Cumulus Documentation

Bo Li, Joshua Gould, and et al.

Sep 23, 2020

Contents

1 Version 1.0.0 *September 23, 2020* 3

2 Version 0.15.0 *May 6, 2020* 5

3 Version 0.14.0 *February 28, 2020* 7

4 Version 0.13.0 *February 7, 2020* 9

5 Version 0.12.0 *December 14, 2019* 11

6 Version 0.11.0 *December 4, 2019* 13

7 Version 0.10.0 *October 2, 2019* 15

8 Version 0.7.0 *Feburary 14, 2019* 17

9 Version 0.6.0 *January 31, 2019* 19

10 Version 0.5.0 *November 18, 2018* 21

11 Version 0.4.0 *October 26, 2018* 23

12 Version 0.3.0 *October 24, 2018* 25

13 Version 0.2.0 *October 19, 2018* 27

14 Version 0.1.0 *July 27, 2018* 29

All of our docker images are publicly available on Docker Hub and Quay. Our workflows use Docker Hub as the default Docker registry. Users can use Quay as the Docker registry by entering `quay.io/cumulus/` for the workflow input **"docker_registry"**, or enter a custom registry URL of their own choice.

If you use Cumulus in your research, please consider citing:

Li, B., Gould, J., Yang, Y. et al. Cumulus provides cloud-based data analysis for large-scale single-cell and single-nucleus RNA-seq. *Nat Methods* **17**, 793–798 (2020). https://doi.org/10.1038/s41592-020-0905-x

CHAPTER 1

Version 1.0.0 *September 23, 2020*

- Add *demultiplexing* workflow for cell-hashing/nucleus-hashing/genetic-pooling analysis.
- Add support on CellRanger version `4.0.0`.
- **Update *cumulus* workflow with Pegasus version `1.0.0`:**
  - Use `zarr` file format to handle data, which has a better I/O performance in general.
  - Support focus analysis on Unimodal data, and appending other Unimodal data to it. (`focus` and `append` inputs in *cluster* step).
  - Quality-Control: Change `percent_mito` default from `10.0` to `20.0`; by default remove bounds on UMIs (`min_umis` and `max_umis` inputs in *cluster* step).
  - Quality-Control: Automatically figure out name prefix of mitochondrial genes for `GRCh38` and `mm10` genome reference data.
  - Support signature / gene module score calculation. (`calc_signature_scores` input in *cluster* step)
  - Add *Scanorama* method to batch correction. (`correction_method` input in *cluster* step).
  - Calculate UMAP embedding by default, instead of FIt-SNE.
  - Differential Expression (DE) analysis: remove inputs `mwu` and `auc` as they are calculated by default. And cell-type annotation uses MWU test result by default.
- Remove *cumulus_subcluster* workflow.

CHAPTER 2

Version 0.15.0 *May 6, 2020*

- Update all workflows to OpenWDL version 1.0.
- Cumulus now supports multi-job execution from Terra data table input.
- Cumulus generates Cirrocumulus input in `.cirro` folder, instead of a huge `.parquet` file.

CHAPTER 3

Version 0.14.0 *February 28, 2020*

- Added support for gene-count matrices generation using alternative tools (STARsolo, Optimus, Salmon alevin, Kallisto BUStools).
- Cumulus can process demultiplexed data with remapped singlets names and subset of singlets.
- Update VDJ related inputs in Cellranger workflow.
- SMART-Seq2 and Count workflows are in OpenWDL version 1.0.

CHAPTER 4

Version 0.13.0 *February 7, 2020*

- Added support for aggregating scATAC-seq samples.
- Cumulus now accepts mtx format input.

CHAPTER 5

Version 0.12.0 *December 14, 2019*

- Added support for building references for sc/snRNA-seq, scATAC-seq, single-cell immune profiling, and SMART-Seq2 data.

CHAPTER 6

Version 0.11.0 *December 4, 2019*

- Reorganized Cumulus documentation.

CHAPTER 7

Version 0.10.0 *October 2, 2019*

- scCloud is renamed to Cumulus.
- Cumulus can accept either a sample sheet or a single file.

CHAPTER 8

Version 0.7.0 *Feburary 14, 2019*

- Added support for 10x genomics scATAC assays.
- scCloud runs FIt-SNE as default.

CHAPTER 9

Version 0.6.0 *January 31, 2019*

- Added support for 10x genomics V3 chemistry.
- Added support for extracting feature matrix for Perturb-Seq data.
- Added R script to convert output_name.seurat.h5ad to Seurat object. Now the raw.data slot stores filtered raw counts.
- Added min_umis and max_umis to filter cells based on UMI counts.
- Added QC plots and improved filtration spreadsheet.
- Added support for plotting UMAP and FLE.
- Now users can upload their JSON file to annotate cell types.
- Improved documentation.
- Added lightGBM based marker detection.

CHAPTER 10

Version 0.5.0 *November 18, 2018*

- Added support for plated-based SMART-Seq2 scRNA-Seq data.

CHAPTER 11

Version 0.4.0 *October 26, 2018*

- Added CITE-Seq module for analyzing CITE-Seq data.

CHAPTER 12

Version 0.3.0 *October 24, 2018*

- Added the demuxEM module for demultiplexing cell-hashing/nuclei-hashing data.

CHAPTER 13

Version 0.2.0 *October 19, 2018*

- Added support for V(D)J and CITE-Seq/cell-hashing/nuclei-hashing.

CHAPTER 14

Version 0.1.0 *July 27, 2018*

- KCO tools released!

14.1 First Time Running

14.1.1 Authenticate with Google

If you've done this before you can skip this step - you only need to do this once.

1. Ensure the Google Cloud SDK is installed on your computer.

   Note: Broad users do not have to install this-they can type:

   ```
   reuse Google-Cloud-SDK
   ```

   to make the Google Cloud tools available.

2. Execute the following command to login to Google Cloud.:

   ```
   gcloud auth login
   ```

3. Copy and paste the link in your unix terminal into your web browser.
4. Enter authorization code in unix terminal.

14.1.2 Create a Terra workspace

1. Create a new Terra workspace by clicking Create New Workspace in Terra

   For more detailed instructions please see this document.

14.2 Latest and stable versions on Terra

Cumulus is a fast growing project. As a result, we frequently update WDL snapshot versions on Terra. See below for latest and stable WDL versions you can use.

14.2.1 Stable version - v1.0.0

WDL	Snapshot	Function
cumulus/cellranger_workflow	12	Run Cell Ranger tools, which include extracting sequence reads using cellranger mkfastq or cellranger-atac mkfastq, generate count matrix using cellranger count or cellranger-atac count, run cellranger vdj or feature-barcode extraction
cumulus/count	14	Run alternative tools (STARsolo, Optimus, Salmon alevin, or Kallisto BUStools) to generate gene-count matrices from FASTQ files.
cumulus/demultiplexing	20	Run tools (demuxEM, souporcell, or demuxlet) for cell-hashing/nucleus-hashing/genetic-pooling analysis.
cumulus/cellranger_create_reference	9	Run Cell Ranger tools to build sc/snRNA-seq references.
cumulus/cellranger_atac_aggr	2	Run Cell Ranger tools to aggregate scATAC-seq samples.
cumulus/cellranger_atac_create_reference	2	Run Cell Ranger tools to build scATAC-seq references.
cumulus/cellranger_vdj_create_reference	3	Run Cell Ranger tools to build single-cell immune profiling references.
cumulus/smartseq2	7	Run HISAT2/STAR/Bowtie2-RSEM to generate gene-count matrices for SMART-Seq2 data from FASTQ files
cumulus/smartseq2_create_reference	8	Generate user-customized genome references for SMART-Seq2 data.
cumulus/cumulus	31	Run cumulus analysis module for variable gene selection, batch correction, PCA, diffusion map, clustering, visualization, differential expression analysis, cell type annotation, etc.
cumulus/cumulus_hashing_cite_seq	10	Run cumulus for cell-hashing/nucleus-hashing/CITE-Seq analysis

14.2.2 Stable version - v0.15.0

WDL	Snapshot	Function
cumulus/cellranger_workflow	10	Run Cell Ranger tools, which include extracting sequence reads using cellranger mkfastq or cellranger-atac mkfastq, generate count matrix using cellranger count or cellranger-atac count, run cellranger vdj or feature-barcode extraction
cumulus/count	14	Run alternative tools (STARsolo, Optimus, Salmon alevin, or Kallisto BUStools) to generate gene-count matrices from FASTQ files.
cumulus/cellranger_create_reference	8	Run Cell Ranger tools to build sc/snRNA-seq references.
cumulus/cellranger_atac_aggr	2	Run Cell Ranger tools to aggregate scATAC-seq samples.
cumulus/cellranger_atac_create_reference	2	Run Cell Ranger tools to build scATAC-seq references.
cumulus/cellranger_vdj_create_reference	9	Run Cell Ranger tools to build single-cell immune profiling references.
cumulus/smartseq2	7	Run HISAT2/STAR/Bowtie2-RSEM to generate gene-count matrices for SMART-Seq2 data from FASTQ files
cumulus/smartseq2_create_reference	8	Generate user-customized genome references for SMART-Seq2 data.
cumulus/cumulus	24	Run cumulus analysis module for variable gene selection, batch correction, PCA, diffusion map, clustering, visualization, differential expression analysis, cell type annotation, etc.
cumulus/cumulus_subcluster	16	Run subcluster analysis using cumulus
cumulus/cumulus_hashing_cite_seq	10	Run cumulus for cell-hashing/nucleus-hashing/CITE-Seq analysis

14.2.3 Stable version - v0.14.0

WDL	Snapshot	Function
cumulus/cellranger_workflow	8	Run Cell Ranger tools, which include extracting sequence reads using cellranger mkfastq or cellranger-atac mkfastq, generate count matrix using cellranger count or cellranger-atac count, run cellranger vdj or feature-barcode extraction
cumulus/count	11	Run alternative tools (STARsolo, Optimus, Salmon alevin, or Kallisto BUStools) to generate gene-count matrices from FASTQ files.
cumulus/cellranger_create_reference	6	Run Cell Ranger tools to build sc/snRNA-seq references.
cumulus/cellranger_atac_aggr	1	Run Cell Ranger tools to aggregate scATAC-seq samples.
cumulus/cellranger_atac_create_reference	1	Run Cell Ranger tools to build scATAC-seq references.
cumulus/cellranger_vdj_create_reference	1	Run Cell Ranger tools to build single-cell immune profiling references.
cumulus/smartseq2	7	Run HISAT2/STAR/Bowtie2-RSEM to generate gene-count matrices for SMART-Seq2 data from FASTQ files
cumulus/smartseq2_create_reference	8	Generate user-customized genome references for SMART-Seq2 data.
cumulus/cumulus	16	Run cumulus analysis module for variable gene selection, batch correction, PCA, diffusion map, clustering, visualization, differential expression analysis, cell type annotation, etc.
cumulus/cumulus_subcluster	10	Run subcluster analysis using cumulus
cumulus/cumulus_hashing_cite_seq	8	Run cumulus for cell-hashing/nucleus-hashing/CITE-Seq analysis

14.2.4 Stable version - v0.13.0

WDL	Snapshot	Function
cumulus/cellranger_workflow	7	Run Cell Ranger tools, which include extracting sequence reads using cellranger mkfastq or cellranger-atac mkfastq, generate count matrix using cellranger count or cellranger-atac count, run cellranger vdj or feature-barcode extraction
cumulus/cellranger_create_reference	1	Run Cell Ranger tools to build sc/snRNA-seq references.
cumulus/cellranger_atac_aggr	1	Run Cell Ranger tools to aggregate scATAC-seq samples.
cumulus/cellranger_atac_create_reference	1	Run Cell Ranger tools to build scATAC-seq references.
cumulus/cellranger_vdj_create_reference	1	Run Cell Ranger tools to build single-cell immune profiling references.
cumulus/smartseq2	5	Run Bowtie2 and RSEM to generate gene-count matrices for SMART-Seq2 data from FASTQ files
cumulus/smartseq2_create_reference	1	Generate user-customized genome references for SMART-Seq2 data.
cumulus/cumulus	14	Run cumulus analysis module for variable gene selection, batch correction, PCA, diffusion map, clustering, visualization, differential expression analysis, cell type annotation, etc.
cumulus/cumulus_subcluster	9	Run subcluster analysis using cumulus
cumulus/cumulus_hashing_cite_seq	7	Run cumulus for cell-hashing/nucleus-hashing/CITE-Seq analysis

14.2.5 Stable version - v0.12.0

WDL	Snapshot	Function
cumulus/cellranger_workflow	6	Run Cell Ranger tools, which include extracting sequence reads using cellranger mkfastq or cellranger-atac mkfastq, generate count matrix using cellranger count or cellranger-atac count, run cellranger vdj or feature-barcode extraction
cumulus/cellranger_create_reference	1	Run Cell Ranger tools to build sc/snRNA-seq references.
cumulus/cellranger_atac_create_reference	1	Run Cell Ranger tools to build scATAC-seq references.
cumulus/cellranger_vdj_create_reference	1	Run Cell Ranger tools to build single-cell immune profiling references.
cumulus/smartseq2	5	Run Bowtie2 and RSEM to generate gene-count matrices for SMART-Seq2 data from FASTQ files
cumulus/smartseq2_create_reference	1	Generate user-customized genome references for SMART-Seq2 workflow.
cumulus/cumulus	11	Run cumulus analysis module for variable gene selection, batch correction, PCA, diffusion map, clustering, visualization, differential expression analysis, cell type annotation, etc.
cumulus/cumulus_subcluster	8	Run subcluster analysis using cumulus
cumulus/cumulus_hashing_cite_seq	6	Run cumulus for cell-hashing/nucleus-hashing/CITE-Seq analysis

14.2.6 Stable version - v0.11.0

WDL	Snapshot	Function
cumulus/cellranger_workflow	4	Run Cell Ranger tools, which include extracting sequence reads using cellranger mkfastq or cellranger-atac mkfastq, generate count matrix using cellranger count or cellranger-atac count, run cellranger vdj or feature-barcode extraction
cumulus/smartseq2	3	Run Bowtie2 and RSEM to generate gene-count matrices for SMART-Seq2 data from FASTQ files
cumulus/cumulus	8	Run cumulus analysis module for variable gene selection, batch correction, PCA, diffusion map, clustering, visualization, differential expression analysis, cell type annotation, etc.
cumulus/cumulus_subcluster	5	Run subcluster analysis using cumulus
cumulus/cumulus_hashing_cite_seq	5	Run cumulus for cell-hashing/nucleus-hashing/CITE-Seq analysis

14.2.7 Stable version - v0.10.0

WDL	Snapshot	Function
cumulus/cellranger_workflow	3	Run Cell Ranger tools, which include extracting sequence reads using cellranger mkfastq or cellranger-atac mkfastq, generate count matrix using cellranger count or cellranger-atac count, run cellranger vdj or feature-barcode extraction
cumulus/smartseq2	3	Run Bowtie2 and RSEM to generate gene-count matrices for SMART-Seq2 data from FASTQ files
cumulus/cumulus	7	Run cumulus analysis module for variable gene selection, batch correction, PCA, diffusion map, clustering, visualization, differential expression analysis, cell type annotation, etc.
cumulus/cumulus_subcluster	4	Run subcluster analysis using cumulus
cumulus/cumulus_hashing_cite_seq	4	Run cumulus for cell-hashing/nucleus-hashing/CITE-Seq analysis

14.2.8 Stable version - HTAPP v2

WDL	Snapshot	Function
regev/cellranger_mkfastq_count	45	Run Cell Ranger to extract FASTQ files and generate gene-count matrices for 10x genomics data
scCloud/smartseq2	5	Run Bowtie2 and RSEM to generate gene-count matrices for SMART-Seq2 data from FASTQ files
scCloud/scCloud	14	Run scCloud analysis module for variable gene selection, batch correction, PCA, diffusion map, clustering and more
scCloud/scCloud_subcluster	9	Run subcluster analysis using scCloud
scCloud/scCloud_hashing_cite_seq	9	Run scCloud for cell-hashing/nucleus-hashing/CITE-Seq analysis

14.2.9 Stable version - HTAPP v1

WDL	Snapshot	Function
regev/cellranger_mkfastq_count	39	Run Cell Ranger to extract FASTQ files and generate gene-count matrices for 10x genomics data
scCloud/scCloud	3	Run scCloud analysis module for variable gene selection, batch correction, PCA, diffusion map, clustering and more

14.3 Run Cell Ranger tools using cellranger_workflow

`cellranger_workflow` wraps Cell Ranger to process single-cell/nucleus RNA-seq, single-cell ATAC-seq and single-cell immune profiling data, and supports feature barcoding (cell/nucleus hashing, CITE-seq, Perturb-seq). It also provide routines to build cellranger references.

14.3.1 A general step-by-step instruction

This section mainly considers jobs starting from BCL files. If your job starts with FASTQ files, and only need to run `cellranger count` part, please refer to this subsection.

1. Import `cellranger_workflow`

Import *cellranger_workflow* workflow to your workspace.

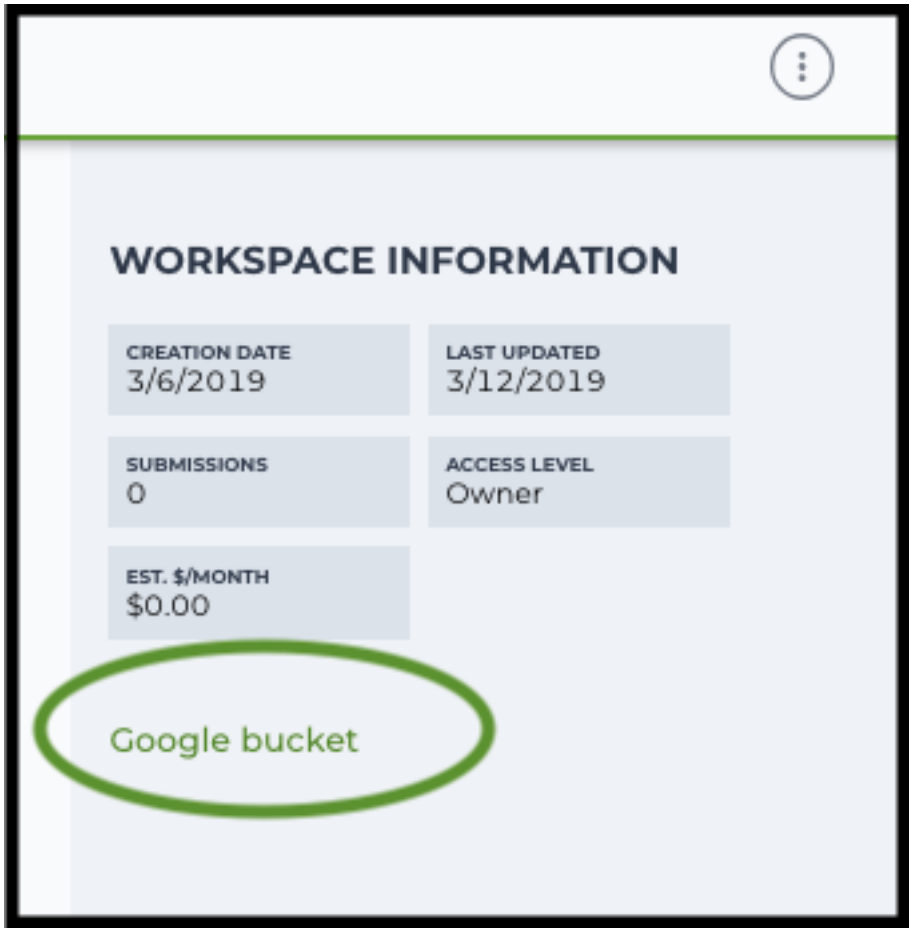
See the Terra documentation for adding a workflow. The *cellranger_workflow* workflow is under `Broad Methods Repository` with name "**cumulus/cellranger_workflow**".

Moreover, in the workflow page, click the `Export to Workspace...` button, and select the workspace to which you want to export *cellranger_workflow* workflow in the drop-down menu.

2. Upload sequencing data to Google bucket

Copy your sequencing output to your workspace bucket using gsutil (you already have it if you've installed Google cloud SDK) in your unix terminal.

You can obtain your bucket URL in the dashboard tab of your Terra workspace under the information panel.



Use `gsutil cp [OPTION]... src_url dst_url` to copy data to your workspace bucket. For example, the following command copies the directory at /foo/bar/nextseq/Data/VK18WBC6Z4 to a Google bucket:

```
gsutil -m cp -r /foo/bar/nextseq/Data/VK18WBC6Z4 gs://fc-e0000000-0000-0000-
↪0000-000000000000/VK18WBC6Z4
```

`-m` means copy in parallel, `-r` means copy the directory recursively, and `gs://fc-e0000000-0000-0000-0000-000000000000` should be replaced by your own workspace Google bucket URL.

Note: If input is a folder of BCL files, users do not need to upload the whole folder to the Google bucket. Instead, they only need to upload the following files:

```
RunInfo.xml
RTAComplete.txt
runParameters.xml
Data/Intensities/s.locs
Data/Intensities/BaseCalls
```

If data are generated using MiSeq or NextSeq, the location files are inside lane subfloders `L001` under `Data/Intensities/`. In addition, if users' data only come from a subset of lanes (e.g. `L001` and `L002`), users only need to upload lane subfolders from the subset (e.g. `Data/Intensities/BaseCalls/L001`, `Data/Intensities/BaseCalls/L002` and `Data/Intensities/L001`, `Data/Intensities/L002` if se-

quencer is MiSeq or NextSeq).

Alternatively, users can submit jobs through command line interface (CLI) using altocumulus, which will smartly upload BCL folders according to the above rules.

Note: Broad users need to be on an UGER node (not a login node) in order to use the `-m` flag

Request an UGER node:

```
reuse UGER
qrsh -q interactive -l h_vmem=4g -pe smp 8 -binding linear:8 -P regevlab
```

The above command requests an interactive node with 4G memory per thread and 8 threads. Feel free to change the memory, thread, and project parameters.

Once you're connected to an UGER node, you can make gsutil available by running:

```
reuse Google-Cloud-SDK
```

3. Prepare a sample sheet

3.1 Sample sheet format:

Please note that the columns in the CSV can be in any order, but that the column names must match the recognized headings.

The sample sheet describes how to demultiplex flowcells and generate channel-specific count matrices. Note that *Sample*, *Lane*, and *Index* columns are defined exactly the same as in 10x's simple CSV layout file.

A brief description of the sample sheet format is listed below **(required column headers are shown in bold)**.

Column	Description
Sample	Contains sample names. Each 10x channel should have a unique sample name.
Reference	Provides the reference genome used by Cell Ranger for each 10x channel. The elements in the *reference* column can be either Google bucket URLs to reference tarballs or keywords such as *GRCh38_v3.0.0*. A full list of available keywords is included in each of the following data type sections (e.g. sc/snRNA-seq) below.
Flowcell	Indicates the Google bucket URLs of uploaded BCL folders. If starts with FASTQ files, this should be Google bucekt URLs of uploaded FASTQ folders. The FASTQ folders should contain one subfolder for each sample in the flowcell with the sample name as the subfolder name. Each subfolder contains FASTQ files for that sample.
Lane	Tells which lanes the sample was pooled into. Can be either single lane (e.g. 8) or a range (e.g. 7-8) or all (e.g. *).
Index	Sample index (e.g. SI-GA-A12).
Chemistry	Describes the 10x chemistry used for the sample. This column is optional.
DataType	Describes the data type of the sample — *rna*, *vdj*, *adt*, or *crispr*. **rna** refers to gene expression data (*cellranger count*), **vdj** refers to V(D)J data (*cellranger vdj*), **adt** refers to antibody tag data, which can be either CITE-Seq, cell-hashing, or nucleus-hashing, **crispr** refers to Perturb-seq guide tag data, **atac** refers to scATAC-Seq data (*cellranger-atac count*). This column is optional and the default data type is *rna*.
FeatureBarcodeFile	Google bucket urls pointing to feature barcode files for *adt* and *crispr* data. Features can be either antibody for CITE-Seq, cell-hashing, nucleus-hashing or gRNA for Perburb-seq. This column is optional provided no *adt* or *crispr* data are in the sample sheet.

