OAuth required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get, List and Search methods (except on Jobs)</td>
<td>False</td>
</tr>
<tr>
<td>Create, Delete, Patch and Update methods</td>
<td>True</td>
</tr>
<tr>
<td>Import and Export methods</td>
<td>True</td>
</tr>
<tr>
<td>All Job methods</td>
<td>True</td>
</tr>
</tbody>
</table>

### APIs in testing

<table>
<thead>
<tr>
<th>API method</th>
<th>OAuth required</th>
</tr>
</thead>
<tbody>
<tr>
<td>genomics.experimental.*</td>
<td>True</td>
</tr>
</tbody>
</table>

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see [https://cloud.google.com/genomics/support](https://cloud.google.com/genomics/support).

The Python client

The api-client-python project provides a simple genome browser that pulls data from the Genomics API.

Setting up the Python client on Windows

- In order to setup Python 2.7 for Windows, first download it from [https://www.python.org/downloads/](https://www.python.org/downloads/)
- After installing Python, add to your PATH the location of the Python directory and the Scripts directory within it.

  For example, if Python is installed in C:\Python27, proceed by right-clicking on My Computer on the Start Menu and select “Properties”. Select “Advanced system settings” and then click on the “Environment Variables” button. In the window that comes up, append the following to the system variable PATH (if you chose a different installation location, change this path accordingly):

  ```
  ;C:\Python27;C:\Python27\Scripts
  ```

- Get the api-client-python code onto your machine by cloning the repository:

  ```
  git clone https://github.com/googlegenomics/api-client-python.git
  ```
Running the client with App Engine

Only follow the instructions in this section if you want to run the Python client with App Engine.

- Download the “Google App Engine SDK for Python” for Windows from https://cloud.google.com/appengine/downloads and install it.

- From within the api-client-python directory that you clones, run the dev_appserver.py script. If we assume the installation directory for your app engine SDK was C:\Google\google_appengine, then you would run the following command:

  ```
  python C:\Google\google_appengine\dev_appserver.py .
  ```

  If you get an error like google.appengine.tools.devappserver2.wsgi_server.BindError: Unable to bind localhost:8000, try specifying a specific port with this command:

  ```
  python C:\Google\google_appengine\dev_appserver.py --admin_port=12000 .
  ```

- To view your running server, open your browser to localhost:8080.

Running the client without App Engine

Only follow the instructions in this section if you do not want to use App Engine. See the section above for App Engine instructions.

- First you will need to download Pip from https://raw.github.com/pypa/pip/master/contrib/get-pip.py

- To install Pip, open up a cmd.exe window by selecting Start->Run->cmd and type the following (replace directory_of_get-pip.py with the location of where get-pip.py resides):

  ```
  cd directory_of_get-pip.py
  python get-pip.py
  ```

- Afterwards in the same command window, type the following command to update your Python environment with the required modules:

  ```
  pip install WebOb Paste webapp2 jinja2
  ```

- You should then be able to run the localserver with the following commands:

  ```
  cd api-client-python
  python localserver.py
  ```

Enabling the Google API provider

If you want to pull in data from Google Genomics API you will need to set API_KEY in main.py to a valid Google API key.

- First apply for access to the Genomics API by following the instructions at https://cloud.google.com/genomics/

- Then create a project in the Google Cloud Platform Console or select an existing one.

- On the APIs & auth tab, select APIs and turn the Genomics API to ON

- On the Credentials tab, click create new key under the Public API access section.
• Select **Server key** in the dialog that pops up, and then click **Create**. (You don’t need to enter anything in the text box)

• Copy the **API key** field value that now appears in the Public API access section into the top of the `main.py` file inside of your api-client-python directory. It should look something like this:

```python
API_KEY = "abcdef12345abcdef"
```

Note: You can also reuse an existing API key if you have one. Just make sure the Genomics API is turned on.

• Run your server as before, and view your server at `localhost:8080`.

• Google should now show up as an option in the Readset choosing dialog.

---

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see [https://cloud.google.com/genomics/support](https://cloud.google.com/genomics/support).

### GABrowse URL format

The genome browser code supports direct linking to specific backends, readsets, and genomic positions. These parameters are set using the hash. The format is very simple with only 3 supported key value pairs separated by `&` and then `=`:

- **backend**
  The backend to use for API calls. example: GOOGLE or NCBI

- **readsetId**
  The ID of the readset that should be loaded. See [Important constants and links](#) for more information.

- **location**
  The genomic position to display at. Takes the form of `<chromosome>:<base pair position>`. example: `14:25419886` This can also be an RS ID or a string that will be searched on [snpedia](https://snpedia.com).