The sample sheet supports sequencing the same 10x channels across multiple flowcells. If a sample is sequenced across multiple flowcells, simply list it in multiple rows, with one flowcell per row. In the following example, we have 4 samples sequenced in two flowcells.

Example:

```
Sample,Reference,Flowcell,Lane,Index,Chemistry,DataType,FeatureBarcodeFile
sample_1,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK18WBC6Z4,1-2,SI-GA-A8,threeprime,rna
```

(continues on next page)

(continued from previous page)

```
sample_2,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK18WBC6Z4,3-4,SI-GA-B8,SC3Pv3,rna
sample_3,mm10_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4,
↪5-6,SI-GA-C8,fiveprime,rna
sample_4,mm10_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4,
↪7-8,SI-GA-D8,fiveprime,rna
sample_1,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK10WBC9Z2,1-2,SI-GA-A8,threeprime,rna
sample_2,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK10WBC9Z2,3-4,SI-GA-B8,SC3Pv3,rna
sample_3,mm10_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2,
↪5-6,SI-GA-C8,fiveprime,rna
sample_4,mm10_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2,
↪7-8,SI-GA-D8,fiveprime,rna
```

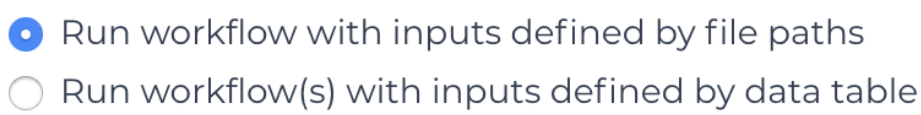
3.2 Upload your sample sheet to the workspace bucket:

Example:

```
gsutil cp /foo/bar/projects/sample_sheet.csv gs://fc-e0000000-0000-
↪0000-0000-000000000000/
```

4. Launch analysis

In your workspace, open `cellranger_workflow` in `WORKFLOWS` tab. Select the desired snapshot version (e.g. latest). Select `Run workflow with inputs defined by file paths` as below



and click `SAVE` button. Select `Use call caching` and click `INPUTS`. Then fill in appropriate values in the `Attribute` column. Alternative, you can upload a JSON file to configure input by clicking `Drag or click to upload json`.

Once INPUTS are appropriated filled, click `RUN ANALYSIS` and then click `LAUNCH`.

5. Notice: run `cellranger mkfastq` if you are non Broad Institute users

Non Broad Institute users that wish to run `cellranger mkfastq` must create a custom docker image that contains `bcl2fastq`.

See *bcl2fastq* instructions.

6. Run `cellranger count` only

Sometimes, users might want to perform demultiplexing locally and only run the count part on the cloud. This section describes how to only run the count part via `cellranger_workflow`.

1. Copy your FASTQ files to the workspace using gsutil in your unix terminal.

You should upload folders of FASTQ files. The uploaded folder (for one flowcell) should contain one subfolder for each sample belong to the this flowcell. **In addition, the subfolder name and the sample name in your sample sheet MUST be the same.** Each subfolder contains FASTQ files for that sample.

Example:

```
gsutil -m cp -r /foo/bar/fastq_path/K18WBC6Z4 gs://fc-e0000000-0000-0000-
↪0000-000000000000/K18WBC6Z4_fastq
```

2. Create a sample sheet following the similar structure as above, except the following differences:

   - **Flowcell** column should list Google bucket URLs of the FASTQ folders for flowcells.
   - **Lane** and **Index** columns are NOT required in this case.

   Example:

```
Sample,Reference,Flowcell
sample_1,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪K18WBC6Z4_fastq
```

3. Set optional input `run_mkfastq` to `false`.

14.3.2 Single-cell and single-nucleus RNA-seq

To process sc/snRNA-seq data, follow the specific instructions below.

Sample sheet

1. **Reference** column.

   Pre-built scRNA-seq references are summarized below.

Keyword	Description
GRCh38-2020-A	Human GRCh38 (GENCODE v32/Ensembl 98)
mm10-2020-A	Mouse mm10 (GENCODE vM23/Ensembl 98)
GRCh38_and_mm10-2020-A	Human GRCh38 (GENCODE v32/Ensembl 98) and mouse mm10 (GENCODE vM23/Ensembl 98)
GRCh38_v3.0.0	Human GRCh38, cellranger reference 3.0.0, Ensembl v93 gene annotation
hg19_v3.0.0	Human hg19, cellranger reference 3.0.0, Ensembl v87 gene annotation
mm10_v3.0.0	Mouse mm10, cellranger reference 3.0.0, Ensembl v93 gene annotation
GRCh38_and_mm10_v3.1.0	Human (GRCh38) and mouse (mm10), cellranger references 3.1.0, Ensembl v93 gene annotations for both human and mouse
hg19_and_mm10_v3.0.0	Human (hg19) and mouse (mm10), cellranger reference 3.0.0, Ensembl v93 gene annotations for both human and mouse
GRCh38_v1.2.0 or **GRCh38**	Human GRCh38, cellranger reference 1.2.0, Ensembl v84 gene annotation
hg19_v1.2.0 or **hg19**	Human hg19, cellranger reference 1.2.0, Ensembl v82 gene annotation
mm10_v1.2.0 or **mm10**	Mouse mm10, cellranger reference 1.2.0, Ensembl v84 gene annotation
GRCh38_and_mm10_v1.2.0 or **GRCh38_and_mm10**	Human and mouse, built from GRCh38 and mm10 cellranger references, Ensembl v84 gene annotations are used
GRCh38_and_SARSCoV2	Human GRCh38 and SARS-COV-2 RNA genome, cellranger reference 3.0.0, generated by Carly Ziegler. The SARS-COV-2 viral sequence and gtf are as described in [Kim et al. Cell 2020] (https://github.com/hyeshik/sars-cov-2-transcriptome, BetaCov/South Korea/KCDC03/2020 based on NC_045512.2). The GTF was edited to include only CDS regions, and regions were added to describe the 5' UTR ("SARSCoV2_5prime"), the 3' UTR ("SARSCoV2_3prime"), and reads aligning to anywhere within the Negative Strand("SARSCoV2_NegStrand"). Additionally, trailing A's at the 3' end of the virus were excluded from the SARSCoV2 fasta, as these were found to drive spurious viral alignment in pre-COVID19 samples.

Pre-built snRNA-seq references are summarized below.

Keyword	Description
GRCh38_premrna_v3.0.0	Human, introns included, built from GRCh38 cellranger reference 3.0.0, Ensembl v93 gene annotation, treating annotated transcripts as exons
GRCh38_premrna_v1.2.0 or **GRCh38_premrna**	Human, introns included, built from GRCh38 cellranger reference 1.2.0, Ensembl v84 gene annotation, treating annotated transcripts as exons
mm10_premrna_v1.2.0 or **mm10_premrna**	Mouse, introns included, built from mm10 cellranger reference 1.2.0, Ensembl v84 gene annotation, treating annotated transcripts as exons
GRCh38_premrna_and_mm10_premrna_v1.2.0 or **GRCh38_premrna_and_mm10_premrna**	Human and mouse, introns included, built from GRCh38_premrna_v1.2.0 and mm10_premrna_v1.2.0
GRCh38_premrna_and_SARSCoV2	Human, introns included, built from GRCh38_premrna_v3.0.0, and SARS-COV-2 RNA genome. This reference was generated by Carly Ziegler. The SARS-COV-2 RNA genome is from [Kim et al. Cell 2020] (https://github.com/hyeshik/sars-cov-2-transcriptome, BetaCov/South Korea/KCDC03/2020 based on NC_045512.2). Please see the description of *GRCh38_and_SARSCoV2* above for details.

2. **Index** column.

   Put 10x single cell 3' sample index set names (e.g. SI-GA-A12) here.

3. *Chemistry* column.

   According to *cellranger count*'s documentation, chemistry can be

Chemistry	Explanation
auto	autodetection (default). If the index read has extra bases besides cell barcode and UMI, autodetection might fail. In this case, please specify the chemistry
threeprime	Single Cell 3
fiveprime	Single Cell 5
SC3Pv1	Single Cell 3 v1
SC3Pv2	Single Cell 3 v2
SC3Pv3	Single Cell 3 v3. You should set cellranger version input parameter to >= 3.0.2
SC5P-PE	Single Cell 5 paired-end (both R1 and R2 are used for alignment)
SC5P-R2	Single Cell 5 R2-only (where only R2 is used for alignment)

4. *DataType* column.

   This column is optional with a default **rna**. If you want to put a value, put **rna** here.

5. *FetureBarcodeFile* column.

   Leave it blank for scRNA-seq and snRNA-seq.

6. Example:

```
Sample,Reference,Flowcell,Lane,Index,Chemistry,DataType,FeatureBarcodeFile
sample_1,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4,1-
↪2,SI-GA-A8,threeprime,rna
sample_2,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4,3-
↪4,SI-GA-B8,SC3Pv3,rna
sample_3,mm10_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4,5-6,
↪SI-GA-C8,fiveprime,rna
```

(continues on next page)

(continued from previous page)

```
sample_4,mm10_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4,7-8,
↪SI-GA-D8,fiveprime,rna
sample_1,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2,1-
↪2,SI-GA-A8,threeprime,rna
sample_2,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2,3-
↪4,SI-GA-B8,SC3Pv3,rna
sample_3,mm10_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2,5-6,
↪SI-GA-C8,fiveprime,rna
sample_4,mm10_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2,7-8,
↪SI-GA-D8,fiveprime,rna
```

Workflow input

For sc/snRNA-seq data, `cellranger_workflow` takes Illumina outputs as input and runs `cellranger mkfastq` and `cellranger count`. Revalant workflow inputs are described below, with required inputs highlighted in bold.

Name	Description	Example	Default
input_sample_file	Sample Sheet (contains Sample, Reference, Flowcell, Lane, Index as required and Chemistry, DataType, FeatureBarcodeFile as optional)	"gs://fc-e0000000-0000-0000-0000-000000000000/sample_sheet.csv"	
output_directory	Output directory	"gs://fc-e0000000-0000-0000-0000-000000000000/cellranger_output"	Results are written to $output_directory/$bcl_directory_fastqs/fastq_path/ and will overwrite any existing files at this location.
run_mkfastq	If you want to run `cellranger mkfastq`	true	true
run_count	If you want to run `cellranger count`	true	true
delete_input_bcl_dir	If delete BCL directories after demux. If false, you should delete this folder yourself so as to not incur storage charges	false	false
mkfastq_barcode_mismatches	Number of mismatches allowed in matching barcode indices (bcl2fastq2 default is 1)	0	
force_cells	Force pipeline to use this number of cells, bypassing the cell detection algorithm, mutually exclusive with expect_cells	6000	
expect_cells	Expected number of recovered cells. Mutually exclusive with force_cells	3000	
secondary	Perform Cell Ranger secondary analysis (dimensionality reduction, clustering, etc.)	false	false
cellranger_version	cellranger version, could be 4.0.0, 3.1.0, 3.0.2, or 2.2.0	"4.0.0"	"4.0.0"
docker_registry	Docker registry to use for cellranger_workflow. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"
cellranger_mkfastq_docker_registry	Docker registry for `cellranger mkfastq`. Default is the registry to which only Broad users have access. See *bcl2fastq* for making your own registry.	"gcr.io/broad-cumulus"	"gcr.io/broad-cumulus"
zones	Google cloud zones	"us-central1-a us-west1-a"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
num_cpu	Number of cpus to request for one node for cellranger mkfastq and cellranger count	32	32

Workflow output

See the table below for important sc/snRNA-seq outputs.

Name	Type	Description
output_fastqs_directory	Array[String]	A list of google bucket urls containing FASTQ files, one url per flowcell.
output_count_directory	Array[String]	A list of google bucket urls containing count matrices, one url per sample.
metrics_summaries	File	A excel spreadsheet containing QCs for each sample.
output_web_summary	Array[File]	A list of htmls visualizing QCs for each sample (cellranger count output).
count_matrix	String	gs url for a template count_matrix.csv to run Cumulus.

14.3.3 Feature barcoding assays (cell & nucleus hashing, CITE-seq and Perturb-seq)

`cellranger_workflow` can extract feature-barcode count matrices in CSV format for feature barcoding assays such as *cell and nucleus hashing*, *CITE-seq*, and *Perturb-seq*. For cell and nucleus hashing as well as CITE-seq, the feature refers to antibody. For Perturb-seq, the feature refers to guide RNA. Please follow the instructions below to configure `cellranger_workflow`.

Prepare feature barcode files

Prepare a CSV file with the following format: feature_barcode,feature_name. See below for an example:

```
TTCCTGCCATTACTA,sample_1
CCGTACCTCATTGTT,sample_2
GGTAGATGTCCTCAG,sample_3
TGGTGTCATTCTTGA,sample_4
```

The above file describes a cell hashing application with 4 samples.

If cell hashing and CITE-seq data share a same sample index, you should concatenate hashing and CITE-seq barcodes together and add a third column indicating the feature type. See below for an example:

```
TTCCTGCCATTACTA,sample_1,hashing
CCGTACCTCATTGTT,sample_2,hashing
GGTAGATGTCCTCAG,sample_3,hashing
TGGTGTCATTCTTGA,sample_4,hashing
CTCATTGTAACTCCT,CD3,citeseq
GCGCAACTTGATGAT,CD8,citeseq
```

Then upload it to your google bucket:

```
gsutil antibody_index.csv gs://fc-e0000000-0000-0000-0000-000000000000/
↪antibody_index.csv
```

Sample sheet

1. **Reference** column.

   This column is not used for extracting feature-barcode count matrix. To be consistent, please put the reference for the associated scRNA-seq assay here.

2. **Index** column.

   The ADT/HTO index can be either Illumina index primer sequence (e.g. `ATTACTCG`, also known as `D701`), or 10x single cell 3' sample index set names (e.g. SI-GA-A12).

   Note 1: All ADT/HTO index sequences (including 10x's) should have the same length (8 bases). If one index sequence is shorter (e.g. ATCACG), pad it with P7 sequence (e.g. ATCACGAT).

   Note 2: It is users' responsibility to avoid index collision between 10x genomics' RNA indexes (e.g. SI-GA-A8) and Illumina index sequences for used here (e.g. `ATTACTCG`).

   Note 3: For NextSeq runs, please reverse complement the ADT/HTO index primer sequence (e.g. use reverse complement `CGAGTAAT` instead of `ATTACTCG`).

3. *Chemistry* column.

   The following keywords are accepted for *Chemistry* column:

Chemistry	Explanation
SC3Pv3	Single Cell 3 v3 (default).
SC3Pv2	Single Cell 3 v2
fiveprime	Single Cell 5
SC5P-PE	Single Cell 5 paired-end (both R1 and R2 are used for alignment)
SC5P-R2	Single Cell 5 R2-only (where only R2 is used for alignment)

4. *DataType* column.

   Put **adt** here if the assay is CITE-seq, cell or nucleus hashing. Put **crispr** here if Perturb-seq.

5. *FetureBarcodeFile* column.

   Put Google Bucket URL of the feature barcode file here.

6. Example:

```
Sample,Reference,Flowcell,Lane,Index,Chemistry,DataType,FeatureBarcodeFile
sample_1_rna,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK18WBC6Z4,1-2,SI-GA-A8,threeprime,rna
sample_1_adt,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK18WBC6Z4,1-2,ATTACTCG,threeprime,adt,gs://fc-e0000000-0000-0000-0000-
↪000000000000/antibody_index.csv
sample_2_adt,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK18WBC6Z4,3-4,TCCGGAGA,SC3Pv3,adt,gs://fc-e0000000-0000-0000-0000-000000000000/
↪antibody_index.csv
sample_3_crispr,GRCh38_v3.0.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK18WBC6Z4,5-6,CGCTCATT,SC3Pv3,crispr,gs://fc-e0000000-0000-0000-0000-
↪000000000000/crispr_index.csv
```

In the sample sheet above, despite the header row,

- First row describes the normal 3' RNA assay;
- Second row describes its associated antibody tag data, which can from either a CITE-seq, cell hashing, or nucleus hashing experiment.

- Third row describes another tag data, which is in 10x genomics' V3 chemistry. For tag and crispr data, it is important to explicitly state the chemistry (e.g. `SC3Pv3`).
- Last row describes one gRNA guide data for Perturb-seq (see `crispr` in *DataType* field).

Workflow input

For feature barcoding data, `cellranger_workflow` takes Illumina outputs as input and runs `cellranger mkfastq` and `cumulus adt`. Revalant workflow inputs are described below, with required inputs highlighted in bold.

Name	Description	Example	Default
input_csv_file	Sample Sheet (contains Sample, Reference, Flowcell, Lane, Index as required and Chemistry, DataType, FeatureBarcodeFile as optional)	"gs://fc-e0000000-0000-0000-0000-000000000000/sample_sheet.csv"	
output_directory	Output directory	"gs://fc-e0000000-0000-0000-0000-000000000000/cellranger_output"	
run_mkfastq	If you want to run `cellranger mkfastq`	true	true
delete_input_bcl_dir	If delete BCL directories after demux. If false, you should delete this folder yourself so as to not incur storage charges	false	false
mkfastq_number_of_mismatches	Number of mismatches allowed in matching barcode indices (bcl2fastq2 default is 1)	0	
scaffold_sequence	Scaffold sequence in sgRNA for Purturb-seq, only used for crispr data type. If it is "", we assume guide barcode starts at position 0 of read 2	"GTTTAAGAGCTAAGCTGGAA"	""
max_mismatch	Maximum hamming distance in feature barcodes for the adt task	3	3
min_read_ratio	Minimum read count ratio (non-inclusive) to justify a feature given a cell barcode and feature combination, only used for the adt task and crispr data type	0.1	0.1
cellranger_version	cellranger version, could be 4.0.0, 3.1.0, 3.0.2, 2.2.0	"4.0.0"	"4.0.0"
cumulus_feature_barcoding_version	Cumulus feature barcoding version for extracting feature barcode matrix. Version available: 0.3.0, 0.2.0.	"0.3.0"	"0.3.0"
docker_registry	Docker registry to use for cellranger_workflow. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"
mkfastq_docker_registry	Docker registry to use for `cellranger mkfastq`. Default is the registry to which only Broad users have access. See *bcl2fastq* for making your own registry.	"gcr.io/broad-cumulus"	"gcr.io/broad-cumulus"
zones	Google cloud zones	"us-central1-a us-west1-a"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
num_cpu	Number of cpus to request for one node for cellranger mkfastq	32	32

Parameters used for feature count matrix extraction

If the chemistry is V2, 10x genomics v2 cell barcode white list will be used, a hamming distance of 1 is allowed for matching cell barcodes, and the UMI length is 10. If the chemistry is V3, 10x genomics v3 cell barcode white list will be used, a hamming distance of 0 is allowed for matching cell barcodes, and the UMI length is 12.

For Perturb-seq data, a small number of sgRNA protospace sequences will be sequenced ultra-deeply and we may have PCR chimeric reads. Therefore, we generate filtered feature count matrices as well in a data driven manner:

1. First, plot the histogram of UMIs with certain number of read counts. The number of UMIs with `x` supporting reads decreases when `x` increases. We start from `x = 1`, and a valley between two peaks is detected if we find `count[x] < count[x + 1] < count[x + 2]`. We filter out all UMIs with `< x` supporting reads since they are likely formed due to chimeric reads.
2. In addition, we also filter out barcode-feature-UMI combinations that have their read count ratio, which is defined as total reads supporting barcode-feature-UMI over total reads supporting barcode-UMI, no larger than `min_read_ratio` parameter set above.

Workflow outputs

See the table below for important outputs.

Name	Type	Description
output_fastqs_directory	Array[String]	A list of google bucket urls containing FASTQ files, one url per flowcell.
output_count_directory	Array[String]	A list of google bucket urls containing feature-barcode count matrices, one url per sample.
count_matrix	String	gs url for a template count_matrix.csv to run cumulus.

In addition, For each antibody tag or crispr tag sample, a folder with the sample ID is generated under `output_directory`. In the folder, two files — `sample_id.csv` and `sample_id.stat.csv.gz` — are generated.

`sample_id.csv` is the feature count matrix. It has the following format. The first line describes the column names: `Antibody/CRISPR,cell_barcode_1,cell_barcode_2,...,cell_barcode_n`. The following lines describe UMI counts for each feature barcode, with the following format: `feature_name,umi_count_1,umi_count_2,...,umi_count_n`.

`sample_id.stat.csv.gz` stores the gzipped sufficient statistics. It has the following format. The first line describes the column names: `Barcode,UMI,Feature,Count`. The following lines describe the read counts for every barcode-umi-feature combination.

If the feature barcode file has a third column, there will be two files for each feature type in the third column. For example, if `hashing` presents, `sample_id.hashing.csv` and `sample_id.hashing.stat.csv.gz` will be generated.

If data type is `crispr`, three additional files, `sample_id.umi_count.pdf`, `sample_id.filt.csv` and `sample_id.filt.stat.csv.gz`, are generated.

`sample_id.umi_count.pdf` plots number of UMIs against UMI with certain number of reads and colors UMIs with high likelihood of being chimeric in blue and other UMIs in red. This plot is generated purely based on number of reads each UMI has.

`sample_id.filt.csv` is the filtered feature count matrix. It has the same format as `sample_id.csv`.

`sample_id.filt.stat.csv.gz` is the filtered sufficient statistics. It has the same format as `sample_id.stat.csv.gz`.

14.3.4 Single-cell ATAC-seq

To process scATAC-seq data, follow the specific instructions below.

Sample sheet

1. **Reference** column.

   Pre-built scATAC-seq references are summarized below.

Keyword	Description
GRCh38_atac_v1.2.0	Human GRCh38, cellranger-atac reference 1.2.0
mm10_atac_v1.2.0	Mouse mm10, cellranger-atac reference 1.2.0
hg19_atac_v1.2.0	Human hg19, cellranger-atac reference 1.2.0
b37_atac_v1.2.0	Human b37 build, cellranger-atac reference 1.2.0
GRCh38_and_mm10_atac_v1.2.0	Human GRCh38 and mouse mm10, cellranger-atac reference 1.2.0
hg19_and_mm10_atac_v1.2.0	Human hg19 and mouse mm10, cellranger-atac reference 1.2.0
GRCh38_atac_v1.1.0	Human GRCh38, cellranger-atac reference 1.1.0
mm10_atac_v1.1.0	Mouse mm10, cellranger-atac reference 1.1.0
hg19_atac_v1.1.0	Human hg19, cellranger-atac reference 1.1.0
b37_atac_v1.1.0	Human b37 build, cellranger-atac reference 1.1.0
GRCh38_and_mm10_atac_v1.1.0	Human GRCh38 and mouse mm10, cellranger-atac reference 1.1.0
hg19_and_mm10_atac_v1.1.0	Human hg19 and mouse mm10, cellranger-atac reference 1.1.0

2. **Index** column.

   Put 10x single cell ATAC sample index set names (e.g. SI-NA-B1) here.

3. *Chemistry* column.

   This column is not used for scATAC-seq data. Put **auto** here as a placeholder if you decide to include the Chemistry column.

4. *DataType* column.

   Set it to **atac**.

5. *FetureBarcodeFile* column.

   Leave it blank for scATAC-seq.

6. Example:

```
Sample,Reference,Flowcell,Lane,Index,Chemistry,DataType
sample_atac,GRCh38_atac_v1.1.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK10WBC9YB,*,SI-NA-A1,auto,atac
```

Workflow input

`cellranger_workflow` takes Illumina outputs as input and runs `cellranger-atac mkfastq` and `cellranger-atac count`. Please see the description of inputs below. Note that required inputs are shown in bold.

Name	Description	Example	Default
input_csv_file	Sample Sheet (contains Sample, Reference, Flowcell, Lane, Index as required and Chemistry, DataType, FeatureBarcodeFile as optional)	"gs://fc-e0000000-0000-0000-0000-000000000000/sample_sheet.csv"	
output_directory	Output directory	"gs://fc-e0000000-0000-0000-0000-000000000000/cellranger_output"	
run_mkfastq	If you want to run `cellranger-atac mkfastq`	true	true
run_count	If you want to run `cellranger-atac count`	true	true
delete_input_bcl_dir	If delete BCL directories after demux. If false, you should delete this folder yourself so as to not incur storage charges	false	false
mkfastq_barcode_mismatches	Number of mismatches allowed in matching barcode indices (bcl2fastq2 default is 1)	0	
force_cells	Force pipeline to use this number of cells, bypassing the cell detection algorithm	6000	
cellranger_atac_version	cellranger-atac version. Available options: 1.2.0, 1.1.0	"1.2.0"	"1.2.0"
docker_registry	Docker registry to use for cellranger_workflow. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"
zones	Google cloud zones	"us-central1-a us-west1-a"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
atac_num_core	Number of cpus for cellranger-atac count	64	64
atac_memory	Memory string for cellranger-atac count	"57.6G"	"57.6G"
mkfastq_disk_space	Optional disk space in GB for cellranger-atac mkfastq	1500	1500
atac_disk_space	Disk space in GB needed for cellranger-atac count	500	500
preemptible	Number of preemptible tries	2	2

Workflow output

See the table below for important scATAC-seq outputs.

Name	Type	Description
output_fastqs_directory	Array[String]	A list of google bucket urls containing FASTQ files, one url per flowcell.
output_count_directory	Array[String]	A list of google bucket urls containing cellranger-atac count outputs, one url per sample.
metrics_summaries	File	A excel spreadsheet containing QCs for each sample.
output_web_summary	Array[File]	A list of htmls visualizing QCs for each sample (cellranger count output).
count_matrix	String	gs url for a template count_matrix.csv to run cumulus.

Aggregate scATAC-Seq Samples

To aggregate multiple scATAC-Seq samples, follow the instructions below:

1. Import `cellranger_atac_aggr` workflow. Please see Step 1 here, and the name of workflow is "**cumulus/cellranger_atac_aggr**".
2. Set the inputs of workflow. Please see the description of inputs below. Notice that required inputs are shown in bold:

Name	Description	Example	Default
aggr_id	Aggregate ID.	"aggr_sample"	
input_counts_directories	A comma-separated URLs to directories of samples to be aggregated.	"gs://fc-e0000000-0000-0000-0000-000000000000/data/sample1,gs://fc-e0000000-0000-0000-0000-000000000000/data/sample2"	
output_directory	Output directory	"gs://fc-e0000000-0000-0000-0000-000000000000/aggregate_result"	
genome	The reference genome name used by Cell Ranger, can be either a keyword of pre-built genome, or a Google Bucket URL. See this table for the list of keywords of pre-built genomes.	"GRCh38_atac_v1.2.0"	
normalize	Sample normalization mode. Options are: `none`, `depth`, or `signal`.	"none"	"none"
secondary	Perform secondary analysis (dimensionality reduction, clustering and visualization).	false	false
dim_reduce	Choose the algorithm for dimensionality reduction prior to clustering and tsne. Options are: `lsa`, `plsa`, or `pca`.	"lsa"	"lsa"
cellranger_atac_version	Cell Ranger ATAC version to use. Options: `1.2.0`, `1.1.0`.	"1.2.0"	"1.2.0"
zones	Google cloud zones	"us-central1-a us-west1-a"	"us-central1-b"
num_cpu	Number of cpus to request for cellranger atac aggr.	64	64
memory	Memory size string for cellranger atac aggr.	"57.6G"	"57.6G"
disk_space	Disk space in GB needed for cellranger atac aggr.	500	500
preemptible	Number of preemptible tries.	2	2
docker_registry	Docker registry to use for cellranger_workflow. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"

3. Check out the output in `output_directory/aggr_id` folder, where `output_directory` and `aggr_id` are the inputs you set in Step 2.

14.3.5 Single-cell immune profiling

To process single-cell immune profiling (scIR-seq) data, follow the specific instructions below.

Sample sheet

1. **Reference** column.

Pre-built scIR-seq references are summarized below.

Keyword	Description
GRCh38_vdj_v4.0.0	Human GRCh38 V(D)J sequences, cellranger reference 4.0.0, annotation built from Ensembl *Homo_sapiens.GRCh38.94.chr_patch_hapl_scaff.gtf*
GRCm38_vdj_v4.0.0	Mouse GRCm38 V(D)J sequences, cellranger reference 4.0.0, annotation built from Ensembl *Mus_musculus.GRCm38.94.gtf*
GRCh38_vdj_v3.1.0	Human GRCh38 V(D)J sequences, cellranger reference 3.1.0, annotation built from Ensembl *Homo_sapiens.GRCh38.94.chr_patch_hapl_scaff.gtf*
GRCm38_vdj_v3.1.0	Mouse GRCm38 V(D)J sequences, cellranger reference 3.1.0, annotation built from Ensembl *Mus_musculus.GRCm38.94.gtf*
GRCh38_vdj_v2.0.0 or **GRCh38_vdj**	Human GRCh38 V(D)J sequences, cellranger reference 2.0.0, annotation built from Ensembl *Homo_sapiens.GRCh38.87.chr_patch_hapl_scaff.gtf* and *vdj_GRCh38_alts_ensembl_10x_genes-2.0.0.gtf*
GRCm38_vdj_v2.2.0 or **GRCm38_vdj**	Mouse GRCm38 V(D)J sequences, cellranger reference 2.2.0, annotation built from Ensembl *Mus_musculus.GRCm38.90.chr_patch_hapl_scaff.gtf*

2. **Index** column.

   Put 10x single cell V(D)J sample index set names (e.g. SI-GA-A3) here.

3. *Chemistry* column.

   This column is not used for scIR-seq data. Put **fiveprime** here as a placeholder if you decide to include the Chemistry column.

4. *DataType* column.

   Set it to **vdj**.

5. *FetureBarcodeFile* column.

   Leave it blank for scIR-seq.

6. Example:

```
Sample,Reference,Flowcell,Lane,Index,Chemistry,DataType
sample_vdj,GRCh38_vdj_v3.1.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK10WBC9ZZ,1,SI-GA-A1,fiveprime,vdj
```

Workflow input

For scIR-seq data, `cellranger_workflow` takes Illumina outputs as input and runs `cellranger mkfastq` and `cellranger vdj`. Revalant workflow inputs are described below, with required inputs highlighted in bold.

Name	Description	Example	Default
input_csv_file	Sample Sheet (contains Sample, Reference, Flowcell, Lane, Index as required and Chemistry, DataType, FeatureBarcodeFile as optional)	"gs://fc-e0000000-0000-0000-0000-000000000000/sample_sheet.csv"	
output_directory	Output directory	"gs://fc-e0000000-0000-0000-0000-000000000000/cellranger_output"	
run_mkfastq	If you want to run `cellranger mkfastq`	true	true
delete_input_tarball	Delete tarball directories after demux. If false, you should delete this folder yourself so as to not incur storage charges	false	false
mkfastq_barcode_mismatches	Number of mismatches allowed in matching barcode indices (bcl2fastq2 default is 1)	0	
force_cells	Force pipeline to use this number of cells, bypassing the cell detection algorithm	6000	
vdj_denovo	Do not align reads to reference V(D)J sequences before de novo assembly	false	false
cellranger_version	cellranger version, could be 4.0.0, 3.1.0, 3.0.2, 2.2.0	"4.0.0"	"4.0.0"
docker_registry	Docker registry to use for cellranger_workflow. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"
cellranger_mkfastq_docker_registry	Docker registry to use for `cellranger mkfastq`. Default is the registry to which only Broad users have access. See *bcl2fastq* for making your own registry.	"gcr.io/broad-cumulus"	"gcr.io/broad-cumulus"
zones	Google cloud zones	"us-central1-a us-west1-a"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
num_cpu	Number of cpus to request for one node for cellranger mkfastq and cellranger vdj	32	32
memory	Memory size string for cellranger mkfastq and cellranger vdj	"120G"	"120G"
mkfastq_disk_space	Optional disk space in GB for mkfastq	1500	1500
vdj_disk_space	Disk space in GB needed for cellranger vdj	500	500
preemptible	Number of preemptible tries	2	2

Workflow output

See the table below for important scIR-seq outputs.

Name	Type	Description
output_fastqs_directory	Array[String]	A list of google bucket urls containing FASTQ files, one url per flowcell.
output_vdj_directory	Array[String]	A list of google bucket urls containing vdj results, one url per sample.
metrics_summaries	File	A excel spreadsheet containing QCs for each sample.
output_web_summary	Array[File]	A list of htmls visualizing QCs for each sample (cellranger count output).
count_matrix	String	gs url for a template count_matrix.csv to run cumulus.

14.3.6 Build Cell Ranger References

We provide routines wrapping Cell Ranger tools to build references for sc/snRNA-seq, scATAC-seq and single-cell immune profiling data.

Build references for sc/snRNA-seq

We provide a wrapper of `cellranger mkref` to build sc/snRNA-seq references. Please follow the instructions below.

1. Import `cellranger_create_reference`

Import *cellranger_create_reference* workflow to your workspace.

See the Terra documentation for adding a workflow. The *cellranger_workflow* workflow is under `Broad Methods Repository` with name "**cumulus/cellranger_create_reference**".

Moreover, in the workflow page, click the `Export to Workspace...` button, and select the workspace to which you want to export *cellranger_create_reference* workflow in the drop-down menu.

2. Upload requred data to Google Bucket

Required data may include input sample sheet, genome FASTA files and gene annotation GTF files.

3. Input sample sheet

If multiple species are specified, a sample sheet in CSV format is required. We describe the sample sheet format below, with required columns highlighted in bold:

Column	Description
Genome	Genome name
Fasta	Location to the genome assembly in FASTA/FASTA.gz format
Genes	Location to the gene annotation file in GTF/GTF.gz format
Attributes	Optional, A list of `key:value` pairs separated by `;`. If set, `cellranger mkgtf` will be called to filter the user-provided GTF file. See 10x filter with mkgtf for more details

Please note that the columns in the CSV can be in any order, but that the column names must match the recognized headings.

See below for an example for building Example:

```
Genome,Fasta,Genes,Attributes
GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/GRCh38.fa.gz,gs://fc-
↪e0000000-0000-0000-0000-000000000000/GRCh38.gtf.gz,gene_biotype:protein_
↪coding;gene_biotype:lincRNA;gene_biotype:antisense
mm10,gs://fc-e0000000-0000-0000-0000-000000000000/mm10.fa.gz,gs://fc-
↪e0000000-0000-0000-0000-000000000000/mm10.gtf.gz
```

If multiple species are specified, the reference will built under **Genome** names concatenated by '_and_'s. In the above example, the reference is stored under 'GRCh38_and_mm10'.

4. Workflow input

Required inputs are highlighted in bold. Note that **input_sample_sheet** and **input_fasta**, **input_gtf** , **genome** and attributes are mutually exclusive.

Name	Description	Example	Default
input_sample_sheet	A sample sheet in CSV format allows users to specify more than 1 genomes to build references (e.g. human and mouse). If a sample sheet is provided, **input_fasta**, **input_gtf**, and attributes will be ignored.	"gs://fc-e0000000-0000-0000-0000-000000000000/input_sample_sheet.csv"	
input_fasta	Input genome reference in either FASTA or FASTA.gz format	"gs://fc-e0000000-0000-0000-0000-000000000000/Homo_sapiens.GRCh38.dna.toplevel.fa.gz"	
input_gtf	Input gene annotation file in either GTF or GTF.gz format	"gs://fc-e0000000-0000-0000-0000-000000000000/Homo_sapiens.GRCh38.94.chr_patch_hapl_scaff.gtf.gz"	
genome	Genome reference name. New reference will be stored in a folder named **genome**	refdata-cellranger-vdj-GRCh38-alts-ensembl-3.1.0	
output_directory	Output directory	"gs://fc-e0000000-0000-0000-0000-000000000000/cellranger_reference"	
attributes	A list of `key:value` pairs separated by `;`. If this option is not None, `cellranger mkgtf` will be called to filter the user-provided GTF file. See 10x filter with mkgtf for more details	"gene_biotype:protein_coding;gene_biotype:lincRNA;gene_biotype:antisense"	
pre_mrna	If we want to build pre-mRNA references, in which we use full length transcripts as exons in the annotation file. We follow 10x build Cell Ranger compatible pre-mRNA Reference Package to build pre-mRNA references	true	false
ref_version	reference version string	Ensembl v94	
cellranger_version	cellranger version, could be 4.0.0, 3.1.0, 3.0.2, or 2.2.0	"4.0.0"	"4.0.0"
docker_registry	Docker registry to use for cellranger_workflow. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"
zones	Google cloud zones	"us-central1-a us-west1-a"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
num_cpu	Number of cpus to request for one node for building indices	1	1
memory	Memory size in GB	32	32
disk_space	Optional disk space in GB	100	100
preemptible	Number of preemptible tries	2	2

5. Workflow output

Name	Type	Description
output_reference	File	Gzipped reference folder with name *genome.tar.gz*. We will also store a copy of the gzipped tarball under **output_directory** specified in the input.

Build references for scATAC-seq

We provide a wrapper of `cellranger-atac mkref` to build scATAC-seq references. Please follow the instructions below.

1. Import `cellranger_atac_create_reference`

Import *cellranger_atac_create_reference* workflow to your workspace.

See the Terra documentation for adding a workflow. The *cellranger_workflow* workflow is under `Broad Methods Repository` with name "**cumulus/cellranger_atac_create_reference**".

Moreover, in the workflow page, click the `Export to Workspace...` button, and select the workspace to which you want to export *cellranger_atac_create_reference* workflow in the drop-down menu.

2. Upload required data to Google Bucket

Required data include config JSON file, genome FASTA file, gene annotation file (GTF or GFF3 format) and motif input file (JASPAR format).

3. Workflow input

Required inputs are highlighted in bold.

Name	Description	Example	Default
genome	Genome reference name. New reference will be stored in a folder named **genome**	refdata-cellranger-atac-mm10-1.1.0	
config_json	Configuration file defined in 10x genomics configuration file. Note that links to files in the JSON must be Google bucket URLs	"gs://fc-e0000000-0000-0000-0000-000000000000/config.json"	
output_directory	Output directory	"gs://fc-e0000000-0000-0000-0000-000000000000/cellranger_atac_reference"	
cellranger_atac_version	cellranger-atac version, could be: 1.2.0, 1.1.0	"1.2.0"	"1.2.0"
docker_registry	Docker registry to use for cellranger_workflow. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"
zones	Google cloud zones	"us-central1-a us-west1-a"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
memory	Memory size string for cellranger-atac mkref	"32G"	"32G"
disk_space	Optional disk space in GB	100	100
preemptible	Number of preemptible tries	2	2

4. Workflow output

Name	Type	Description
output_reference	File	Gzipped reference folder with name *genome.tar.gz*. We will also store a copy of the gzipped tarball under **output_directory** specified in the input.

Build references for single-cell immune profiling data

We provide a wrapper of `cellranger mkvdjref` to build single-cell immune profiling references. Please follow the instructions below.

1. Import `cellranger_vdj_create_reference`

Import *cellranger_vdj_create_reference* workflow to your workspace.

See the Terra documentation for adding a workflow. The *cellranger_workflow* workflow is under `Broad Methods Repository` with name "**cumulus/cellranger_vdj_create_reference**".

Moreover, in the workflow page, click the `Export to Workspace...` button, and select the workspace to which you want to export *cellranger_vdj_create_reference* workflow in the drop-down menu.

2. Upload requred data to Google Bucket

Required data include genome FASTA file and gene annotation file (GTF format).

3. Workflow input

Required inputs are highlighted in bold.

Name	Description	Example	Default
input_fasta	Input genome reference in either FASTA or FASTA.gz format	"gs://fc-e0000000-0000-0000-0000-000000000000/Homo_sapiens.GRCh38.dna.toplevel.fa.gz"	
input_gtf	Input gene annotation file in either GTF or GTF.gz format	"gs://fc-e0000000-0000-0000-0000-000000000000/Homo_sapiens.GRCh38.94.chr_patch_hapl_scaff.gtf.gz"	
genome	Genome reference name. New reference will be stored in a folder named **genome**	refdata-cellranger-vdj-GRCh38-alts-ensembl-3.1.0	
output_dir	Output directory	"gs://fc-e0000000-0000-0000-0000-000000000000/cellranger_vdj_reference"	
ref_version	reference version string	Ensembl v94	
cellranger_version	cellranger version, could be 4.0.0, 3.1.0, 3.0.2, or 2.2.0	"4.0.0"	"4.0.0"
docker_registry	Docker registry to use for cellranger_workflow. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"
zones	Google cloud zones	"us-central1-a us-west1-a"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
memory	Memory size string for cellranger-atac mkref	"32G"	"32G"
disk_space	Optional disk space in GB	100	100
preemptible	Number of preemptible tries	2	2

4. Workflow output

Name	Type	Description
output_reference	File	Gzipped reference folder with name *genome.tar.gz*. We will also store a copy of the gzipped tarball under **output_directory** specified in the input.

14.4 bcl2fastq

14.4.1 License

bcl2fastq license

14.4.2 Workflows

Workflows such as **cellranger_workflow** and **dropseq_workflow** provide the option of running `bcl2fastq`. We provide dockers containing `bcl2fastq` that are accessible only by members of the Broad Institute. Non-Broad Institute members will have to provide their own docker images. Please note that if you're a Broad Institute member and are not able to pull the docker image, please check https://app.terra.bio/#groups to see that you're a member of the all_broad_users group. If not, please contact Terra support and ask to be added to the all_broad_users@firecloud.org group.

14.4.3 Docker

Read this tutorial if you are new to Docker.

Then for a Debian based docker (e.g. continuumio/miniconda3), create the Dockerfile as follows:

```
RUN apt-get update && apt-get install --no-install-recommends -y alien unzip
ADD bcl2fastq2-v2-20-0-linux-x86-64.zip /software/
RUN unzip -d /software/ /software/bcl2fastq2-v2-20-0-linux-x86-64.zip && alien -i /
↪software/bcl2fastq2-v2.20.0.422-Linux-x86_64.rpm && rm /software/bcl2fastq2-v2*
```

Next, download `bcl2fastq` from the Illumina website, which requires registration. Choose the `Linux rpm` file format and download bcl2fastq2-v2-20-0-linux-x86-64.zip to the same directory as your Dockerfile.

You can host your private docker images in the Google Container Registry.

14.4.4 Example

In this example we create a docker image for running `cellranger mkfastq` version 3.0.2.

1. Create a GCP project or reuse an existing project.
2. Enable the Google Container Registry
3. Clone the cumulus repository:

   ```
   git clone https://github.com/klarman-cell-observatory/cumulus.git
   ```

4. Add the lines to cumulus/docker/cellranger/3.0.2/Dockerfile to include bcl2fastq (see *Docker*).

5. Ensure you have Docker installed
6. Download cellranger from https://support.10xgenomics.com/single-cell-gene-expression/software/downloads/3.0
7. Build, tag, and push the docker. Remember to replace PROJECT_ID with your GCP project id:

```
cd cumulus/docker/cellranger/3.0.2/
docker build -t cellranger-3.0.2 .
docker tag cellranger-3.0.2 gcr.io/PROJECT_ID/cellranger:3.0.2
gcr.io/PROJECT_ID/cellranger:3.0.2
```

8. Import **cellranger_workflow** workflow to your workspace (see cellranger_workflow steps), and enter your docker registry URL (in this example, `"gcr.io/PROJECT_ID/"`) in `cellranger_mkfastq_docker_registry` field of cellranger_workflow inputs.

14.5 Cell Ranger alternatives to generate gene-count matrices for 10X data

This `count` workflow generates gene-count matrices from 10X FASTQ data using alternative methods other than Cell Ranger.

14.5.1 Prepare input data and import workflow

1. Run `cellranger_workflow` to generate FASTQ data

You can skip this step if your data are already in FASTQ format.

Otherwise, you need to first run *cellranger_workflow* to generate FASTQ files from BCL raw data for each sample. Please follow cellranger_workflow manual.

Notice that you should set **run_mkfastq** to `true` to get FASTQ output. You can also set **run_count** to `false` if you want to skip Cell Ranger count, and only use the result from *count* workflow.

For Non-Broad users, you'll need to build your own docker for *bcl2fastq* step. Instructions are here.

2. Import `count`

Import *count* workflow to your workspace.

See the Terra documentation for adding a workflow. The *count* workflow is under `Broad Methods Repository` with name "**cumulus/count**".

Moreover, in the workflow page, click the `Export to Workspace...` button, and select the workspace to which you want to export *count* workflow in the drop-down menu.

3. Prepare a sample sheet

3.1 Sample sheet format:

The sample sheet for *count* workflow should be in TSV format, i.e. columns are seperated by tabs not commas. Please note that the columns in the TSV can be in any order, but that the column names must match the recognized headings.

The sample sheet describes how to identify flowcells and generate channel-specific count matrices.

A brief description of the sample sheet format is listed below **(required column headers are shown in bold)**.

Column	Description
Sample	Contains sample names. Each 10x channel should have a unique sample name.
Flowcells	Indicates the Google bucket URLs of folder(s) holding FASTQ files of this sample.

The sample sheet supports sequencing the same 10x channel across multiple flowcells. If a sample is sequenced across multiple flowcells, simply list all of its flowcells in a comma-seperated way. In the following example, we have 2 samples sequenced in two flowcells.

Example:

```
Sample  Flowcells
sample_1        gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4/
↪sample_1_fastqs,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2/
↪sample_1_fastqs
sample_2        gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4/
↪sample_2_fastqs
```

Moreover, if one flowcell of a sample contains multiple FASTQ files for each read, i.e. sequences from multiple lanes, you should keep your sample sheet as the same, and *count* workflow will automatically merge lanes altogether for the sample before performing counting.

3.2 Upload your sample sheet to the workspace bucket:

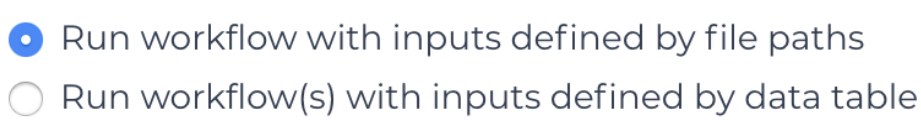
Use gsutil (you already have it if you've installed Google cloud SDK) in your unix terminal to upload your sample sheet to workspace bucket.