As you navigate in the browser (either locally or at [http://gabrowse.appspot.com](http://gabrowse.appspot.com)), the hash will automatically populate to include these parameters. But you can also manually create a direct link without having to go through the UI.

Putting all the pieces together, here is what a valid url looks like:

```
http://gabrowse.appspot.com/#backend=GOOGLE&readsetId=CPHG3MzoCRDY5Ircq2q8hMIB&location=14:25419886
```

---

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see [https://cloud.google.com/genomics/support](https://cloud.google.com/genomics/support).

The Python client does not currently use Google’s Python client library. If you want to use the client library, the method documentation for genomics can be very useful.

---

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.
BioC 2015: Where Software and Biology Connect

Google has some pretty amazing big data computational “hammers” that they have been applying to search and video data for a long time. In this workshop we take those same hammers and apply them to whole genome sequences.

We will work with both the 1,000 Genomes reads and variants and also the Illumina Platinum Genomes gVCF variants.

We do this all from the comfort of the R prompt using common packages including VariantAnnotation, ggbio, ggplot2, dplyr, bigquery, and the new Bioconductor package GoogleGenomics which provides an R interface to Google’s implementation of the Global Alliance for Genomics and Health API.

And we’ll do this in a reproducible fashion running RMarkdown files via Dockerized Bioconductor running on Google Compute Engine VMs!

Contents

- BioC 2015: Where Software and Biology Connect
  - Get Started with Google Cloud Platform
    - Create a Google Cloud Platform project
    - Enable APIs
    - Install gcloud
  - Set up Bioconductor
  - Run the Codelabs
  - Looking for more to try?
  - Wrap up
    - “Stop” or “Delete” your VM
    - Stay Involved

Get Started with Google Cloud Platform

Create a Google Cloud Platform project
Details

If you already have a Google Cloud Platform project, this link will take you to your list of projects.

Sign up for Google Cloud Platform by clicking on this link: https://console.cloud.google.com/billing/freetrial

Enable APIs

Enable all the Google Cloud Platform APIs we will use in this workshop by clicking on this link.

Install gcloud

Follow the Windows, Mac OS X or Linux instructions to install gcloud on your local machine: https://cloud.google.com/sdk/

• Download and install the Google Cloud SDK by running this command in your shell or Terminal:

```
curl https://sdk.cloud.google.com | bash
```

Or, you can download google-cloud-sdk.zip or google-cloud-sdk.tar.gz, unpack it, and launch the /google-cloud-sdk/install.sh script.

Restart your shell or Terminal.

• Authenticate:

```
$ gcloud auth login
```

• Configure the project:

```
$ gcloud config set project <YOUR_PROJECT_ID>
```

Set up Bioconductor

Details

This will create a virtual machine on Google Cloud Platform with a locked down network (only SSH port 22 open). Your local machine will securely connect to the VM via an ssh tunnel.

Within the docker container the directory /home/rstudio/data will correspond to directory /mnt/data on the virtual machine. This is where the persistent data disk is attached to the VM. Store important files there. Docker containers are stateless, so if the container restarts for any reason, then files you created within the container will be lost.

To further the goals of reproducibility, ease of use, and convenience, you can run this codelab in a Bioconductor Docker container deployed to Google Compute Engine. But this codelab can be run from anywhere since all the heavy lifting is happening in the cloud regardless of where R is running.

Bioconductor maintains Docker containers with R, Bioconductor packages, and RStudio Server all ready to go! Its a great way to set up your R environment quickly and start working. The instructions are below but if you want to learn more, see http://www.bioconductor.org/help/docker/.

1. Click on click-to-deploy Bioconductor to navigate to the deployer page on the Cloud Platform Console.
2. In field Docker Image choose item custom.
3. Click on More to display the additional form fields.
4. In field Custom docker image paste in value gcr.io/bioc_2015/devel_sequencing.
5. Click on the Deploy Bioconductor button.
6. Follow the post-deployment instructions to log into RStudioServer via your browser!

If you prefer to run this docker container locally, click here to Show/Hide Instructions

To run the docker container locally:

1. Install Docker for your platform.
2. Run command docker run gcr.io/bioc_2015/devel_sequencing


Note that its big, over 4GB, since it is derived from the Bioconductor Sequencing view and contains many annotation databases.