Example:

```
gsutil cp /foo/bar/projects/sample_sheet.tsv gs://fc-e0000000-0000-0000-0000-
↪000000000000/
```

4. Launch analysis

In your workspace, open `count` in `WORKFLOWS` tab. Select the desired snapshot version (e.g. latest). Select `Process single workflow from files` as below



and click `SAVE` button. Select `Use call caching` and click `INPUTS`. Then fill in appropriate values in the `Attribute` column. Alternative, you can upload a JSON file to configure input by clicking `Drag or click to upload json`.

Once INPUTS are appropriated filled, click `RUN ANALYSIS` and then click `LAUNCH`.

14.5.2 Workflow inputs

Below are inputs for *count* workflow. Notice that required inputs are in bold.

Name	Description	Example	Default
input_tsv_file	Input TSV sample sheet describing metadata of each sample.	"gs://fc-e0000000-0000-0000-0000-000000000000/sample_sheet.tsv"	
genome	Genome reference name. Current support: GRCh38, mm10.	"GRCh38"	
chemistry	10X genomics' chemistry name. Current support: "tenX_v3" (for V3 chemistry), "tenX_v2" (for V2 chemistry).	"tenX_v3"	
output_directory	GS URL of output directory.	"gs://fc-e0000000-0000-0000-0000-000000000000/count_result"	
run_count	If you want to run count tools to generate gene-count matrices.	true	true
count_tool	Count tool to generate result. Options: • "StarSolo": Use STARsolo. • "Optimus": Use Optimus pipeline, developed by the Data Coordination Platform team of the Human Cell Atlas. • "Bustools": Use Kallisto BUSTools. • "Alevin": Use Salmon Alevin.	"StarSolo"	"StarSolo"
docker_registry	Docker registry to use. Notice that docker image for Bustools is seperate. • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.		"cumulusprod"
zones	Google cloud zones to consider for execution.	"us-east1-d us-west1-a us-west1-b"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
num_cpu	Number of CPUs to request for count per channel. Notice that when use Optimus for count, this input only affects steps of copying files. Optimus uses CPUs due to its own strategy.	32	32
disk_space	Disk space in GB needed for count per channel. Notice that when use Optimus for count, this input only affects steps of copying files. Optimus uses disk space due to its own strategy.	500	500
memory	Memory size in GB needed for count per channel.	120	120

14.5.3 Workflow outputs

See the table below for *count* workflow outputs.

Name	Type	Description
output_folder	String	Google Bucket URL of output directory. Within it, each folder is for one sample in the input sample sheet.

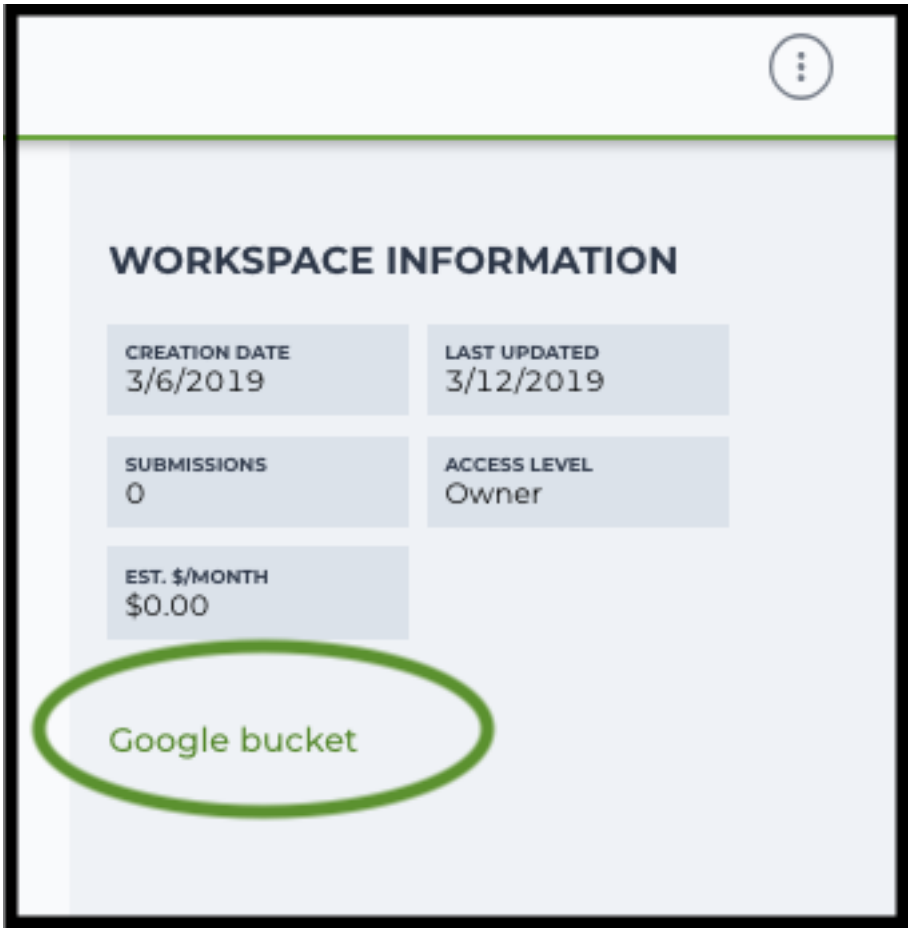
14.6 Extract gene-count matrices from plated-based SMART-Seq2 data

14.6.1 Run SMART-Seq2 Workflow

Follow the steps below to extract gene-count matrices from SMART-Seq2 data on Terra. This WDL aligns reads using *STAR*, *HISAT2*, or *Bowtie 2* and estimates expression levels using *RSEM*.

1. Copy your sequencing output to your workspace bucket using gsutil in your unix terminal.

   You can obtain your bucket URL in the dashboard tab of your Terra workspace under the information panel.



   Note: Broad users need to be on an UGER node (not a login node) in order to use the `-m` flag

Request an UGER node:

```
reuse UGER
qrsh -q interactive -l h_vmem=4g -pe smp 8 -binding linear:8 -P regevlab
```

The above command requests an interactive node with 4G memory per thread and 8 threads. Feel free to change the memory, thread, and project parameters.

Once you're connected to an UGER node, you can make gsutil available by running:

```
reuse Google-Cloud-SDK
```

Use `gsutil cp [OPTION]... src_url dst_url` to copy data to your workspace bucket. For example, the following command copies the directory at /foo/bar/nextseq/Data/VK18WBC6Z4 to a Google bucket:

```
gsutil -m cp -r /foo/bar/nextseq/Data/VK18WBC6Z4 gs://fc-e0000000-0000-
↪0000-0000-000000000000/VK18WBC6Z4
```

`-m` means copy in parallel, `-r` means copy the directory recursively.

2. Create a sample sheet.

   Please note that the columns in the CSV can be in any order, but that the column names must match the recognized headings.

   The sample sheet provides metadata for each cell:

Column	Description
Cell	Cell name.
Plate	Plate name. Cells with the same plate name are from the same plate.
Read1	Location of the FASTQ file for read1 in the cloud (gsurl).
Read2	(Optional). Location of the FASTQ file for read2 in the cloud (gsurl). This field can be skipped for single-end reads.

   Example:

```
Cell,Plate,Read1,Read2
cell-1,plate-1,gs://fc-e0000000-0000-0000-0000-000000000000/smartseq2/
↪cell-1_L001_R1_001.fastq.gz,gs://fc-e0000000-0000-0000-0000-
↪000000000000/smartseq2/cell-1_L001_R2_001.fastq.gz
cell-2,plate-1,gs://fc-e0000000-0000-0000-0000-000000000000/smartseq2/
↪cell-2_L001_R1_001.fastq.gz,gs://fc-e0000000-0000-0000-0000-
↪000000000000/smartseq2/cell-2_L001_R2_001.fastq.gz
cell-3,plate-2,gs://fc-e0000000-0000-0000-0000-000000000000/smartseq2/
↪cell-3_L001_R1_001.fastq.gz,
cell-4,plate-2,gs://fc-e0000000-0000-0000-0000-000000000000/smartseq2/
↪cell-4_L001_R1_001.fastq.gz,
```

3. Upload your sample sheet to the workspace bucket.

   Example:

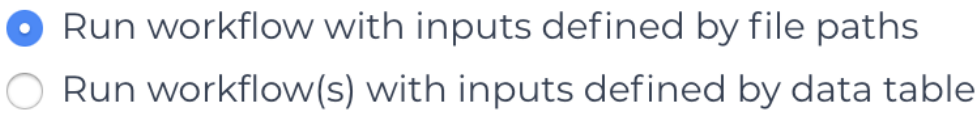
```
gsutil cp /foo/bar/projects/sample_sheet.csv gs://fc-e0000000-0000-0000-
↪0000-000000000000/
```

4. Import *smartseq2* workflow to your workspace.

See the Terra documentation for adding a workflow. The *smartseq2* workflow is under `Broad Methods Repository` with name "**cumulus/smartseq2**".

Moreover, in the workflow page, click `Export to Workspace...` button, and select the workspace to which you want to export *smartseq2* workflow in the drop-down menu.

5. In your workspace, open `smartseq2` in `WORKFLOWS` tab. Select `Run workflow with inputs defined by file paths` as below



and click `SAVE` button.

Inputs:

Please see the description of inputs below. Note that required inputs are shown in bold.

Name	Description	Example	Default
input_csv_file	Sample Sheet (contains Cell, Plate, Read1, Read2)	"gs://fc-e0000000-0000-0000-0000-000000000000/sample_sheet.csv"	
output_directory	Output directory	"gs://fc-e0000000-0000-0000-0000-000000000000/smartseq2_output"	
reference	Reference transcriptome to align reads to. Acceptable values: • Pre-created genome references: – "GRCh38_ens93filt" for human, genome version is GRCh38, gene annotation is generated using human Ensembl 93 GTF according to cellranger mkgtf; – "GRCm38_ens93filt" for mouse, genome version is GRCm38, gene annotation is generated using mouse Ensembl 93 GTF according to cellranger mkgtf; • Create a custom genome reference using smartseq2_create_reference workflow, and specify its Google bucket URL here.	"GRCh38_ens93filt", or "gs://fc-e0000000-0000-0000-0000-000000000000/rsem_ref.tar.gz"	
aligner	Which aligner to use for read alignment. Options are "hisat2-hca", "star" and "bowtie"	"star"	"hisat2-hca"
output_genome_bam	Whether to output bam file with alignments mapped to genomic coordinates and annotated with their posterior probabilities.	false	false
normalize_tpm_by_sequencing_depth	Whether to normalize TPM values by sequencing depth.	true	true
smartseq2_version	SMART-Seq2 version to use. Versions available: 1.1.0.	"1.1.0"	"1.1.0"
docker_registry	Docker registry to use. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"
zones	Google cloud zones	"us-east1-d us-west1-a us-west1-b"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
num_cpu	Number of cpus to request for one node	4	4

Outputs:

Name	Type	Description
output_count_matrix	Array[String]	A list of google bucket urls containing gene-count matrices, one per plate. Each gene-count matrix file has the suffix `.dge.txt.gz`.
output_qc_report	Array[String]	A list of google bucket urls containing simple quality control statistics, one per plate. Each file contains one line per cell and each line has three columns: Total reads, Alignment rate and Unique rate.
rsem_gene	Array[Array[File]]	A 2D array of RSEM gene expression estimation files.
rsem_gene	Array[Array[File]]	A 2D array of RSEM gene expression estimation files.
rsem_isoform	Array[Array[File]]	A 2D array of RSEM isoform expression estimation files.
rsem_trans_bam	Array[Array[File]]	A 2D array of RSEM transcriptomic BAM files.
rsem_genome_bam	Array[Array[File]]	A 2D array of RSEM genomic BAM files if `output_genome_bam` is `true`.
rsem_time	Array[Array[File]]	A 2D array of RSEM execution time log files.
aligner_log	Array[Array[File]]	A 2D array of Aligner log files.
rsem_cnt	Array[Array[File]]	A 2D array of RSEM count files.
rsem_model	Array[Array[File]]	A 2D array of RSEM model files.
rsem_theta	Array[Array[File]]	A 2D array of RSEM generated theta files.

This WDL generates one gene-count matrix per SMART-Seq2 plate. The gene-count matrix uses Drop-Seq format:

- The first line starts with `"Gene"` and then gives cell barcodes separated by tabs.
- Starting from the second line, each line describes one gene. The first item in the line is the gene name and the rest items are TPM-normalized count values of this gene for each cell.

The gene-count matrices can be fed directly into **cumulus** for downstream analysis.

TPM-normalized counts are calculated as follows:

1. Estimate the gene expression levels in TPM using *RSEM*.
2. Suppose `c` reads are achieved for one cell, then calculate TPM-normalized count for gene `i` as `TPM_i / 1e6 * c`.

TPM-normalized counts reflect both the relative expression levels and the cell sequencing depth.

14.6.2 Custom Genome

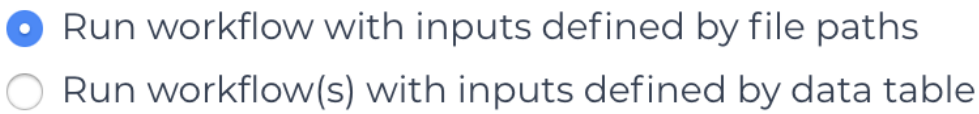
We also provide a way of generating user-customized Genome references for SMART-Seq2 workflow.

1. Import smartseq2_create_reference workflow to your workspace.

   See the Terra documentation for adding a workflow. The smartseq2_create_reference workflow is under `Broad Methods Repository` with name "**cumulus/smartseq2_create_reference**".

Moreover, in the workflow page, click `Export to Workflow...` button, and select the workspace to which you want to export `smartseq2_create_reference` in the drop-down menu.

2. In your workspace, open `smartseq2_create_reference` in `WORKFLOWS` tab. Select `Run workflow with inputs defined by file paths` as below



and click `SAVE` button.

Inputs:

Please see the description of inputs below. Note that required inputs are shown in bold.

Name	Description	Type or Example	Default
fasta	Genome fasta file	File. For example, "gs://fc-e0000000-0000-0000-0000-000000000000/Homo_sapiens.GRCh38.dna.primary_assembly.fa"	
gtf	GTF gene annotation file (e.g. Homo_sapiens.GRCh38.83.gtf)	File. For example, "gs://fc-e0000000-0000-0000-0000-000000000000/Homo_sapiens.GRCh38.83.gtf"	
output_directory	Google bucket url for the output folder	"gs://fc-e0000000-0000-0000-0000-000000000000/output_refs"	
genome	Output reference genome name. Output reference is a gzipped tarball with name genome_aligner.tar.gz	"GRCm38_ens97filt"	
aligner	Build indices for which aligner, choices are hisat2-hca, star, or bowtie2.	"hisat2-hca"	"hisat2-hca"
smartseq2_version	SMART-Seq2 version to use. Versions available: 1.1.0. Versions obsoleted: 1.0.0.	"1.1.0"	"1.1.0"
docker_registry	Docker registry to use. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"quay.io/cumulus"	"cumulusprod"
zones	Google cloud zones	"us-central1-c"	"us-central1-b"
cpu	Number of CPUs	Integer	If aligner is bowtie2 or hisat2-hca, 8; otherwise 32
memory	Memory size string	String	If aligner is bowtie2 or hisat2-hca, "7.2G"; otherwise "120G"

Outputs

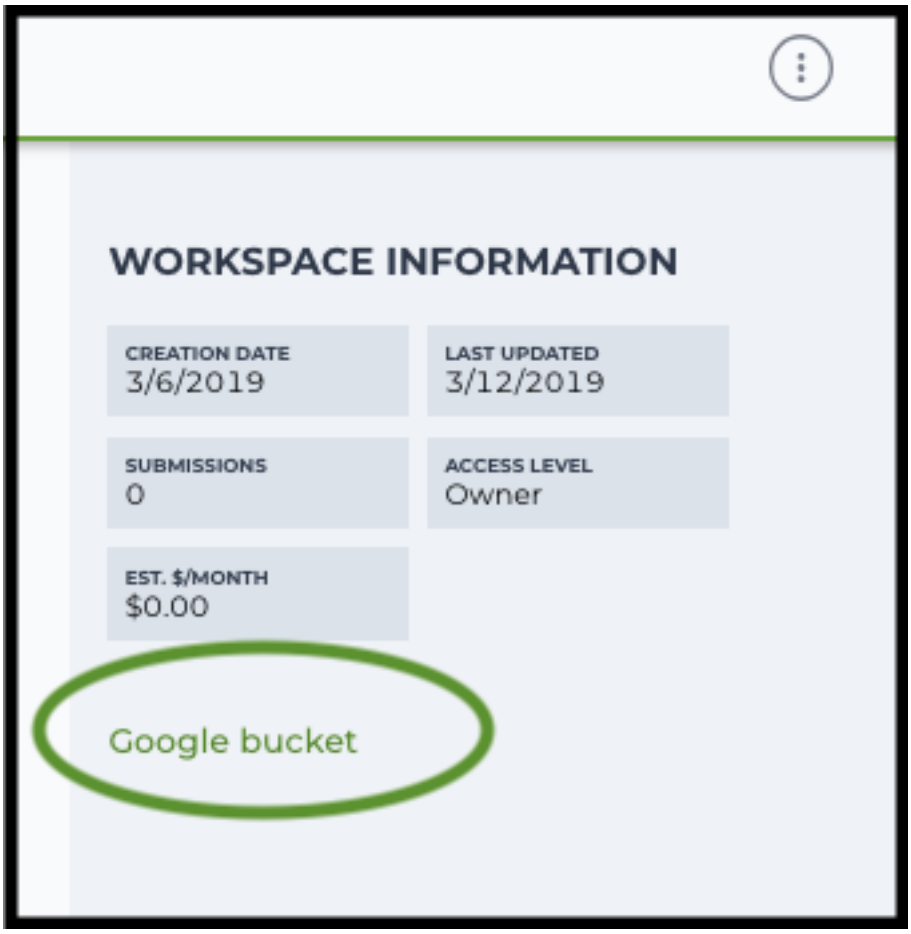
Name	Type	Description
output_reference	File	The custom Genome reference generated. Its default file name is `genome_aligner.tar.gz`.
monitoring_log	File	CPU and memory profiling log.

14.7 Drop-seq pipeline

This workflow follows the steps outlined in the Drop-seq alignment cookbook from the McCarroll lab , except the default STAR aligner flags are *–limitOutSJcollapsed 1000000 –twopassMode Basic*. Additionally the pipeline provides the option to generate count matrices using dropEst.

1. Copy your sequencing output to your workspace bucket using gsutil in your unix terminal.

   You can obtain your bucket URL in the dashboard tab of your Terra workspace under the information panel.



   Note: Broad users need to be on an UGER node (not a login node) in order to use the `-m` flag

   Request an UGER node:

```
reuse UGER
qrsh -q interactive -l h_vmem=4g -pe smp 8 -binding linear:8 -P regevlab
```

The above command requests an interactive node with 4G memory per thread and 8 threads. Feel free to change the memory, thread, and project parameters.

Once you're connected to an UGER node, you can make gsutil available by running:

```
reuse Google-Cloud-SDK
```

Use `gsutil cp [OPTION]... src_url dst_url` to copy data to your workspace bucket. For example, the following command copies the directory at /foo/bar/nextseq/Data/VK18WBC6Z4 to a Google bucket:

```
gsutil -m cp -r /foo/bar/nextseq/Data/VK18WBC6Z4 gs://fc-e0000000-0000-
↪0000-0000-000000000000/VK18WBC6Z4
```

`-m` means copy in parallel, `-r` means copy the directory recursively.

2. Non Broad Institute users that wish to run bcl2fastq must create a custom docker image.

   See *bcl2fastq* instructions.

3. Create a sample sheet.

   Please note that the columns in the CSV must be in the order shown below and does not contain a header line. The sample sheet provides either the FASTQ files for each sample if you've already run bcl2fastq or a list of BCL directories if you're starting from BCL directories. Please note that BCL directories must contain a valid bcl2fastq sample sheet (SampleSheet.csv):

Column	Description
Name	Sample name.
Read1	Location of the FASTQ file for read1 in the cloud (gsurl).
Read2	Location of the FASTQ file for read2 in the cloud (gsurl).

   Example using FASTQ input files:

```
sample-1,gs://fc-e0000000-0000-0000-0000-000000000000/dropseq-1/sample1-1_
↪L001_R1_001.fastq.gz,gs://fc-e0000000-0000-0000-0000-000000000000/
↪dropseq-1/sample-1_L001_R2_001.fastq.gz
sample-2,gs://fc-e0000000-0000-0000-0000-000000000000/dropseq-1/sample-2_
↪L001_R1_001.fastq.gz,gs://fc-e0000000-0000-0000-0000-000000000000/
↪dropseq-1/sample-2_L001_R2_001.fastq.gz
sample-1,gs://fc-e0000000-0000-0000-0000-000000000000/dropseq-2/sample1-1_
↪L001_R1_001.fastq.gz,gs://fc-e0000000-0000-0000-0000-000000000000/
↪dropseq-2/sample-1_L001_R2_001.fastq.gz
```

   Note that in this example, sample-1 was sequenced across two flowcells.

   Example using BCL input directories:

```
gs://fc-e0000000-0000-0000-0000-000000000000/flowcell-1
gs://fc-e0000000-0000-0000-0000-000000000000/flowcell-2
```

   Note that the flow cell directory must contain a bcl2fastq sample sheet named SampleSheet.csv.

4. Upload your sample sheet to the workspace bucket.

   Example:

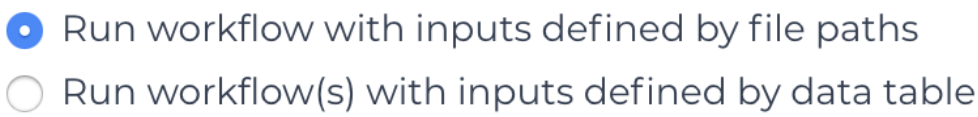
```
gsutil cp /foo/bar/projects/sample_sheet.csv gs://fc-e0000000-0000-0000-
↪0000-000000000000/
```

5. Import *dropseq_workflow* workflow to your workspace.

   See the Terra documentation for adding a workflow. The *dropseq_workflow* is under `Broad Methods Repository` with name "**cumulus/dropseq_workflow**".

   Moreover, in the workflow page, click the `Export to Workspace...` button, and select the workspace you want to export *dropseq_workflow* workflow in the drop-down menu.

6. In your workspace, open `dropseq_workflow` in `WORKFLOWS` tab. Select `Run workflow with inputs defined by file paths` as below

   and click the `SAVE` button.

14.7.1 Inputs

Please see the description of important inputs below.

Name	Description
input_csv_file	CSV file containing sample name, read1, and read2 or a list of BCL directories.
output_directory	Pipeline output directory (gs URL e.g. "gs://fc-e0000000-0000-0000-0000-000000000000/dropseq_output")
reference	hg19, GRCh38, mm10, hg19_mm10, mmul_8.0.1 or a path to a custom reference JSON file
run_bcl2fastq	Whether your sample sheet contains one BCL directory per line or one sample per line (default false)
run_dropseq_tools	Whether to generate count matrixes using Drop-Seq tools from the McCarroll lab (default true)
run_dropest	Whether to generate count matrixes using dropEst (default false)
cellular_barcode_whitelist	Optional whitelist of known cellular barcodes
drop_seq_tools_force_cells	If supplied, bypass the cell detection algorithm (the elbow method) and use this number of cells.
dropest_cells_max	Maximal number of output cells
dropest_genes_min	Minimal number of genes for cells after the merge procedure (default 100)
dropest_min_merge_threshold	The threshold for the merge procedure (default 0.2)
dropest_max_cb_merge_dist	Max edit distance between barcodes (default 2)
dropest_max_umi_merge_dist	Max edit distance between UMIs (default 1)
dropest_min_genes_before_merge	Minimal number of genes for cells before the merge procedure. Used mostly for optimization. (default 10)
dropest_merge_barcodes_per_cell	Use dropEst per cell merge strategy (can be slow), recommended to use when the list of real barcodes is not available (default true)
dropest_velocyto	Save separate count matrices for exons, introns and exon/intron spanning reads (default true)
trim_sequence	The sequence to look for at the start of reads for trimming (default "AAGCAGTGGTATCAACGCAGAGTGAATGGG")
trim_num_bases	How many bases at the beginning of the sequence must match before trimming occur (default 5)
umi_base_range	The base location of the molecular barcode (default 13-20)
cellular_barcode_base_range	The base location of the cell barcode (default 1-12)
star_flags	Additional options to pass to STAR aligner

Please note that run_bcl2fastq must be set to true if you're starting from BCL files instead of FASTQs.

Custom Genome JSON

If you're reference is not one of the predefined choices, you can create a custom JSON file. Example:

```
{
        "refflat":        "gs://fc-e0000000-0000-0000-0000-000000000000/human_mouse/
↪hg19_mm10_transgenes.refFlat",
        "genome_fasta":    "gs://fc-e0000000-0000-0000-0000-000000000000/human_mouse/
↪hg19_mm10_transgenes.fasta",
        "star_genome":    "gs://fc-e0000000-0000-0000-0000-000000000000/human_mouse/
↪STAR2_5_index_hg19_mm10.tar.gz",
        "gene_intervals":        "gs://fc-e0000000-0000-0000-0000-000000000000/human_
↪mouse/hg19_mm10_transgenes.genes.intervals",
        "genome_dict":    "gs://fc-e0000000-0000-0000-0000-000000000000/human_mouse/
↪hg19_mm10_transgenes.dict",
        "star_cpus": 32,
        "star_memory": "120G"
}
```

The fields star_cpus and star_memory are optional and are used as the default cpus and memory for running STAR with your genome.

14.7.2 Outputs

The pipeline outputs a list of google bucket urls containing one gene-count matrix per sample. Each gene-count matrix file produced by Drop-seq tools has the suffix 'dge.txt.gz', matrices produced by dropEst have the extension .rds.

Building a Custom Genome

The tool **dropseq_bundle** can be used to build a custom genome. Please see the description of important inputs below.

Name	Description
fasta_file	Array of fasta files. If more than one species, fasta and gtf files must be in the same order.
gtf_file	Array of gtf files. If more than one species, fasta and gtf files must be in the same order.
genomeSAindexNbases	Length (bases) of the SA pre-indexing string. Typically between 10 and 15. Longer strings will use much more memory, but allow faster searches. For small genomes, must be scaled down to min(14, log2(GenomeLength)/2 - 1)

dropseq_workflow Terra Release Notes

Version 11

- Added fastq_to_sam_memory and trim_bam_memory workflow inputs

Version 10

- Updated workflow to WDL version 1.0

Version 9

- Changed input bcl2fastq_docker_registry from optional to required

Version 8

- Added additional parameters for bcl2fastq

Version 7

- Added support for multi-species genomes (Barnyard experiments)

Version 6

- Added star_extra_disk_space and star_disk_space_multiplier workflow inputs to adjust disk space allocated for STAR alignment task.

Version 5

- Split preprocessing steps into separate tasks (FastqToSam, TagBam, FilterBam, and TrimBam).

Version 4

- Handle uncompressed fastq files as workflow input.
- Added optional prepare_fastq_disk_space_multiplier input.

Version 3

- Set default value for docker_registry input.

Version 2

- Added docker_registry input.

Version 1

- Renamed sccloud to cumulus
- Added use_bases_mask option when running bcl2fastq

Version 18

- Created a separate docker image for running bcl2fastq

Version 17

- Fixed bug that ignored WDL input star_flags (thanks to Carly Ziegler for reporting)
- Changed default value of star_flags to the empty string (Prior versions of the WDL incorrectly indicated that basic 2-pass mapping was done)

Version 16

- Use cumulus dockerhub organization
- Changed default dropEst version to 0.8.6

Version 15

- Added drop_deq_tools_prep_bam_memory and drop_deq_tools_dge_memory options

Version 14

- Fix for downloading files from user pays buckets

Version 13

- Set GCLOUD_PROJECT_ID for user pays buckets

Version 12

- Changed default dropEst memory from 52G to 104G

Version 11

- Updated formula for computing disk size for dropseq_count

Version 10

- Added option to specify merge_bam_alignment_memory and sort_bam_max_records_in_ram

Version 9

- Updated default drop_seq_tools_version from 2.2.0 to 2.3.0

Version 8

- Made additional options available for running dropEst

Version 7

- Changed default dropEst memory from 104G to 52G

Version 6

- Added option to run dropEst

Version 5

- Specify full version for bcl2fastq (2.20.0.422-2 instead of 2.20.0.422)

Version 4

- Fixed issue that prevented bcl2fastq from running

Version 3

- Set default run_bcl2fastq to false
- Create shortcuts for commonly used genomes

Version 2

- Updated QC report

Version 1

- Initial release

dropseq_bundle Terra Release Notes

Version 4

- Added create_intervals_memory and extra_star_flags inputs

Version 3

- Added extra disk space inputs
- Fixed bug that prevented creating multi-genome bundles

Version 2

- Added docker_registry input

Version 1

- Renamed sccloud to cumulus

Version 1

- Changed docker organization

Version 1

- Initial release

14.8 Demultiplex genetic-pooling/cell-hashing/nucleus-hashing sc/snRNA-Seq data

This `demultiplexing` workflow generates gene-count matrices from cell-hashing/nucleus-hashing/genetic-pooling data by demultiplexing.

In the workflow, `demuxEM` is used for analyzing cell-hashing/nucleus-hashing data, while `souporcell` and `demuxlet` are for genetic-pooling data.

14.8.1 Prepare input data and import workflow

1. Run `cellranger_workflow`

To demultiplex, you'll need raw gene count and hashtag matrices for cell-hashing/nucleus-hashing data, or raw gene count matrices and genome BAM files for genetic-pooling data. You can generate these data by running the `cellranger_workflow`.

Please refer to the cellranger_workflow tutorial for details.

When finished, you should be able to find the raw gene count matrix (e.g. `raw_gene_bc_matrices_h5.h5`), hashtag matrix (e.g. `sample_1_ADT.csv`) / genome BAM file (e.g. `possorted_genome_bam.bam`) for each sample.

2. Import `demultiplexing`

Import *demultiplexing* workflow to your workspace.

See the Terra documentation for adding a workflow. The *demultiplexing* workflow is under `Broad Methods Repository` with name "**cumulus/demultiplexing**".

Moreover, in the workflow page, click the `Export to Workspace...` button, and select the workspace to which you want to export *demultiplexing* workflow in the drop-down menu.

3. Prepare a sample sheet

3.1 Sample sheet format:

Create a sample sheet, **sample_sheet_demux.csv**, which describes the metadata for each pair of RNA and hashtag data. A brief description of the sample sheet format is listed below (**required column headers are shown in bold**).

Column	Description
OUTNAME	Output name for one pair of RNA and hashtag data. Must be unique per pair.
RNA	Google bucket url to the raw gene count matrix generated in Step 1.
TagFile/ADT	Google bucket url to the hashtag file generated in Step 1. The column name can be either *TagFile* or *ADT*, where *ADT* is to be backward compatible with sample sheets working with *cumulus/cumulus_hashing_cite_seq* workflow.
TYPE	Assay type, which can be `cell-hashing`, `nucleus-hashing`, or `genetic-pooling`.
Genotype	Google bucket url to the reference genotypes in `vcf.gz` format. This column is **not required** in the following cases: • When *TYPE* is `cell-hashing` or `nucleus-hashing`; • When *TYPE* is `genetic-pooling`, *demultiplexing_algorithm* input is `souporcell`, and user wish to run in *de novo* mode without reference genotypes, and don't need to rename cluster names by information from a known genotype vcf file.

Example:

```
OUTNAME,RNA,TagFile,TYPE,Genotype
sample_1,gs://exp/data_1/raw_gene_bc_matrices_h5.h5,gs://exp/data_1/sample_1_
↪ADT.csv,cell-hashing
sample_2,gs://exp/data_2/raw_gene_bc_matrices_h5.h5,gs://exp/data_2/sample_2_
↪ADT.csv,nucleus-hashing
sample_3,gs://exp/data_3/raw_gene_bc_matrices_h5.h5,gs://exp/data_3/
↪possorted_genome_bam.bam,genetic-pooling
sample_4,gs://exp/data_4/raw_gene_bc_matrices_h5.h5,gs://exp/data_4/
↪possorted_genome_bam.bam,genetic-pooling,gs://exp/variants/ref_genotypes.
↪vcf.gz
```

3.2 Upload your sample sheet to the workspace bucket:

Use gsutil (you already have it if you've installed Google Cloud SDK) in your unix terminal to upload your sample sheet to workspace bucket.

Example:

```
gsutil cp /foo/bar/projects/sample_sheet_demux.csv gs://fc-e0000000-0000-
↪0000-0000-000000000000/
```

14.8.2 Workflow inputs

Below are inputs for *demultiplexing* workflow. We'll first introduce global inputs, and then inputs for each of the demultiplexing tools. Notice that required inputs are in bold.

global inputs

Name	Description	Example	Default
input_sample_sheet	Input CSV file describing metadata of RNA and hashtag data pairing.	"gs://fc-e0000000-0000-0000-0000-000000000000/sample_sheet_demux.csv"	
output_directory	This is the output directory (gs url + path) for all results. There will be one folder per RNA-hashtag data pair under this directory.	"gs://fc-e0000000-0000-0000-0000-000000000000/demux_output"	
genome	Reference genome name. You should choose one from this genome reference list.	"GRCh38"	
demultiplexing_algorithm	demultiplexing algorithm to use for *genetic-pooling* data. Options: • "souporcell": Use souporcell, a reference-genotypes-free algorithm for demultiplexing droplet scRNA-Seq data. • "demuxlet": Use demuxlet, a canonical algorithm for demultiplexing droplet scRNA-Seq data.	"souporcell"	"souporcell"
min_num_genes	Only demultiplex cells/nuclei with at least <min_num_genes> expressed genes	100	100
docker_registry	Docker registry to use. Notice that docker image for Bustools is seperate. • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"
config_version	Version of config docker image to use. This docker is used for parsing the input sample sheet for downstream execution. Currently only one version is available: "0.1".	"0.1"	"0.1"
zones	Google cloud zones to consider for execution.	"us-east1-d us-west1-a us-west1-b"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
preemptible	Number of maximum preemptible tries allowed.	2	2

demuxEM inputs

Name	Description	Example	Default
demuxEM_alpha_on_samples	demuxEM parameter. The Dirichlet prior concentration parameter (alpha) on samples. An alpha value < 1.0 will make the prior sparse.	0.0	0.0
demuxEM_min_num_umis	demuxEM parameter. Only demultiplex cells/nuclei with at least <demuxEM_min_num_umis> of UMIs.	100	100
demuxEM_min_signal_hashtag	demuxEM parameter. Any cell/nucleus with less than <demuxEM_min_signal_hashtag> hashtags from the signal will be marked as unknown.	10.0	10.0
demuxEM_random_seed	demuxEM parameter. The random seed used in the KMeans algorithm to separate empty ADT droplets from others.	0	0
demuxEM_generate_diagnostic_plots	demuxEM parameter. If generate a series of diagnostic plots, including the background/signal between HTO counts, estimated background probabilities, HTO distributions of cells and non-cells, etc.	true	true
demuxEM_generate_gender_plot	demuxEM parameter. If generate violin plots using gender-specific genes (e.g. Xist). <demuxEM_generate_gender_plot> is a comma-separated list of gene names	"XIST"	
demuxEM_version	demuxEM version to use. Currently only support "0.1.4".	"0.1.4"	"0.1.4"
demuxEM_num_cpu	demuxEM parameter. Number of CPUs to request for demuxEM per pair.	8	8
demuxEM_memory	demuxEM parameter. Memory size (integer) in GB needed for demuxEM per pair.	10	10
demuxEM_disk_space	demuxEM parameter. Disk space (integer) in GB needed for demuxEM per pair.	20	20

souporcell inputs

Name	Description	Example	Default
souporcell_version	souporcell version to use. Available versions: "2020.06", "2020.03".	"2020.06"	"2020.06"
souporcell_de_novo_mode	souporcell parameter. If `true`, run souporcell in de novo mode without reference genotypes; and if a reference genotype vcf file is provided in the sample sheet, use it **only** for matching the cluster labels computed by souporcell. If `false`, run souporcell with `--known_genotypes` option using the reference genotype vcf file specified in sample sheet, and *souporcell_rename_donors* is required in this case.	true	true
souporcell_num_clusters	souporcell parameter. Number of expected clusters when doing clustering. **This needs to be set when running souporcell.**	8	
souporcell_rename_donors	souporcell parameter. A comma-separated list of donor names for renaming clusters achieved by souporcell. By default, the resulting donors are *Donor1*, *Donor2*, ...	"CB1,CB2,CB3,CB4"	
souporcell_num_cpu	souporcell parameter. Number of CPUs to request for souporcell per pair.	32	32
souporcell_memory	souporcell parameter. Memory size (integer) in GB needed for souporcell per pair.	120	120
souporcell_disk_space	souporcell parameter. Disk space (integer) in GB needed for souporcell per pair.	500	500

demuxlet inputs

Name	Description	Example	Default
demuxlet_version	demuxlet version to use. Currently only support "0.1b".	"0.1b"	"0.1b"
demuxlet_memory	demuxlet parameter. Memory size (integer) in GB needed for demuxlet per pair.	10	10
demuxlet_disk_space	demuxlet parameter. Disk space size (integer) in GB needed for demuxlet per pair. Notice that the overall disk space for demuxlet is this disk space plus the size of provided reference genotypes file in the sample sheet.	2	2

14.8.3 Workflow outputs

See the table below for *demultiplexing* workflow outputs.

Name	Type	Description
output_folders	Array[String]	A list of Google Bucket URLs of the output folders. Each folder is associated with one RNA-hashtag pair in the given sample sheet.
output_zarr_files	Array[File]	A list of demultiplexed RNA count matrices in zarr format. Each zarr file is associated with one RNA-hashtag pair in the given sample sheet. Please refere to section load demultiplexing results into Python and R for its structure.

In the output subfolder of each cell-hashing/nuclei-hashing RNA-hashtag data pair, you can find the following files:

Name	Description
output_name_demux.zarr.zip	Demultiplexed RNA count matrix in zarr format. Please refer to section load demultiplexing results into Python and R for its structure.
output_name.out.demuxEM.zarr.zip	RNA expression matrix with demultiplexed sample identities in zarr format. To load this file into Python, you need to first install Pegasusio on your local machine. Then use `import pegasusio as io; data = io.read_input("output_name.out.demuxEM.zarr.zip")` in Python environment. It contains 2 UnimodalData objects: one with key name suffix `-hashing` is the hashtag count matrix, the other one with key name suffix `-rna` is the demultiplexed RNA count matrix. To load the hashtag count matrix, type `hash_data = data.get_data('<genome>-hashing')`, where `<genome>` is the genome name of the data. The count matrix is `hash_data.X`; cell barcode attributes are stored in `hash_data.obs`; sample names are in `hash_data.var_names`. Moreover, the estimated background probability regarding hashtags is in `hash_data.uns['background_probs']`. To load the RNA matrix, type `rna_data = data.get_data('<genome>-rna')`, where `<genome>` is the genome name of the data. It only contains cells which have estimated sample assignments. The count matrix is `rna_data.X`. Cell barcode attributes are stored in `rna_data.obs`: `rna_data.obs['demux_type']` stores the estimated droplet types (singlet/doublet/unknown) of cells; `rna_data.obs['assignment']` stores the estimated hashtag(s) that each cell belongs to. Moreover, for cell-hashing/nucleus-hashing data, you can find estimated sample fractions (sample1, sample2, ..., samplen, background) for each droplet in `rna_data.obsm['raw_probs']`.
output_name.ambient_hashtag.hist.png	Optional output. A histogram plot depicting hashtag distributions of empty droplets and non-empty droplets.
output_name.background_probabilities.png	Optional output. A bar plot visualizing the estimated hashtag background probability distribution.
output_name.real_content.hist.png	Optional output. A histogram plot depicting hashtag distributions of not-real-cells and real-cells as defined by total number of expressed genes in the RNA assay.
output_name.rna_demux.hist.png	Optional output. A histogram plot depicting RNA UMI distribution for singlets, doublets and unknown cells.
output_name.gene_name.violin.png	Optional outputs. Violin plots depicting gender-specific gene expression across samples. We can have multiple plots if a gene list is provided in `demuxEM_generate_gender_plot` field of cumulus_hashing_cite_seq inputs.

In the output subfolder of each genetic-pooling RNA-hashtag data pair generated by *souporcell*, you can find the following files:

Name	Description
output_name_demux.zarr.zip	Demultiplexed RNA count matrix in zarr format. Please refer to section load demultiplexing results into Python and R for its structure.
clusters.tsv	Inferred droplet type and cluster assignment for each cell barcode.
cluster_genotypes.vcf	Inferred genotypes for each cluster.
match_donors.log	Log of matching donors step, with information of donor matching included.

In the output subfolder of each genetic-pooling RNA-hashtag data pair generated by *demuxlet*, you can find the following files:

Name	Description
output_name_demux.zarr.zip	Demultiplexed RNA count matrix in zarr format. Please refer to section load demultiplexing results into Python and R for its structure.
output_name.best	Inferred droplet type and cluster assignment for each cell barcode.

14.8.4 Load demultiplexing results into Python and R

To load demultiplexed RNA count matrix into Python, you need to install Python package pegasusio first. Then follow the codes below:

```
import pegasusio as io
data = io.read_input('output_name_demux.zarr.zip')
```

Once you load the data object, you can find estimated droplet types (singlet/doublet/unknown) in `data.obs['demux_type']`. Notices that there are cell barcodes with no sample associated, and therefore have no droplet type.

You can also find estimated sample assignments in `data.obs['assignment']`.

For cell-hashing/nucleus-hashing data, if one sample name can correspond to multiple feature barcodes, each feature barcode is assigned to a unique sample name, and this deduplicated sample assignment results are in `data.obs['assignment.dedup']`.

To load the results into R, you need to install R package `reticulate` in addition to Python package `pegasusio`. Then follow the codes below:

```
library(reticulate)
ad <- import("pegasusio", convert = FALSE)
data <- ad$read_input("output_name_demux.zarr.zip")
```

Results are in `data$obs['demux_type']`, `data$obs['assignment']`, and similarly as above, for cell-hashing/nucleus-hashing data, you'll find an additional field `data$obs['assignment.dedup']` for deduplicated sample assignment in the case that one sample name can correspond to multiple feature barcodes.

14.9 Run Cumulus for sc/snRNA-Seq data analysis

14.9.1 Run Cumulus analysis

Prepare Input Data

Case One: Sample Sheet

Follow the steps below to run **cumulus** on Terra.

1. Create a sample sheet, **count_matrix.csv**, which describes the metadata for each sample count matrix. The sample sheet should at least contain 2 columns — *Sample* and *Location*. *Sample* refers to sample names and *Location* refers to the location of the channel-specific count matrix in either of

- 10x format with v2 chemistry. For example, `gs://fc-e0000000-0000-0000-0000-000000000000/my_dir/sample_1/raw_gene_bc_matrices_h5.h5`.
- 10x format with v3 chemistry. For example, `gs://fc-e0000000-0000-0000-0000-000000000000/my_dir/sample_1/raw_feature_bc_matrices.h5`.
- Drop-seq format. For example, `gs://fc-e0000000-0000-0000-0000-000000000000/my_dir/sample_2/sample_2.umi.dge.txt.gz`.
- Matrix Market format (mtx). If the input is mtx format, location should point to a mtx file with a file suffix of either '.mtx' or '.mtx.gz'. In addition, the associated barcode and gene tsv/txt files should be located in the same folder as the mtx file. For example, if we generate mtx file using BUStools, we set *Location* to `gs://fc-e0000000-0000-0000-0000-000000000000/my_dir/mm10/cells_x_genes.mtx`. We expect to see `cells_x_genes.barcodes.txt` and `cellx_x_genes.genes.txt` under folder `gs://fc-e0000000-0000-0000-0000-000000000000/my_dir/mm10/`. We support loading mtx files in HCA DCP, 10x Genomics V2/V3, SCUMI, dropEST and BUStools format. Users can also set *Location* to a folder `gs://fc-e0000000-0000-0000-0000-000000000000/my_dir/hg19_and_mm10/`. This folder must be generated by Cell Ranger for multi-species samples and we expect one subfolder per species (e.g. 'hg19') and each subfolder should contain mtx file, barcode file, gene name file as generated by Cell Ranger.
- csv format. If it is HCA DCP csv format, we expect the expression file has the name of `expression.csv`. In addition, we expect that `cells.csv` and `genes.csv` files are located under the same folder as the `expression.csv`. For example, `gs://fc-e0000000-0000-0000-0000-000000000000/my_dir/sample_3/`.
- tsv or loom format.

Additionally, an optional Reference column can be used to select samples generated from a same reference (e.g. `mm10`). If the count matrix is in either DGE, mtx, csv, tsv, or loom format, the value in this column will be used as the reference since the count matrix file does not contain reference name information. The only exception is mtx format. If users do not provide a Reference column, we will use the basename of the folder containing the mtx file as its reference. In addition, the Reference column can be used to aggregate count matrices generated from different genome versions or gene annotations together under a unified reference. For example, if we have one matrix generated from `mm9` and the other one generated from `mm10`, we can write `mm9_10` for these two matrices in their Reference column. Pegasus will change their references to `mm9_10` and use the union of gene symbols from the two matrices as the gene symbols of the aggregated matrix. For HDF5 files (e.g. 10x v2/v3), the reference name contained in the file does not need to match the value in this column. In fact, we use this column to rename references in HDF5 files. For example, if we have two HDF files, one generated from `mm9` and the other generated from `mm10`. We can set these two files' Reference column value to `mm9_10`, which will rename their reference names into `mm9_10` and the aggregated matrix will contain all genes from either `mm9` or `mm10`. This renaming feature does not work if one HDF5 file contain multiple references (e.g. `mm10` and `GRCh38`).

You are free to add any other columns and these columns will be used in selecting channels for futher analysis. In the example below, we have *Source*, which refers to the tissue of origin, *Platform*, which refers to the sequencing platform, *Donor*, which refers to the donor ID, and *Reference*, which refers to the reference genome.

Example:

```
Sample,Source,Platform,Donor,Reference,Location
sample_1,bone_marrow,NextSeq,1,GRCh38,gs://fc-e0000000-0000-0000-0000-
↪000000000000/my_dir/sample_1/raw_gene_bc_matrices_h5.h5
sample_2,bone_marrow,NextSeq,2,GRCh38,gs://fc-e0000000-0000-0000-0000-
↪000000000000/my_dir/sample_2/raw_gene_bc_matrices_h5.h5
sample_3,pbmc,NextSeq,1,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/
↪my_dir/sample_3/raw_feature_bc_matrices.h5
sample_4,pbmc,NextSeq,2,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/
↪my_dir/sample_4/raw_feature_bc_matrices.h5
```

If you ran **cellranger_workflow** ahead, you should already obtain a template **count_matrix.csv** file that you can modify from **generate_count_config**'s outputs.