If you prefer to setup R manually instead, click here to Show/Hide Instructions

# Install BiocInstaller.
source("http://bioconductor.org/biocLite.R")
# See http://www.bioconductor.org/developers/how-to/useDevel/
useDevel()
# Install devtools which is needed for the special use of biocLite() below.
biocLite("devtools")
# Install the workshop material.
biocLite("googlegenomics/bioconductor-workshop-r", build_vignettes=TRUE,
˓
˓→dependencies=TRUE)

Run the Codelabs

1. View the workshop documentation.

help(package="GoogleGenomicsBioc2015Workshop")

2. Click on “User guides, package vignettes and other documentation.”
3. Early on in the workshop you will need an API_KEY. You can get this by clicking on this link: https://console.cloud.google.com/project/_/apiui/credential
4. Click on vignette “BioC2015Workshop” and follow the instructions there to run the vignettes line-by-line or chunk-by-chunk!
   • To run line-by-line, put your cursor on the desired line and click the “Run” button or use keyboard shortcuts for Windows/Linux: Ctrl+Enter and Mac: Command+Enter.
   • To run chunk-by-chunk, put your cursor in the desired chunk and click the “Chunks -> Run Current Chuck” button. or use keyboard shortcuts for Windows/Linux: Ctrl+Alt+C and Mac: Command+Option+C.
If you just want to read the rendered results of the four codelabs, here they are:

- Working with Reads
- Working with Variants
- Analyzing Variants with BigQuery
- Data Analysis using Google Genomics, also available on YouTube:
  - brief video Google Genomics: Data Analysis Overview
  - extended video Google Genomics Codelab: Data Analysis in R

Looking for more to try?

- Try these samples on different datasets Discover Published Data.
- Find more example BigQuery queries in:
  - getting-started-bigquery
  - bigquery-examples
  - Quality Control using Google Genomics
- Run a Google Cloud Dataflow pipeline:
  - Compute Identity By State
  - Compute Principal Coordinate Analysis

Wrap up

“Stop” or “Delete” your VM

If you would like to pause your VM when not using it:

1. Go to the Google Cloud Platform Console and select your project: https://console.cloud.google.com/project/_/compute/instances
2. Click on the checkbox next to your VM.
3. Click on Stop to pause your VM.
4. When you are ready to use it again, Start your VM. For more detail, see: https://cloud.google.com/compute/docs/instances/stopping-or-deleting-an-instance

If you want to delete your deployment:

1. First copy any data off of the data disk that you wish to keep. The data disk will be deleted when the deployment is deleted.
2. Click on Deployments to navigate to your deployment and delete it.

Stay Involved

- Collaborate on the GA4GH specification.
- Subscribe to the mailing list.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Find Samples on GitHub

Here’s a list of what you’ll find on GitHub.

Interactive Data Analysis Samples and Tools

generating-started-bigquery Example queries to show how to get started with genomic data in BigQuery.
codelabs Codelabs demonstrating usage of several tools and systems on genomic data.
bigquery-examples Advanced BigQuery examples on genomic data.
datalab-examples Example Google Cloud Datalab iPython Notebooks for genomics use cases.
bioconductor-workshop-r R package containing instructional materials for using GoogleGenomics Bioconductor and bigquery packages.
api-client-r An R package for Google Genomics API queries.
gatk-tools-java Tools for using Picard and GATK with the Google Genomics API.
beacon-go AppEngine implementation of the Beacon API from the Global Alliance for Genomics and Health written in Go.

Cluster Computing Data Analysis Samples

dataflow-java Google Cloud Dataflow pipelines such as Identity-By-State as well as useful utility classes.
spark-examples Apache Spark jobs such as Principal Coordinate Analysis.
Working with the Google Genomics API

getting-started-with-the-api Examples of how to get started with the Google Genomics API in many languages.
utilis-java Common Java files for Google Genomics integrations.

Data Visualization Application Samples

api-client-javascript A simple web application that demonstrates how javascript can be used to fetch data from the Google Genomics APIs.
api-client-android A sample Android app that calls the Google Genomics API.
api-client-python Google AppEngine implementation of a simple genome browser that pulls data from the Google Genomics API.
api-client-r

An R package for Google Genomics API queries.
  • See the Shiny example.
  • See the ggbio example.

Data Analysis Application Samples

denovo-variant-caller-java A de novo variant caller which uses information from a mother, father and child trio with a Bayesian inference method.
linkage-disequilibrium A suite of Java tools to calculate linkage disequilibrium between variants and load the results into BigQuery and BigTable.

Miscellaneous

gce-images Scripts that create Google Compute Engine images and Docker containers with popular genomics software installed.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.

Subscribe to the mailing list

The Google Genomics Discuss mailing list is a good way to sync up with other people who use genomics-tools including the core developers. You can subscribe by sending an email to google-genomics-discuss+subscribe@googlegroups.com or just post using the web forum page.

Have feedback or corrections? All improvements to these docs are welcome! You can click on the “Edit on GitHub” link at the top right corner of this page or file an issue.

Need more help? Please see https://cloud.google.com/genomics/support.