1. Upload your sample sheet to the workspace.

   Example:

   ```
   gsutil cp /foo/bar/projects/my_count_matrix.csv gs://fc-e0000000-0000-
   ↪0000-0000-000000000000/
   ```

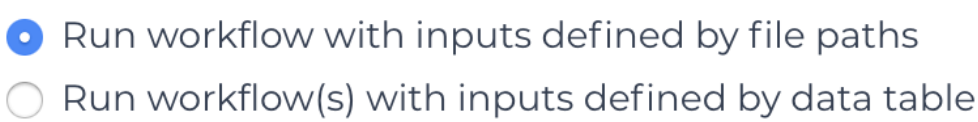
   where `/foo/bar/projects/my_count_matrix.csv` is the path to your sample sheet in local machine, and `gs://fc-e0000000-0000-0000-0000-000000000000/` is the location on Google bucket to hold it.

2. Import *cumulus* workflow to your workspace.

   See the Terra documentation for adding a workflow. The *cumulus* workflow is under `Broad Methods Repository` with name "**cumulus/cumulus**".

   Moreover, in the workflow page, click the `Export to Workspace...` button, and select the workspace to which you want to export *cumulus* workflow in the drop-down menu.

3. In your workspace, open `cumulus` in `WORKFLOWS` tab. Select `Run workflow with inputs defined by file paths` as below

   and click the `SAVE` button.

Case Two: Single File

Alternatively, if you only have one single count matrix for analysis, you can go without sample sheets. **Cumulus** currently supports the following formats:

- 10x genomics v2/v3 format (hdf5);
- Drop-seq dge format;
- csv (no HCA DCP format), tsv or loom formats.

Simply upload your data to the Google Bucket of your workspace, and specify its URL in `input_file` field of Cumulus' global inputs (see below). For hdf5 files, there is no need to specify genome names. For other formats, you can specify genome name in `considered_refs` field in cluster inputs; otherwise, default name `''` will be used.

In this case, the **aggregate_matrices** step will be skipped.

Case Three: Multiple samples without aggregation

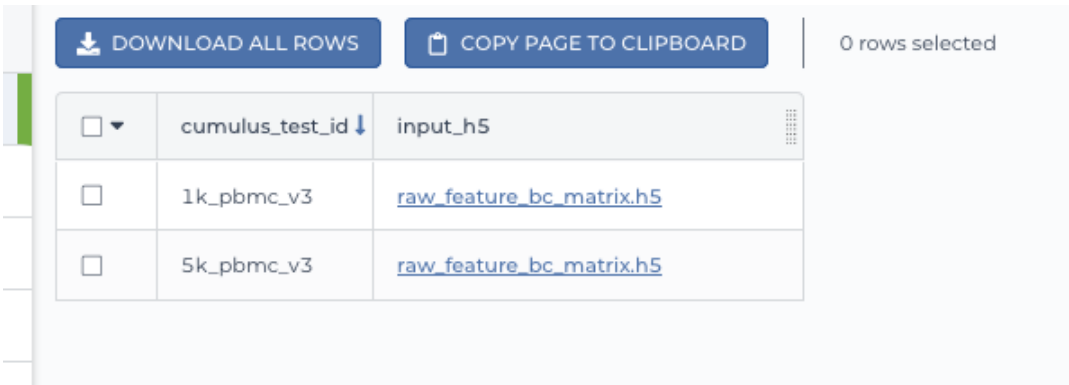
Sometimes, you may want to run Cumulus on multiple samples simultaneously. This is different from Case one, because samples are analyzed separately without aggregation.

1. To do it, you need to first create a data table on Terra. An example TSV file is the following:

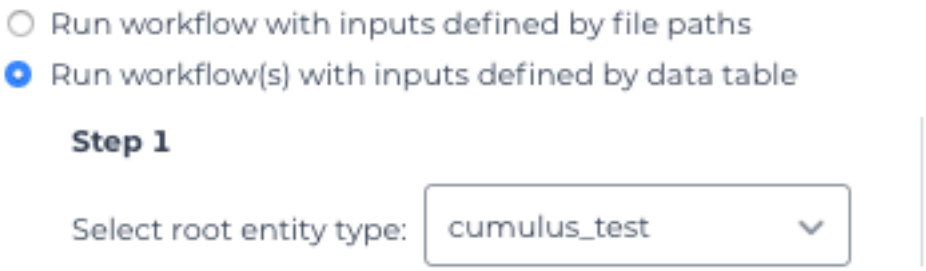
```
entity:cumulus_test_id  input_h5
5k_pbmc_v3  gs://fc-e0000000-0000-0000-0000-000000000000/5k_pbmc_v3/raw_feature_
↪bc_matrix.h5
1k_pbmc_v3  gs://fc-e0000000-0000-0000-0000-000000000000/1k_pbmc_v3/raw_feature_
↪bc_matrix.h5
```

You are free to add more columns, but sample ids and URLs to RNA count matrix files are required. I'll use this example TSV file for the rest of steps in this case.

1. Upload your TSV file to your workspace. Open the `DATA` tab on your workspace. Then click the upload button on left `TABLE` panel, and select the TSV file above. When uploading is done, you'll see a new data table with name "cumulus_test":



2. Import *cumulus* workflow to your workspace as in Case one. Then open `cumulus` in `WORKFLOW` tab. Select `Run workflow(s) with inputs defined by data table`, and choose *cumulus_test* from the drop-down menu.



3. In the input field, specify:

- `input_file`: Type `this.input_h5`, where `this` refers to the data table selected, and `input_h5` is the column name in this data table for RNA count matrices.
- `output_directory`: Type Google bucket URL for the main output folder. For example, `gs://fc-e0000000-0000-0000-0000-000000000000/cumulus_results`.
- `output_name`: Type `this.cumulus_test_id`, where `cumulus_test_id` is the column name in data table for sample ids.

An example is in the screen shot below:

Task name	Variable	Type	Attribute
cumulus	input_file	File	this.input_h5
cumulus	output_directory	String	"gs://fc-e0000000-0000-0000-0000-000000000000/cumulus_results"
cumulus	output_name	String	this.cumulus_test_id

Then finish setting up other inputs following the description in sections below. When you are done, click `SAVE`, and then `RUN ANALYSIS`.

When all the jobs are done, you'll find output for the 2 samples in subfolders `gs://fc-e0000000-0000-0000-0000-000000000000/cumulus_results/5k_pbmc_v3` and `gs://fc-e0000000-0000-0000-0000-000000000000/cumulus_results/1k_pbmc_v3`, respectively.

Cumulus steps:

Cumulus processes single cell data in the following steps:

1. **aggregate_matrices** (optional). When given a CSV format sample sheet, this step aggregates channel-specific count matrices into one big count matrix. Users can specify which channels they want to analyze and which sample attributes they want to import to the count matrix in this step. Otherwise, if a single count matrix file is given, skip this step.
2. **cluster**. This is the main analysis step. In this step, **Cumulus** performs low quality cell filtration, highly variable gene selection, batch correction, dimension reduction, diffusion map calculation, graph-based clustering and 2D visualization calculation (e.g. t-SNE/UMAP/FLE).
3. **de_analysis**. This step is optional. In this step, **Cumulus** can calculate potential markers for each cluster by performing a variety of differential expression (DE) analysis. The available DE tests include Welch's t test, Fisher's exact test, and Mann-Whitney U test. **Cumulus** can also calculate the area under ROC (AUROC) curve values for putative markers. If `find_markers_lightgbm` is on, **Cumulus** will try to identify cluster-specific markers by training a LightGBM classifier. If the samples are human or mouse immune cells, **Cumulus** can also optionally annotate putative cell types for each cluster based on known markers.
4. **plot**. This step is optional. In this step, **Cumulus** can generate 6 types of figures based on the **cluster** step results:
   - **composition** plots which are bar plots showing the cell compositions (from different conditions) for each cluster. This type of plots is useful to fast assess library quality and batch effects.
   - **tsne**, **fitsne**, and **net_tsne**: t-SNE like plots based on different algorithms, respectively. Users can specify cell attributes (e.g. cluster labels, conditions) for coloring side-by-side.
   - **umap** and **net_umap**: UMAP like plots based on different algorithms, respectively. Users can specify cell attributes (e.g. cluster labels, conditions) for coloring side-by-side.
   - **fle** and **net_fle**: FLE (Force-directed Layout Embedding) like plots based on different algorithms, respectively. Users can specify cell attributes (e.g. cluster labels, conditions) for coloring side-by-side.
   - **diffmap** plots which are 3D interactive plots showing the diffusion maps. The 3 coordinates are the first 3 PCs of all diffusion components.
   - If input is CITE-Seq data, there will be **citeseq_fitsne** plots which are FIt-SNE plots based on epitope expression.
5. **cirro_output**. This step is optional. Generate Cirrocumulus inputs for visualization using Cirrocumulus .
6. **scp_output**. This step is optional. Generate analysis result in Single Cell Portal (SCP) compatible format.

7. **organize_results**. Copy analysis results from execution environment to destination location on Google bucket. The output organization is as follows: one top-level output folder specified by `output_directory` in global inputs; each sample has all it output files in a distinct subfolder, with name specified by `output_name` in global inputs; within this subfolder, each file has a common filename prefix specified by `output_name`.

In the following sections, we will first introduce global inputs and then introduce the WDL inputs and outputs for each step separately. But please note that you need to set inputs from all steps simultaneously in the Terra WDL.

Note that we will make the required inputs/outputs bold and all other inputs/outputs are optional.

---

global inputs

Name	Description	Example	Default
input_file	Input CSV sample sheet describing metadata of each 10x channel, or a single input count matrix file	"gs://fc-e0000000-0000-0000-0000-000000000000/my_count_matrix.csv"	
output_directory	Google bucket URL of the output directory.	"gs://fc-e0000000-0000-0000-0000-000000000000/my_results_dir"	
output_name	This is the name of subdirectory for the current sample; and all output files within the subdirectory will have this string as the common filename prefix.	"my_sample"	
cumulus_version	Cumulus version to use. Versions available: 1.0.0, 0.16.0, 0.15.0, 0.13.0, 0.12.0, 0.11.0, 0.10.0.	"1.0.0"	"1.0.0"
docker_registry	Docker registry to use. Options: • "cumulusprod" for Docker Hub images; • "quay.io/cumulus" for backup images on Red Hat registry.	"cumulusprod"	"cumulusprod"
zones	Google cloud zones to consider for execution.	"us-east1-d us-west1-a us-west1-b"	"us-central1-a us-central1-b us-central1-c us-central1-f us-east1-b us-east1-c us-east1-d us-west1-a us-west1-b us-west1-c"
num_cpu	Number of CPUs per Cumulus job	32	64
memory	Memory size string	"200G"	"200G"
disk_space	Total disk space in GB	100	100
preemptible	Number of preemptible tries	2	2

---

aggregate_matrices

aggregate_matrices inputs

Name	Description	Example	Default
restrictions	Select channels that satisfy all restrictions. Each restriction takes the format of name:value,...,value. Multiple restrictions are separated by ';'	"Source:bone_marrow;Platform:NextSeq"	
attributes	Specify a comma-separated list of outputted attributes. These attributes should be column names in the count_matrix.csv file	"Source,Platform,Donor"	
default_reference	If sample count matrix is in either DGE, mtx, csv, tsv or loom format and there is no Reference column in the csv_file, use default_reference as the reference.	"GRCh38"	
select_only_singlets	If we have demultiplexed data, turning on this option will make cumulus only include barcodes that are predicted as singlets.	true	false
minimum_number_of_genes	Only keep barcodes with at least this number of expressed genes	100	100

aggregate_matrices output

Name	Type	Description
output_aggr_zarr	File	Aggregated count matrix in Zarr format

---

cluster

cluster inputs

Name	Description	Example	Default
focus	Focus analysis on Unimodal data with <keys>. <keys> is a comma-separated list of keys. If None, the `self._selected` will be the focused one. Focus key consists of two parts: reference genome name, and data type, connected with a hyphen marker "-". Reference genome name depends on the reference you used when running Cellranger workflow. See here for reference list.	"GRCh38-rna"	

Continued on next page

Table 1 – continued from previous page

Name	Description	Example	Default
append	Append Unimodal data <key> to any <keys> in *focus*. Similarly as focus keys, append key also consists of two parts: reference genome name, and data type, connected with a hyphen marker "–". See here for reference genome list.	"SARSCoV2-rna"	
channel	Specify the cell barcode attribute to represent different samples.	"Donor"	
black_list	Cell barcode attributes in black list will be poped out. Format is "attr1,attr2,...,attrn".	"attr1,attr2,attr3""	
min_genes_before_filtration	If raw data matrix is input, empty barcodes will dominate pre-filtration statistics. To avoid this, for raw data matrix, only consider barcodes with at lease <min_genes_before_filtration> genes for pre-filtration condition.	100	100
select_only_singlets	If we have demultiplexed data, turning on this option will make cumulus only include barcodes that are predicted as singlets	false	false
remap_singlets	For demultiplexed data, user can remap singlet names using assignment in String in this input. This string assignment takes the format "new_name_i:old_name_1,old_name_2;new_name_ii:old_name_3;...". For example, if we hashed 5 libraries from 3 samples: sample1_lib1, sample1_lib2; sample2_lib1, sample2_lib2; sample3, we can remap them to 3 samples using this string: `"sample1:sample1_lib1,sample1_lib2;sample2:sample2_lib1,sample2_lib2"`. In this way, the new singlet names will be in metadata field with key `assignment`, while the old names are kept in metadata with key `assignment.orig`.	"Group1:CB1,CB2;Group2:CB3,CB4,CB5"	
subset_singlets	For demultiplexed data, user can use this input to choose a subset of singlets based on their names. This string takes the format "name1,name2,...". Note that if `remap_singlets` is specified, subsetting happens after remapping, i.e. you should use the new singlet names for choosing subset.	"Group2,CB6,CB7"	
output_filtration_results	If write cell and gene filtration results to a spreadsheet	true	true
plot_filtration_plots	If plot filtration results as PDF files	true	true
plot_filtration_figsize	Figure size for filtration plots. <figsize> is a comma-separated list of two numbers, the width and height of the figure (e.g. 6,4)	6,4	
output_h5ad	Generate Seurat-compatible h5ad file.	true	true
output_loom	If generate loom-formatted file	false	false
min_genes	Only keep cells with at least <min_genes> of genes	500	500

Continued on next page

Table 1 – continued from previous page

Name	Description	Example	Default
max_genes	Only keep cells with less than <max_genes> of genes	6000	6000
min_umis	Only keep cells with at least <min_umis> of UMIs. By default, don't filter cells due to UMI lower bound.	100	
max_umis	Only keep cells with less than <max_umis> of UMIs. By default, don't filter cells due to UMI upper bound.	600000	
mito_prefix	Prefix of mitochondrial gene names. This is to identify mitochondrial genes.	"mt-"	"MT-" for *GRCh38* reference genome data; "mt-" for *mm10* reference genome data; for other reference genome data, must specify this prefix manually.
percent_mito	Only keep cells with mitochondrial ratio less than <percent_mito>% of total counts	50	20.0
gene_percent_cells	Only use genes that are expressed in at <gene_percent_cells>% of cells to select variable genes	50	0.05
counts_per_cell_after	Total counts per cell after normalization, before transforming the count matrix into Log space.	1e5	1e5
select_hvf_flavor	Highly variable feature selection method. Options: • "pegasus": New selection method proposed in Pegasus, the analysis module of Cumulus workflow. • "Seurat": Conventional selection method used by Seurat and SCANPY.	"pegasus"	"pegasus"
select_hvf_ngenes	Select top <select_hvf_ngenes> highly variable features. If <select_hvf_flavor> is "Seurat" and <select_hvf_ngenes> is "None", select HVGs with z-score cutoff at 0.5.	2000	2000
no_select_hvf	Do not select highly variable features.	false	false
correct_batch_effect	If correct batch effects	false	false

Continued on next page

Table 1 – continued from previous page

Name	Description	Example	Default
correction_method	Batch correction method. Options: • "harmony": Harmony algorithm (Korsunsky et al. Nature Methods 2019). • "L/S": Location/Scale adjustment algorithm (Li and Wong. The analysis of Gene Expression Data, 2003). • "scanorama": Scanorama algorithm (Hie et al. Nature Biotechnology 2019).	"harmony"	"harmony"
batch_group_by	Batch correction assumes the differences in gene expression between channels are due to batch effects. However, in many cases, we know that channels can be partitioned into several groups and each group is biologically different from others. In this case, we will only perform batch correction for channels within each group. This option defines the groups. If <expression> is None, we assume all channels are from one group. Otherwise, groups are defined according to <expression>. <expression> takes the form of either 'attr', or 'attr1+attr2+...+attrn', or 'attr=value11,...,value1n_1;value21,...,value2n_2;...;valuem1,...,valuemn_m'. In the first form, 'attr' should be an existing sample attribute, and groups are defined by 'attr'. In the second form, 'attr1',...,'attrn' are n existing sample attributes and groups are defined by the Cartesian product of these n attributes. In the last form, there will be m + 1 groups. A cell belongs to group i (i > 0) if and only if its sample attribute 'attr' has a value among valuei1,...,valuein_i. A cell belongs to group 0 if it does not belong to any other groups	"Donor"	None
random_state	Random number generator seed	0	0
calc_signature_sets	Gene set for calculating signature scores. It can be either of the following forms: • String chosen from: "cell_cycle_human", "cell_cycle_mouse", "gender_human", "gender_mouse", "mitochondrial_genes_human", "mitochondrial_genes_mouse", "ribosomal_genes_human", and "ribosomal_genes_mouse". • Google bucket URL of a GMT format file. For example: "gs://fc-e0000000-0000-0000-0000-000000000000/cell_cycle_sig.gmt".	"cell_cycle_human"	
nPC	Number of principal components	50	50

Continued on next page

Table 1 – continued from previous page

Name	Description	Example	Default
knn_K	Number of nearest neighbors used for constructing affinity matrix.	50	100
knn_full_speed	For the sake of reproducibility, we only run one thread for building kNN indices. Turn on this option will allow multiple threads to be used for index building. However, it will also reduce reproducibility due to the racing between multiple threads.	false	false
run_diffmap	Whether to calculate diffusion map or not. It will be automatically set to `true` when input **run_fle** or **run_net_fle** is set.	false	false
diffmap_ndc	Number of diffusion components	100	100
diffmap_maxt	Maximum time stamp in diffusion map computation to search for the knee point.	5000	5000
run_louvain	Run Louvain clustering algorithm	true	true
louvain_resolution	Resolution parameter for the Louvain clustering algorithm	1.3	1.3
louvain_class_label	Louvain cluster label name in analysis result.	"louvain_labels"	"louvain_labels"
run_leiden	Run Leiden clustering algorithm.	false	false
leiden_resolution	Resolution parameter for the Leiden clustering algorithm.	1.3	1.3
leiden_niter	Number of iterations of running the Leiden algorithm. If negative, run Leiden iteratively until no improvement.	2	-1
leiden_class_label	Leiden cluster label name in analysis result.	"leiden_labels"	"leiden_labels"
run_spectral_louvain	Run Spectral Louvain clustering algorithm	false	false
spectral_louvain_basis	Basis used for KMeans clustering. Use diffusion map by default. If diffusion map is not calculated, use PCA coordinates. Users can also specify "pca" to directly use PCA coordinates.	"diffmap"	"diffmap"
spectral_louvain_resolution	Resolution parameter for louvain.	1.3	1.3
spectral_louvain_class_label	Spectral louvain label name in analysis result.	"spectral_louvain_labels"	"spectral_louvain_labels"
run_spectral_leiden	Run Spectral Leiden clustering algorithm.	false	false
spectral_leiden_basis	Basis used for KMeans clustering. Use diffusion map by default. If diffusion map is not calculated, use PCA coordinates. Users can also specify "pca" to directly use PCA coordinates.	"diffmap"	"diffmap"
spectral_leiden_resolution	Resolution parameter for leiden.	1.3	1.3
spectral_leiden_class_label	Spectral leiden label name in analysis result.	"spectral_leiden_labels"	"spectral_leiden_labels"
run_tsne	Run multi-core t-SNE for visualization	false	false
tsne_perplexity	t-SNE's perplexity parameter, also used by FIt-SNE.	30	30
run_fitsne	Run FIt-SNE for visualization	false	false
run_umap	Run UMAP for visualization	true	true
umap_K	K neighbors for UMAP.	15	15
umap_min_dist	UMAP parameter.	0.5	0.5
umap_spread	UMAP parameter.	1.0	1.0
run_fle	Run force-directed layout embedding (FLE) for visualization	false	false
fle_K	Number of neighbors for building graph for FLE	50	50
fle_target_change_per_node	Target change per node to stop FLE.	2.0	2.0
fle_target_steps	Maximum number of iterations before stopping the algoritm	5000	5000

Continued on next page

Table 1 – continued from previous page

Name	Description	Example	Default
net_down_sample_rate	Down sampling fraction for net-related visualization	0.1	0.1
run_net_tsne	Run Net tSNE for visualization	false	false
net_tsne_out_basis	Basis name for Net t-SNE coordinates in analysis result	“net_tsne”	“net_tsne”
run_net_umap	Run Net UMAP for visualization	false	false
net_umap_out_basis	Basisname for Net UMAP coordinates in analysis result	“net_umap”	“net_umap”
run_net_fle	Run Net FLE for visualization	false	false
net_fle_out_basis	Basis name for Net FLE coordinates in analysis result.	“net_fle”	“net_fle”

cluster outputs

Name	Type	Description
output_zarr	File	Output file in zarr format (output_name.zarr.zip). To load this file in Python, you need to first install PegasusIO on your local machine. Then use `import pegasusio as io; data = io.read_input('output_name.zarr.zip')` in Python environment. `data` is a *MultimodalData* object, and points to its default *UnimodalData* element. You can set its default *UnimodalData* to others by `data.set_data(focus_key)` where `focus_key` is the key string to the wanted *UnimodalData* element. For its default *UnimodalData* element, the log-normalized expression matrix is stored in `data.X` as a Scipy CSR-format sparse matrix, with cell-by-gene shape. The `obs` field contains cell related attributes, including clustering results. For example, `data.obs_names` records cell barcodes; `data.obs['Channel']` records the channel each cell comes from; `data.obs['n_genes']`, `data.obs['n_counts']`, and `data.obs['percent_mito']` record the number of expressed genes, total UMI count, and mitochondrial rate for each cell respectively; `data.obs['louvain_labels']`, `data.obs['leiden_labels']`, `data.obs['spectral_louvain_labels']`, and `data.obs['spectral_leiden_labels']` record each cell's cluster labels using different clustering algorithms; The `var` field contains gene related attributes. For example, `data.var_names` records gene symbols, `data.var['gene_ids']` records Ensembl gene IDs, and `data.var['highly_variable_features']` records selected variable genes. The `obsm` field records embedding coordinates. For example, `data.obsm['X_pca']` records PCA coordinates, `data.obsm['X_tsne']` records t-SNE coordinates, `data.obsm['X_umap']` records UMAP coordinates, `data.obsm['X_diffmap']` records diffusion map coordinates, `data.obsm['X_diffmap_pca']` records the first 3 PCs by projecting the diffusion components using PCA, and `data.obsm['X_fle']` records the force-directed layout coordinates from the diffusion components. The `uns` field stores other related information, such as reference genome (`data.uns['genome']`), kNN on PCA coordinates (`data.uns['pca_knn_indices']` and `data.uns['pca_knn_distances']`), etc.
output_log	File	This is a copy of the logging module output, containing important intermediate messages
output_h5ad	Array[File]	List of output file(s) in Seurat-compatible h5ad format (output_name.focus_key.h5ad), in which each file is associated with a focus of the input data. To load this file in Python, first install PegasusIO on your local machine. Then use `import pegasusio as io; data = io.read_input('output_name.focus_key.h5ad')` in Python environment.

de_analysis

de_analysis inputs

Name	Description	Example	Default
perform_de_analysis	Whether perform differential expression (DE) analysis. If performing, by default calculate AUROC scores and Mann-Whitney U test.	true	true
cluster_labels	Specify the cluster label used for DE analysis	"louvain_labels"	"louvain_labels"
alpha	Control false discovery rate at <alpha>	0.05	0.05
fisher	Calculate Fisher's exact test	true	true
t_test	Calculate Welch's t-test.	true	true
find_markers_lightgbm	If also detect markers using LightGBM	false	false
remove_ribo	Remove ribosomal genes with either RPL or RPS as prefixes. Currently only works for human data	false	false
min_gain	Only report genes with a feature importance score (in gain) of at least <gain>	1.0	1.0
annotate_cluster	If also annotate cell types for clusters based on DE results	false	false
annotate_de_test	Differential Expression test to use for inference on cell types. Options: "mwu", "t", or "fisher"	"mwu"	"mwu"
organism	Organism, could either be "human_immune", "mouse_immune", "human_brain", "mouse_brain", "human_lung", or a Google bucket link to a JSON file describing the markers	"mouse_brain"	"human_immune"
minimum_report_score	Minimum cell type score to report a potential cell type	0.5	0.5

de_analysis outputs

Name	Type	Description
output_de_h5ad	Array[File]	List of h5ad-formatted results with DE results updated (output_name.focus_key.h5ad), in which each file is associated with a focus of the input data. To load this file in Python, you need to first install PegasusIO on your local machine. Then type `import pegasusio as io; data = io.read_input('output_name.focus_key.h5ad')` in Python environment. After loading, `data` has the similar structure as *UnimodalData* object in Description of **output_zarr** in cluster outputs section. Besides, there is one additional field `varm` which records DE analysis results in `data.varm['de_res']`. You can use Pandas DataFrame to convert it into a reader-friendly structure: `import pandas as pd; df = pd.DataFrame(data.varm['de_res'], index = data.var_names)`. Then in the resulting data frame, genes are rows, and those DE test statistics are columns. DE analysis in cumulus is performed on each cluster against cells in all the other clusters. For instance, in the data frame, column `mean_logExpr:1` refers to the mean expression of genes in log-scale for cells in Cluster 1. The number after colon refers to the cluster label to which this statistic belongs.
output_de_xlsx	Array[File]	List of spreadsheets reporting DE results (output_name.focus_key.de.xlsx), in which each file is associated with a focus of the input data. Each cluster has two tabs: one for up-regulated genes for this cluster, one for down-regulated ones. In each tab, genes are ranked by AUROC scores. Genes which are not significant in terms of q-values in any of the DE test are not included (at false discovery rate specified in **alpha** field of de_analysis inputs).
output_markers_xlsx	Array[File]	List of Excel spreadsheets containing detected markers (output_name.focus_key.markers.xlsx), in which each file is associated with a focus of the input data. Each cluster has one tab in the spreadsheet and each tab has three columns, listing markers that are strongly up-regulated, weakly up-regulated and down-regulated.
output_anno_file	Array[File]	List of cluster-based cell type annotation files (output_name.focus_key.anno.txt), in which each file is associated with a focus of the input data.

How cell type annotation works

In this subsection, we will describe the format of input JSON cell type marker file, the *ad hoc* cell type inference algorithm, and the format of the output putative cell type file.

JSON file

The top level of the JSON file is an object with two name/value pairs:

- **title**: A string to describe what this JSON file is for (e.g. "Mouse brain cell markers").
- **cell_types**: List of all cell types this JSON file defines. In this list, each cell type is described using a separate object with 2 to 3 name/value pairs:
  - **name**: Cell type name (e.g. "GABAergic neuron").
  - **markers**: List of gene-marker describing objects, each of which has 2 name/value pairs:
    - **genes**: List of positive and negative gene markers (e.g. `["Rbfox3+", "Flt1-"]`).
    - **weight**: A real number between `0.0` and `1.0` to describe how much we trust the markers in **genes**.

    All markers in **genes** share the weight evenly. For instance, if we have 4 markers and the weight is 0.1, each marker has a weight of `0.1 / 4 = 0.025`.

    The weights from all gene-marker describing objects of the same cell type should sum up to 1.0.
  - **subtypes**: Description on cell subtypes for the cell type. It has the same structure as the top level JSON object.

See below for an example JSON snippet:

```
{
  "title" : "Mouse brain cell markers",
    "cell_types" : [
      {
        "name" : "Glutamatergic neuron",
        "markers" : [
          {
            "genes" : ["Rbfox3+", "Reln+", "Slc17a6+", "Slc17a7+"],
            "weight" : 1.0
          }
        ],
        "subtypes" : {
          "title" : "Glutamatergic neuron subtype markers",
            "cell_types" : [
              {
                "name" : "Glutamatergic layer 4",
                "markers" : [
                  {
                    "genes" : ["Rorb+", "Paqr8+"],
                    "weight" : 1.0
                  }
                ]
              }
            ]
        }
      }
    ]
}
```

Inference Algorithm

We have already calculated the up-regulated and down-regulated genes for each cluster in the differential expression analysis step.

First, load gene markers for each cell type from the JSON file specified, and exclude marker genes, along with their associated weights, that are not expressed in the data.

Then scan each cluster to determine its putative cell types. For each cluster and putative cell type, we calculate a score between `0` and `1`, which describes how likely cells from the cluster are of this cell type. The higher the score is, the more likely cells are from the cell type.

To calculate the score, each marker is initialized with a maximum impact value (which is `2`). Then do case analysis as follows:

- For a positive marker:
  - If it is not up-regulated, its impact value is set to `0`.
  - Otherwise, if it is up-regulated:
    - If it additionally has a fold change in percentage of cells expressing this marker (within cluster vs. out of cluster) no less than `1.5`, it has an impact value of `2` and is recorded as a **strong supporting marker**.
    - If its fold change (`fc`) is less than `1.5`, this marker has an impact value of `1 + (fc - 1) / 0.5` and is recorded as a **weak supporting marker**.
- For a negative marker:
  - If it is up-regulated, its impact value is set to `0`.
  - If it is neither up-regulated nor down-regulated, its impact value is set to `1`.
  - Otherwise, if it is down-regulated:
    - If it additionally has `1 / fc` (where `fc` is its fold change) no less than `1.5`, it has an impact value of `2` and is recorded as a **strong supporting marker**.
    - If `1 / fc` is less than `1.5`, it has an impact value of `1 + (1 / fc - 1) / 0.5` and is recorded as a **weak supporting marker**.

The score is calculated as the weighted sum of impact values weighted over the sum of weights multiplied by 2 from all expressed markers. If the score is larger than 0.5 and the cell type has cell subtypes, each cell subtype will also be evaluated.

Output annotation file

For each cluster, putative cell types with scores larger than `minimum_report_score` will be reported in descending order with respect to their scores. The report of each putative cell type contains the following fields:

- **name**: Cell type name.
- **score**: Score of cell type.
- **average marker percentage**: Average percentage of cells expressing marker within the cluster between all positive supporting markers.
- **strong support**: List of strong supporting markers. Each marker is represented by a tuple of its name and percentage of cells expressing it within the cluster.
- **weak support**: List of week supporting markers. It has the same structure as **strong support**.

plot

The h5ad file contains a default cell attribute `Channel`, which records which channel each that single cell comes from. If the input is a CSV format sample sheet, `Channel` attribute matches the `Sample` column in the sample sheet. Otherwise, it's specified in `channel` field of the cluster inputs.

Other cell attributes used in plot must be added via `attributes` field in the `aggregate_matrices` inputs.

plot inputs

Name	Description	Example	Default
plot_composition	Takes the format of "label:attr,label:attr,...,label:attr". If non-empty, generate composition plot for each "label:attr" pair. "label" refers to cluster labels and "attr" refers to sample conditions	"louvain_labels:Donor"	None
plot_fitsne	Takes the format of "attr,attr,...,attr". If non-empty, plot attr colored FIt-SNEs side by side	"louvain_labels,Donor"	None
plot_tsne	Takes the format of "attr,attr,...,attr". If non-empty, plot attr colored t-SNEs side by side	"louvain_labels,Channel"	None
plot_umap	Takes the format of "attr,attr,...,attr". If non-empty, plot attr colored UMAP side by side	"louvain_labels,Donor"	None
plot_fle	Takes the format of "attr,attr,...,attr". If non-empty, plot attr colored FLE (force-directed layout embedding) side by side	"louvain_labels,Donor"	None
plot_net_tsne	Takes the format of "attr,attr,...,attr". If non-empty, plot attr colored t-SNEs side by side based on net t-SNE result.	"leiden_labels,Channel"	None
plot_net_umap	Takes the format of "attr,attr,...,attr". If non-empty, plot attr colored UMAP side by side based on net UMAP result.	"leiden_labels,Donor"	None
plot_net_fle	Takes the format of "attr,attr,...,attr". If non-empty, plot attr colored FLE (force-directed layout embedding) side by side based on net FLE result.	"leiden_labels,Donor"	None

plot outputs

Name	Type	Description
output_pdfs	Array[File]	Outputted pdf files
output_htmls	Array[File]	Outputted html files

Generate input files for Cirrocumulus

Generate Cirrocumulus inputs for visualization using Cirrocumulus .

cirro_output inputs

Name	Description	Example	Default
generate_cirro_output	Whether to generate input files for Cirrocumulus	false	false

cirro_output outputs

Name	Type	Description
output_cirro_path	Google Bucket URL	Path to Cirrocumulus inputs

---

Generate SCP-compatible output files

Generate analysis result in Single Cell Portal (SCP) compatible format.

scp_output inputs

Name	Description	Example	Default
generate_scp_output	Whether to generate SCP format output or not.	false	false
output_dense	Output dense expression matrix, instead of the default sparse matrix format.	false	false

scp_output outputs

Name	Type	Description
output_scp_files	Array[File]	Outputted SCP format files.

14.9.2 Run CITE-Seq analysis

To run CITE-Seq analysis, add `"citeseq"` string to *focus* field in cluster inputs of cumulus workflow.

An embedding of epitope expressions via FIt-SNE is available at basis `X_citeseq_fitsne`.

To plot this epitope embedding, specify attributes to plot in `plot_citeseq_fitsne` field of cluster inputs.

14.9.3 Run subcluster analysis

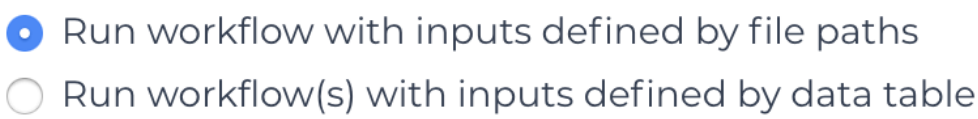
Once we have **cumulus** outputs, we could further analyze a subset of cells by running **cumulus_subcluster**. To run **cumulus_subcluster**, follow the following steps:

1. Import **cumulus_subcluster** method.

   See the Terra documentation for adding a workflow. The cumulus workflow is under `Broad Methods Repository` with name "**cumulus/cumulus_subcluster**".

   Moreover, in the workflow page, click the `Export to Workspace...` button, and select the workspace to which you want to export cumulus workflow in the drop-down menu.

2. In your workspace, open `cumulus_subcluster` in `WORKFLOWS` tab. Select `Run workflow with inputs defined by file paths` as below

   and click the `SAVE` button.

cumulus_subcluster steps:

cumulus_subcluster processes the subset of single cells in the following steps:

1. **subcluster**. In this step, **cumulus_subcluster** first select the subset of cells from **cumulus** outputs according to user-provided criteria. It then performs batch correction, dimension reduction, diffusion map calculation, graph-based clustering and 2D visualization calculation (e.g. t-SNE/UMAP/FLE).
2. **de_analysis** (optional). In this step, **cumulus_subcluster** calculates potential markers for each cluster by performing a variety of differential expression (DE) analysis. The available DE tests include Welch's t test, Fisher's exact test, and Mann-Whitney U test. **cumulus_subcluster** can also calculate the area under ROC curve (AUROC) values for putative markers. If the samples are human or mouse immune cells, **cumulus_subcluster** can optionally annotate putative cell types for each cluster based on known markers.
3. **plot** (optional). In this step, **cumulus_subcluster** can generate the following 5 types of figures based on the **subcluster** step results:
   - **composition** plots which are bar plots showing the cell compositions (from different conditions) for each cluster. This type of plots is useful to fast assess library quality and batch effects.
   - **tsne**, **fitsne**, and **net_tsne**: t-SNE like plots based on different algorithms, respectively. Users can specify different cell attributes (e.g. cluster labels, conditions) for coloring side-by-side.
   - **umap** and **net_umap**: UMAP like plots based on different algorithms, respectively. Users can specify different cell attributes (e.g. cluster labels, conditions) for coloring side-by-side.

- **fle** and **net_fle**: FLE (Force-directed Layout Embedding) like plots based on different algorithms, respectively. Users can specify different cell attributes (e.g. cluster labels, conditions) for coloring side-by-side.
- **diffmap** plots which are 3D interactive plots showing the diffusion maps. The 3 coordinates are the first 3 PCs of all diffusion components.

cumulus_subcluster's inputs

cumulus_subcluster shares many inputs/outputs with **cumulus**, we will only cover inputs/outputs that are specific to **cumulus_subcluster** in this section.

Note that we will make the required inputs/outputs bold and all other inputs/outputs are optional.

Name	Description	Example	Default
input_h5ad	Google bucket URL of input h5ad file containing *cumulus* results	"gs://fc-e0000000-0000-0000-0000-000000000000/my_results_dir/my_results.h5ad"	
output_name	This is the prefix for all output files. It should contain the Google bucket URL, subdirectory name and output name prefix	"gs://fc-e0000000-0000-0000-0000-000000000000/my_results_dir/my_results_sub"	
subset_selections	Specify which cells will be included in the subcluster analysis. This field contains one or more <subset_selection> strings separated by ';'. Each <subset_selection> string takes the format of 'attr:value,...,value', which means select cells with attr in the values. If multiple <subset_selection> strings are specified, the subset of cells selected is the intersection of these strings	"louvain_labels:3,6" or "louvain_labels:3,6;Donor:1,2"	
calculate_pseudotime	Calculate diffusion-based pseudotimes based on <roots>. <roots> should be a comma-separated list of cell barcodes	"sample_1-ACCCGGGTTT-1,sample_1-TCCCGGGAAA-2"	None
num_cpu	Number of cpus per cumulus job	32	64
memory	Memory size string	"200G"	"200G"
disk_space	Total disk space in GB	100	100
preemptible	Number of preemptible tries	2	2

For other **cumulus_subcluster** inputs, please refer to cumulus cluster inputs list for details. Notice that some inputs (as listed below) in **cumulus cluster** inputs list are DISABLED for **cumulus_subcluster**:

- cite_seq
- cite_seq_capping
- output_filtration_results
- plot_filtration_results
- plot_filtration_figsize
- output_seurat_compatible

- batch_group_by
- min_genes
- max_genes
- min_umis
- max_umis
- mito_prefix
- percent_mito
- gene_percent_cells
- min_genes_on_raw
- counts_per_cell_after

cumulus_subcluster's outputs

Name	Type	Description
output_h5ad	File	h5ad-formatted HDF5 file containing all results (output_name.h5ad). If `perform_de_analysis` is on, this file should be the same as *output_de_h5ad*. To load this file in Python, it's similar as in cumulus cluster outputs section. Besides, for subcluster results, there is a new cell attributes in `data.obs['pseudo_time']`, which records the inferred pseudotime for each cell.
output_log	File	This is a copy of the logging module output, containing important intermediate messages
output_loom_file	File	Generated loom file (output_name.loom)
output_de_h5ad	File	Generated h5ad-formatted results with DE results updated (output_name.h5ad)
output_de_xlsx	File	Generated Spreadsheet reporting DE results (output_name.de.xlsx)
output_pdfs	Array[File]	Generated pdf files
output_htmls	Array[File]	Generated html files

14.9.4 Load Cumulus results into Pegasus

Pegasus is a Python package for large-scale single-cell/single-nucleus data analysis, and it uses PegasusIO for read/write. To load Cumulus results into Pegasus, we provide instructions based on file format:

- **zarr**: Annotated Zarr file in zip format. This is the standard output format of Cumulus. You can load it by:

```
import pegasusio as io
data = io.read_input("output_name.zarr.zip")
```

- **h5ad**: When setting **"output_h5ad"** field in *Cumulus cluster* to *true*, a list of annotated H5AD file(s) will be generated besides Zarr result. If the input data have multiple foci, Cumulus will generate one H5AD file per focus. You can load it by:

```
import pegasusio as io
adata = io.read_input("output_name.focus_key.h5ad")
```

Sometimes you may also want to specify how the result is loaded into memory. In this case, `read_input` has argument `mode`. Please see its documentation for details.

- **loom**: When setting **"output_loom"** field in *Cumulus cluster* to **true**, a list of loom format file(s) will be generated besides Zarr result. Similarly as H5AD output, Cumulus generates multiple loom files if the input data have more than one foci. To load loom file, you can optionally set its genome name in the following way as this information is not contained by loom file:

```
import pegasusio as io
data = pg.read_input("output_name.focus_key.loom", genome = "GRCh38")
```

After loading, Pegasus manipulate the data matrix in PegasusIO *MultimodalData* structure.

14.9.5 Load Cumulus results into Seurat

Seurat is a single-cell data analysis package written in R.

Load H5AD File into Seurat

First, you need to set **"output_h5ad"** field to **true** in cumulus cluster inputs to generate Seurat-compatible output files `output_name.focus_key.h5ad`, in addition to the standard result `output_name.zarr.zip`. If the input data have multiple foci, Cumulus will generate one H5AD file per focus.

Notice that Python, and Python package anndata with version at least `0.6.22.post1`, and R package reticulate are required to load the result into Seurat.

Execute the R code below to load the h5ad result into Seurat (working with both Seurat v2 and v3):

```
source("https://raw.githubusercontent.com/klarman-cell-observatory/cumulus/master/
↪workflows/cumulus/h5ad2seurat.R")
ad <- import("anndata", convert = FALSE)
test_ad <- ad$read_h5ad("output_name.focus_key.h5ad")
result <- convert_h5ad_to_seurat(test_ad)
```

The resulting Seurat object `result` has three data slots:

- **raw.data** records filtered raw count matrix.
- **data** records filtered and log-normalized expression matrix.
- **scale.data** records variable-gene-selected, standardized expression matrix that are ready to perform PCA.

Load loom File into Seurat

First, you need to set **"output_loom"** field to **true** in cumulus cluster inputs to generate a loom format output file, say `output_name.focus_key.loom`, in addition to the standard result `output_name.zarr.zip`. If the input data have multiple foci, Cumulus will generate one loom file per focus.

You also need to install *loomR* package in your R environment:

```
install.package("devtools")
devtools::install_github("mojaveazure/loomR", ref = "develop")
```

Execute the R code below to load the loom file result into Seurat (working with Seurat v3 only):

```
source("https://raw.githubusercontent.com/klarman-cell-observatory/cumulus/master/
↪workflows/cumulus/loom2seurat.R")
result <- convert_loom_to_seurat("output_name.focus_key.loom")
```

In addition, if you want to set an active cluster label field for the resulting Seurat object, do the following:

```
Idents(result) <- result@meta.data$louvain_labels
```

where `louvain_labels` is the key to the Louvain clustering result in Cumulus, which is stored in cell attributes `result@meta.data`.

14.9.6 Load Cumulus results into SCANPY

SCANPY is another Python package for single-cell data analysis. We provide instructions on loading Cumulus output into SCANPY based on file format:

- **h5ad**: Annotated H5AD file. This is the standard output format of Cumulus:

```
import scanpy as sc
adata = sc.read_h5ad("output_name.h5ad")
```

Sometimes you may also want to specify how the result is loaded into memory. In this case, `read_h5ad` has argument `backed`. Please see SCANPY documentation for details.

- **loom**: This format is generated when setting **"output_loom"** field in Cumulus cluster to **true**:

```
import scanpy as sc
adata = sc.read_loom("output_name.loom")
```

Besides, `read_loom` has a boolean `sparse` argument to decide whether to read the data matrix as sparse, with default value `True`. If you want to load it as a dense matrix, simply type:

```
adata = sc.read_loom("output_name.loom", sparse = False)
```

After loading, SCANPY manipulates the data matrix in anndata structure.

14.9.7 Visualize Cumulus results in Python

Ensure you have Pegasus installed.

Download your analysis result data, say `output_name.zarr.zip`, from Google bucket to your local machine.

Load the output:

```
import pegasusio as io
data = io.read_input("output_name.zarr.zip")
```

Violin plot of the computed quality measures:

```
fig = pg.violin(data, keys = ['n_genes', 'n_counts', 'percent_mito'], by = 'passed_qc
↪')
fig.savefig('output_file.qc.pdf', dpi = 500)
```

t-SNE plot colored by louvain cluster labels and channel:

```
fig = pg.embedding(data, basis = 'tsne', keys = ['louvain_labels', 'Channel'])
fig.savefig('output_file.tsne.pdf', dpi = 500)
```

t-SNE plot colored by genes of interes (also known as Feature Plot):

```
fig = pg.embedding(data, basis = 'tsne', keys = ['CD4', 'CD8A'])
fig.savefig('output_file.genes.tsne.pdf', dpi = 500)
```

For other embedding plots using FIt-SNE (`fitsne`), Net t-SNE (`net_tsne`), CITE-Seq FIt-SNE (`citeseq_fitsne`), UMAP (`umap`), Net UMAP (`net_umap`), FLE (`fle`), or Net FLE (`net_fle`) coordinates, simply substitute its basis name for `tsne` in the code above.

Composition plot on louvain cluster labels colored by channel:

```
fig = pg.composition_plot(data, by = 'louvain_labels', condition = 'Channel')
fig.savefig('output_file.composition.pdf', dpi = 500)
```

14.10 Topic modeling

14.10.1 Prepare input data

Follow the steps below to run **topic_modeling** on Terra.

1. Prepare your count matrix. **Cumulus** currently supports the following formats: 'zarr', 'h5ad', 'loom', '10x', 'mtx', 'csv', 'tsv' and 'fcs' (for flow/mass cytometry data) formats

2. Upload your count matrix to the workspace.

   Example:

   ```
   gsutil cp /foo/bar/projects/dataset.h5ad gs://fc-e0000000-0000-0000-0000-
   ↪000000000000/
   ```

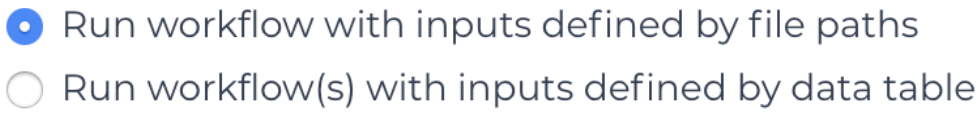
   where `/foo/bar/projects/dataset.h5ad` is the path to your dataset on your local machine, and `gs://fc-e0000000-0000-0000-0000-000000000000/` is the Google bucket destination.

3. Import *topic_modeling* workflow to your workspace.

   See the Terra documentation for adding a workflow. The *cumulus* workflow is under `Broad Methods Repository` with name "**cumulus/topic_modeling**".

   Moreover, in the workflow page, click the `Export to Workspace...` button, and select the workspace to which you want to export *topic_modeling* workflow in the drop-down menu.

4. In your workspace, open `topic_modeling` in `WORKFLOWS` tab. Select `Run workflow with inputs defined by file paths` as below



and click the `SAVE` button.

14.10.2 Workflow input

Inputs for the *topic_modeling* workflow are described below. Required inputs are in bold.

Name	Description	Example	Default
input_file	Google bucket URL of the input count matrix.	"gs://fc-e0000000-0000-0000-0000-000000000000/my_dataset.h5ad"	
number_of_topics	Array of number of topics.	[10,15,20]	
prefix_exclude	Comma separated list of features to exclude that start with prefix.	"mt-,Rpl,Rps"	"mt-,Rpl,Rps"
min_percent_expressed	Exclude features expressed below min_percent.	2	
max_percent_expressed	Exclude features expressed below min_percent.	98	
random_number_seed	Random number seed for reproducibility.	0	0

14.10.3 Workflow output

Name	Type	Description
coherence_plot	File	Plot of coherence scores vs. number of topics
perplexity_plot	File	Plot of perplexity values vs. number of topics
cell_scores	Array[File]	Topic by cells (one file for each topic number)
feature_topics	Array[File]	Topic by features (one file for each topic number)
report	Array[File]	HTML visualization report (one file for each topic number)
stats	Array[File]	Computed coherence and perplexity (one file for each topic number)
model	Array[File]	Serialized LDA model (one file for each topic number)
corpus	File	Serialized corpus
dictionary	File	Serialized dictionary

14.11 Run Terra pipelines via command line

You can run Terra pipelines via the command line by installing the **altocumulus** package.

14.11.1 Install `altocumulus` for Broad users

Request an UGER node:

```
reuse UGER
qrsh -q interactive -l h_vmem=4g -pe smp 8 -binding linear:8 -P regevlab
```

The above command requests an interactive shell using the regevlab project with 4G memory per thread, 8 threads. Feel free to change the memory, thread, and project parameters.

Add conda to your path:

```
reuse Anaconda3
```

Activate the alto virtual environment:

```
source activate /seq/regev_genome_portal/conda_env/cumulus
```

14.11.2 Install `altocumulus` for non-Broad users

1. Make sure you have `conda` installed. If you haven't installed conda, use the following commands to install it on Linux:

   ```
   wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh .
   bash Miniconda3-latest-Linux-x86_64.sh -p /home/foo/miniconda3
   mv Miniconda3-latest-Linux-x86_64.sh /home/foo/miniconda3
   ```

   where `/home/foo/miniconda3` should be replaced by your own folder holding Miniconda3.

Or use the following commdands for MacOS installation:

```
curl -O curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
bash Miniconda3-latest-MacOSX-x86_64.sh -p /Users/foo/miniconda3
mv Miniconda3-latest-MacOSX-x86_64.sh /Users/foo/miniconda3

where ``/Users/foo/miniconda3`` should be replaced by your own folder holding␣
↪Miniconda3.
```

1. Create a conda environment named "alto" and install `altocumulus`:

   ```
   conda create -n alto -y pip
   source activate alto
   pip install altocumulus
   ```

When the installation is done, type `alto -h` in terminal to see if you can see the help information.

14.11.3 Set up Google Cloud Account

Install Google Cloud SDK on your local machine.

Then type the following command in your terminal

```
gcloud auth application-default login
```

and follow the pop-up instructions to set up your Google cloud account.

14.11.4 Run Terra workflows via `alto run`

alto run runs a Terra method. Features:

- Uploads local files/directories in your inputs to a Google Cloud bucket updates the file paths to point to the Google Cloud bucket.

Your sample sheet can point to local file paths. In this case, `alto run` will take care of uploading directories smartly (e.g. only upload necessary files in BCL folders) and modifying the sample sheet to point to a Google Cloud bucket.

- Creates or uses an existing workspace.
- Uses the latest version of a method unless the method version is specified.

Options

Required options are in bold.

Name	Description
-m <METHOD> **–method <METHOD>**	Specify a Terra workflow <*METHOD*> to use. <*METHOD*> is of format *Namespace/Name* (e.g. `cumulus/cellranger_workflow`). A snapshot version number can optionally be specified (e.g. `cumulus/cellranger_workflow/4`); otherwise the latest snapshot of the method is used.
-w <WORKSPACE> **–workspace <WORKSPACE>**	Specify which Terra workspace <*WORKSPACE*> to use. <*WORKSPACE*> is also of format *Namespace/Name* (e.g. `foo/bar`). The workspace will be created if it does not exist.
-i <WDL_INPUTS> **–inputs <WDL_INPUTS>**	Specify the WDL input JSON file to use. It can be a local file, a JSON string, or a Google bucket URL directing to a remote JSON file.
–bucket-folder <folder>	Store inputs to <folder> under workspace's google bucket.
-o <updated_json> –upload <updated_json>	Upload files/directories to Google bucket of the workspace, and generate an updated input JSON file (with local paths replaced by Google bucket URLs) to <updated_json> on local machine.
–no-cache	Disable Terra cache calling

Example

This example shows how to use `alto run` to run cellranger_workflow to extract gene-count matrices from sequencing output.

1. Prepare your sample sheet `example_sample_sheet.csv` as the following:

```
Sample,Reference,Flowcell,Lane,Index,Chemistry
sample_1,GRCh38,/my-local-path/flowcell1,1-2,SI-GA-A8,threeprime
sample_2,GRCh38,/my-local-path/flowcell1,3-4,SI-GA-B8,threeprime
sample_3,mm10,/my-local-path/flowcell1,5-6,SI-GA-C8,fiveprime
sample_4,mm10,/my-local-path/flowcell1,7-8,SI-GA-D8,fiveprime
sample_1,GRCh38,/my-local-path/flowcell2,1-2,SI-GA-A8,threeprime
sample_2,GRCh38,/my-local-path/flowcell2,3-4,SI-GA-B8,threeprime
sample_3,mm10,/my-local-path/flowcell2,5-6,SI-GA-C8,fiveprime
sample_4,mm10,/my-local-path/flowcell2,7-8,SI-GA-D8,fiveprime
```

   where `/my-local-path` is the top-level directory of your BCL files on your local machine.

   Note that `sample_1`, `sample_2`, `sample_3`, and `sample_4` are sequenced on 2 flowcells.

2. Prepare your JSON input file `inputs.json` for cellranger_workflow:

```
{
    "cellranger_workflow.input_csv_file" : "/my-local-path/sample_sheet.csv",
    "cellranger_workflow.output_directory" : "gs://url/outputs",
    "cellranger_workflow.delete_input_bcl_directory": true
}
```

   where `gs://url/outputs` is the folder on Google bucket of your workspace to hold output.

3. Run the following command to kick off your Terra workflow:

```
alto run -m cumulus/cellranger_workflow -i inputs.json -w myworkspace_namespace/
↪myworkspace_name -o inputs_updated.json
```

   where `myworkspace_namespace/myworkspace_name` should be replaced by your workspace namespace and name.

Upon success, `alto run` returns a URL pointing to the submitted Terra job for you to monitor.

If for any reason, your job failed. You could rerun it without uploading files again via the following command:

```
alto run -m cumulus/cellranger_workflow -i inputs_updated.json -w myworkspace_
↪namespace/myworkspace_name
```

because `inputs_updated.json` is the updated version of `inputs.json` with all local paths being replaced by their corresponding Google bucket URLs after uploading.

14.12 Examples

14.12.1 Example of Cell-Hashing and CITE-Seq Analysis on Cloud

In this example, you'll learn how to perform Cell-Hashing and CITE-Seq analysis using **cumulus** on Terra.

---

0. Workspace and Data Preparation

After registering on Terra and creating a workspace there, you'll need the following two information:

- **Terra workspace name**. This is shown on your Terra workspace webpage, with format *"<workspace-namespace>/<workspace-name>"*. Let it be `ws-lab/ws-01` in this example, which means that your workspace has namespace **ws-lab** and name **ws-01**.
- The corresponding **Google Cloud Bucket location** of your workspace. You can check it by clicking the link under **"Google Bucket"** title on your Terra workspace webpage. Let it be `gs://fc-e0000000-0000-0000-0000-000000000000` in this example.

Then upload your BCL directories to Google bucket of your workspace using gsutil:

```
gsutil -m cp -r /my-local-path/BCL/* gs://fc-e0000000-0000-0000-0000-000000000000/
↪data-source
```

where option `-m` means copy in parallel, `-r` means copy the directory recursively, `/my-local-path/BCL` is the path to the top-level directory of your BCL files on your local machine, and `data-source` is the folder on Google bucket to hold the uploaded data.

---

1. Extract Gene-Count Matrices

First step is to extract gene-count matrices from sequencing output.

You need two original files from your dataset to start:

- Cell-Hashing Index CSV file, say its filename is `cell_hashing_index.csv`, of format *"feature_barcode,feature_name"*. See an example below:

  ```
  AATCATCACAAGAAA,CB1
  GGTCACTGTTACGTA,CB2
  ... ...
  ```

  where each line is a pair of feature barcode and feature name of a sample.

- CITE-Seq Index CSV file, say its filename is `cite_seq_index.csv`, of the same format as above. See an example below:

  ```
  TTACATGCATTACGA,CD19
  GCATTAGCATGCAGC,HLA-ABC
  ... ...
  ```

  where each line is a pair of Barcode and Specificity of an Antibody.

Then upload them to your Google Bucket using gsutil. Assuming both files are in folder `/Users/foo/data-source` on your local machine, type the following command to upload:

```
gsutil -m cp -r /Users/foo/data-source gs://fc-e0000000-0000-0000-0000-000000000000/
↪data-source
```

where `gs://fc-e0000000-0000-0000-0000-000000000000/data-source` is your working directory at cloud side, which can be changed at your will.

Next, create a sample sheet, `cellranger_sample_sheet.csv`, for Cell Ranger processing. Below is an example:

```
Sample,Reference,Flowcell,Lane,Index,DataType,FeatureBarcodeFile
sample_control,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/data-source,2,SI-
↪GA-F1,rna
sample_cc,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/data-source,3,SI-GA-A1,
↪rna
sample_cell_hashing,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/data-source,3,
↪ATTACTCG,adt,cell_hashing_index.csv
sample_cite_seq,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/data-source,3,
↪CGTGAT,adt,cite_seq_index.csv
```

For the details on how to prepare this sample sheet, please refer to Step 3 of Cell Ranger sample sheet instruction.

When you are done with the sample sheet, upload it to Google bucket:

```
gsutil cp cellranger_sample_sheet.csv gs://fc-e0000000-0000-0000-0000-000000000000/my-
↪dir/
```

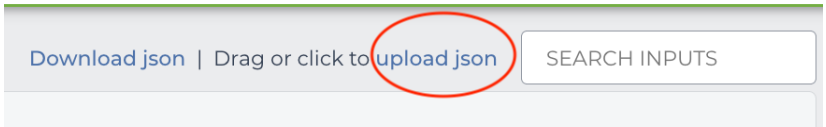
Now we are ready to set up **cellranger_workflow** workflow for this phase. If your workspace doesn't have this workflow, import it to your workspace by following cellranger_workflow import instructions.

Then prepare a JSON file, `cellranger_inputs.json`, which is used to set up the workflow inputs:

```
{
        "cellranger_workflow.input_csv_file" : "gs://fc-e0000000-0000-0000-0000-
↪000000000000/my-dir/cellranger_sample_sheet.csv",
        "cellranger_workflow.output_directory" : "gs://fc-e0000000-0000-0000-0000-
↪000000000000/my-dir"
}
```

where `gs://fc-e0000000-0000-0000-0000-000000000000/my-dir` is the remote directory in which the output of cellranger_workflow will be generated. For the details on the options above, please refer to Cell Ranger workflow inputs.

When you are done with the JSON file, on cellranger_workflow workflow page, upload `cellranger_inputs.json` by clicking `upload json` link as below:



Then Click `SAVE` button to save the inputs, and click `RUN ANALYSIS` button as below to start the job:

When the execution is done, all the output results will be in folder `gs://fc-e0000000-0000-0000-0000-000000000000/my-dir`.

You'll need 4 files for the next phases. 3 are from the output:

- RNA count matrix of the sample group of interest: `gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/sample_cc/raw_feature_bc_matrix.h5`;
- Cell-Hashing Antibody count matrix: `gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/sample_cell_hashing/sample_cell_hashing.csv`;

- CITE-Seq Antibody count matrix: `gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/sample_cite_seq/sample_cite_seq.csv`.

Besides, create a sample sheet, `citeseq_antibody_control.csv`, with content as the following example:

```
Antibody,Control
CD3-0034,Mouse_IgG1
CD4-0045,Mouse_IgG1
... ...
```

where each line is a pair of Antibody name and the Control group name to which it is assigned. You should be able to get this information from your experiment setting or the original dataset.

Copy or upload them to `gs://fc-e0000000-0000-0000-0000-000000000000/my-dir`.

---

2. Demultiplex Cell-Hashing Data

1. Prepare a sample sheet, `demultiplex_sample_sheet.csv`, with the following content:

   ```
   OUTNAME,RNA,TagFile,TYPE
   exp,gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/raw_feature_bc_matrix.h5,
   ↪gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/sample_cell_hashing.csv,
   ↪cell-hashing
   ```

   where **OUTNAME** specifies the subfolder and file names of output, which is free to change, **RNA** and **TagFile** columns specify the RNA and hashing tag meta-data of samples, and **TYPE** is `cell-hashing` for this phase.

   Then upload it to Google bucket:

   ```
   gsutil cp demultiplex_sample_sheet.csv gs://fc-e0000000-0000-0000-0000-
   ↪000000000000/my-dir/
   ```

2. If your workspace doesn't have **demultiplexing** workflow, import it to your workspace by following Step 2 of demultiplexing workflow preparation instructions.

3. Prepare an input JSON file, `demultiplex_inputs.json` with the following content to set up cumulus_hashing_cite_seq workflow inputs:

   ```
   {
           "demultiplexing.input_sample_sheet" : "gs://fc-e0000000-0000-0000-0000-
   ↪000000000000/my-dir/demultiplex_sample_sheet.csv",
           "demultiplexing.output_directory" : "gs://fc-e0000000-0000-0000-0000-
   ↪000000000000/my-dir/"
   }
   ```

   For the details on these options, please refer to demultiplexing workflow inputs.

4. On the page of cumulus_hashing_cite_seq workflow, upload `demultiplex_inputs.json` by clicking `upload json` link. Save the inputs, and click `RUN ANALYSIS` button to start the job.

When the execution is done, you'll get a processed file, `exp_demux.zarr`, stored on cloud `gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/exp/`.

---

3. Merge RNA and ADT Matrices for CITE-Seq Data

1. Prepare a sample sheet, `cite_seq_sample_sheet.csv`, with the following content:

```
OUTNAME,RNA,ADT
exp_raw,gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/exp/exp_demux.zarr,
↪gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/sample_cite_seq.csv
```

   The structure of sample sheet here is the same as Phase 2. The difference is that you are now using the demultiplexed output `h5sc` file from Phase 2 as **RNA** here.

   Then upload it to Google bucket:

```
gsutil cp cite_seq_sample_sheet.csv gs://fc-e0000000-0000-0000-0000-000000000000/
↪my-dir/
```

2. Prepare an input JSON file, `cite_seq_inputs.json`, in the same directory as above, with the following content:

```
{
        "cumulus_cite_seq.input_sample_sheet" : "gs://fc-e0000000-0000-0000-0000-
↪000000000000/my-dir/cite_seq_sample_sheet.csv",
        "cumulus_cite_seq.output_directory" : "gs://fc-e0000000-0000-0000-0000-
↪000000000000/my-dir/",
        "cumulus_cite_seq.antibody_control_csv" : "gs://fc-e0000000-0000-0000-
↪0000-000000000000/my-dir/citeseq_antibody_control.csv"
}
```

   For the details on these options, please refer to cumulus_cite_seq workflow inputs.

3. On **cumulus_cite_seq** workflow page, clear all previous inputs, and then upload `cite_seq_inputs.json` by clicking `upload json` link. Save the new inputs, and click `RUN ANALYSIS` button to start the job.

When the execution is done, you'll get a merged raw matrices file, `exp_raw.zarr`, stored on cloud `gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/exp_raw`.

---

4. Data Analysis

1. Prepare a sample sheet, `cumulus_count_matrix.csv`, with the following content:

```
Sample,Location
exp,gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/exp_raw/exp_raw.zarr
```

   This sample sheet describes the metadata for each 10x channel (as one row in the sheet). **Sample** specifies the name for each channel, which can be renamed; **Location** specifies the file location, which is the output of Phase 3.

   Then upload it to Google bucket:

```
gsutil cp cumulus_count_matrix.csv gs://fc-e0000000-0000-0000-0000-000000000000/
↪my-dir/
```

   Alternative, if you have only one count matrix for analysis, which is the case here, you can skip this step. See this manual for input file formats that cumulus currently supports.

2. If your workspace doesn't have **cumulus** workflow, import it to your workspace by following Step 2 and 3 of cumulus documentation.

3. Prepare a JSON file, `cumulus_inputs.json` with the following content to set up **cumulus** workflow inputs:

```
{
        "cumulus.input_file" : "gs://fc-e0000000-0000-0000-0000-000000000000/my-
↪dir/cumulus_count_matrix.csv",
        "cumulus.output_directory" : "gs://fc-e0000000-0000-0000-0000-
↪000000000000/my-dir/results",
        "cumulus.output_name" : "exp_merged_out",
        "cumulus.num_cpu" : 8,
        "cumulus.select_only_singlets" : true,
        "cumulus.cite_seq" : true,
        "cumulus.run_louvain" : true,
        "cumulus.find_markers_lightgbm" : true,
        "cumulus.remove_ribo" : true,
        "cumulus.mwu" : true,
        "cumulus.annotate_cluster" : true,
        "cumulus.plot_fitsne" : "louvain_labels,assignment",
        "cumulus.plot_citeseq_fitsne" : "louvain_labels,assignment",
        "cumulus.plot_composition" : "louvain_labels:assignment"
}
```

   Alternatively, if you have only one count matrix for analysis and has skipped Step 1, directly set its location in `cumulus.input_file` parameter above. For this example, it is:

```
{
        "cumulus.input_file" : "gs://fc-e0000000-0000-0000-0000-000000000000/my-
↪dir/exp_raw/exp_raw.zarr",
        ... ...
}
```

   All the rest parameters remain the same.

   Notice that for some file formats, `cumulus.genome` is required.

   A typical cumulus pipeline consists of 4 steps, which is given here. For the details of options above, please refer to cumulus inputs.

4. On the page of cumulus workflow, upload `cumulus_inputs.json` by clicking `upload json` link. Save the inputs, and click `RUN ANALYSIS` button to start the job.

When the execution is done, you'll get the following results stored on cloud `gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/results/exp_merged_out/` to check:

- `exp_merged_out.zarr`: The aggregated count matrix data. This file doesn't exist if your `cumulus.input_file` parameter is not a sample sheet.
- `exp_merged_out.h5ad`: The processed RNA matrix data.
- `exp_merged_out.filt.xlsx`: The Quality-Control (QC) summary of the raw data.
- `exp_merged_out.filt.{UMI, gene, mito}.pdf`: The QC plots of the raw data.
- `exp_merged_out.de.xlsx`: Differential Expression analysis result.
- `exp_merged_out.markers.xlsx`: Result on cluster-specific markers predicted by gradient boosting machine.
- `exp_merged_out.anno.txt`: Cell type annotation output.
- `exp_merged_out.fitsne.pdf`: FIt-SNE plot.
- `exp_merged_out.citeseq.fitsne.pdf`: CITE-Seq FIt-SNE plot.

- `exp_merged_out.louvain_labels.assignment.composition.pdf`: Composition plot.

You can directly go to your Google Bucket to view or download these results.

(optional) Run Terra Workflows in Command Line

For Phase 1, 2, and 3, besides uploading sample sheets and setting-up workflow inputs on workflow pages, you can also start the workflow execution via command line using **altocumulus** tool.

First, install *altocumulus* by following altocumulus installation instruction.

1. For Phase 1 above, when you are done with creating a sample sheet `cellranger_sample_sheet.csv` on your local machine, in the same directory, prepare JSON file `cellranger_inputs.json` as below:

```
{
        "cellranger_workflow.input_csv_file" : "cellranger_sample_sheet.csv",
        ... ...
}
```

   where all the rest parameters remain the same as in Phase 1. Import **cellranger_workflow** workflow to your workspace as usual.

   Now run the following command in the same directory on your local machine:

```
alto run -m cumulus/cellranger_workflow -w ws-lab/ws-01 --bucket-folder my-dir -i␣
↪cellranger_input.json -o cellranger_input_updated.json
```

   Notice that if the execution failed, you could rerun the execution by setting `cellranger_input_updated.json` for `-i` option to use the sample sheet already uploaded to Google bucket. Similarly below.

2. For Phase 2 above, similarly, in the same directory of your `demultiplex_sample_sheet.csv` file, prepare JSON file `demultiplex_inputs.json` as below:

```
{
        "demultiplexing.input_sample_sheet" : "demultiplex_sample_sheet.csv",
        ... ...
}
```

   where all the rest parameters remain the same as in Phase 2. Import **demultiplexing** workflow to your workspace as usual.

   Run the following command in the same directory on your local machine:

```
alto run -m cumulus/demultiplexing -w ws-lab/ws-01 --bucket-folder my-dir -i␣
↪demultiplex_inputs.json -o demultiplex_inputs_updated.json
```

3. For Phase 3 above, similarly, in the same directory of your `cite_seq_sample_sheet.csv` file, prepare JSON file `cite_seq_inputs.json` as below:

```
{
        "cumulus_cite_seq.input_sample_sheet" : "cite_seq_sample_sheet.csv",
        ... ...
}
```

   where all the rest parameters remain the same as in Phase 3. Import **cumulus_cite_seq** workflow to your workspace as usual.

   Run the following command in the same directory on your local machine:

```
alto run -m cumulus/cumulus_cite_seq -w ws-lab/ws-01 --bucket-folder my-dir -i␣
↪cite_seq_inputs.json -o cite_seq_inputs_updated.json
```

4. For Phase 4 above, similarly, in the same directory of your `cumulus_count_matrix.csv` file, prepare JSON file `cumulus_inputs.json` as below:

```
{
        "cumulus.input_file" : "cumulus_count_matrix.csv",
        ... ...
}
```

where all the rest parameters remain the same as in Phase 4.

Alternatively, if your input is not a sample sheet, simply set your `cumulus_inputs.json` as:

```
{
        "cumulus.input_file" : "gs://fc-e0000000-0000-0000-0000-000000000000/my-
↪dir/exp_raw/exp_raw.zarr",
        ... ...
}
```

where all the rest parameters remain the same. Import **cumulus** workflow to your workspace as usual.

Run the following command in the same directory of your `cumulus_inputs.json` file:

```
alto run -m cumulus/cumulus -w ws-lab/ws-01 --bucket-folder my-dir/results -i␣
↪cumulus_inputs.json -o cumulus_inputs_updated.json
```

Examples using Terra to perform single-cell sequencing analysis are provided here. Please click the topics on the left panel under title **"Examples"** to explore.

14.13 Contributions

We welcome contributions to our repositories that make up the Cumulus ecosystem:

- pegasus
- pegasusio
- demuxEM
- cumulus
- cumulus_feature_barcoding
- scPlot
- altocumulus
- cirrocumulus

In addition to the Cumulus team, we would like to sincerely thank the following contributors:

Name	Note
Kirk Gosik	Assistance with topic modeling workflow

14.14 Contact us

If you have any questions related to Cumulus, please feel free to contact us via Cumulus Support Google Group.