# Cumulus Documentation

Bo Li, Joshua Gould, and et al.

Dec 21, 2019

# Contents

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 can enter a custom registry URL of their choice.

# CHAPTER 1

## Version 0.10.0 *October 2, 2019*

scCloud renamed to cumulus.

cumulus can accept either a sample sheet or a single file.

CHAPTER 2

# Version 0.7.0 *Feburary 14, 2019*

Added support for 10x genomics scATAC assays.

cumulus runs FIt-SNE as default.

# CHAPTER 3

## 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 4

## Version 0.5.0 *November 18, 2018*

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

# CHAPTER 5

## Version 0.4.0 *October 26, 2018*

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

# CHAPTER 6

## Version 0.3.0 *October 24, 2018*

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

# CHAPTER 7

## Version 0.2.0 *October 19, 2018*

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

# CHAPTER 8

## Version 0.1.0 *July 27, 2018*

KCO tools released!

## 8.1 First Time Running

### 8.1.1 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.

---

### 8.1.2 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.

## 8.2 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.

### 8.2.1 Latest version

| 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 |

### 8.2.2 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 |

### 8.2.3 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 |

### 8.2.4 Stable version - HTAPP v1

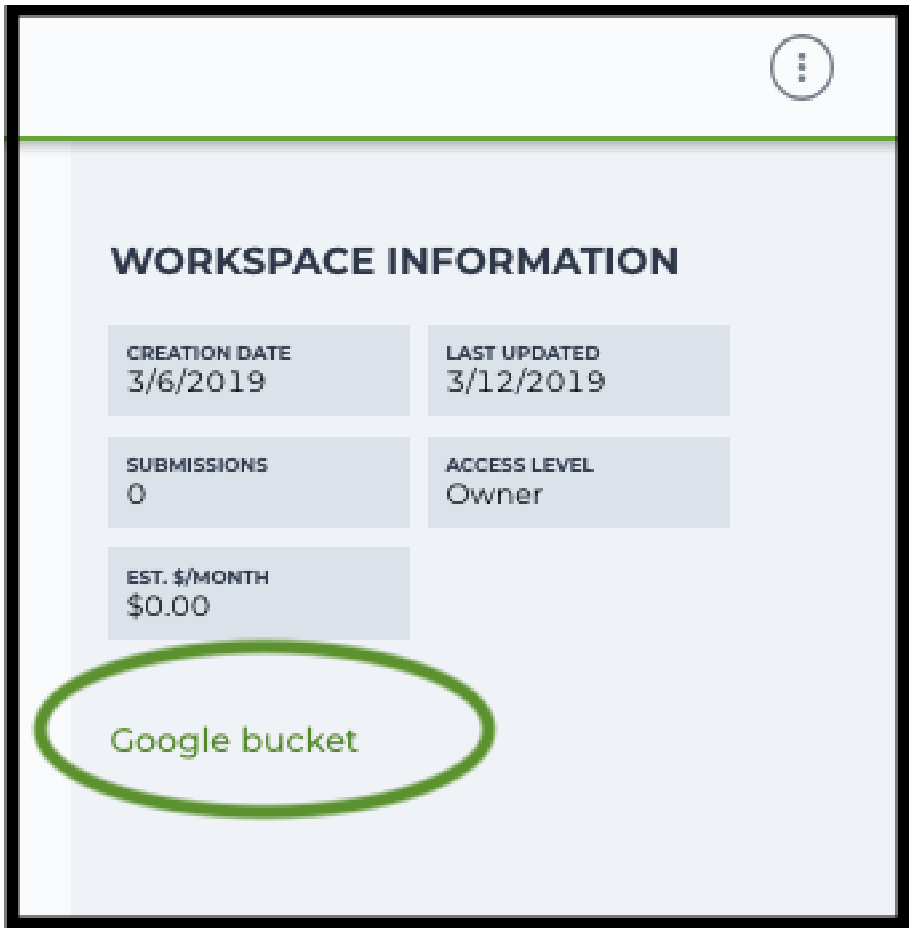
| WDL | Snapshot | Function |
|---|---|---|
| regev/cellranger_mkfastq_count | 39 | Run Cell Ranger to extract FASTQ files and generate gene-count matrices for 10x genomics data |
| scCloud/scCloud | 3 | Run scCloud analysis module for variable gene selection, batch correction, PCA, diffusion map, clustering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

## 8.3 Run Cell Ranger tools using cellranger_workflow

Follow the steps below to run CellRanger mkfastq/count/vdj on Terra.

1. 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.

   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, and `gs://fc-e0000000-0000-0000-0000-000000000000` should be replaced by your own workspace Google bucket URL.

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

   See *bcl2fastq* instructions.

3. Create a scRNA-Seq formatted 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 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.

   scRNA-Seq formatted sample sheet description (**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 *cellranger count* 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** for human GRCh38, cellranger reference 3.0.0, Ensembl v93 gene annotation **hg19_v3.0.0** for human hg19, cellranger reference 3.0.0, Ensembl v87 gene annotation **mm10_v3.0.0** for mouse mm10, cellranger reference 3.0.0, Ensembl v93 gene annotation **GRCh38_and_mm10_v3.1.0** for human and mouse, built from GRCh38 and mm10 cellranger references (3.1.0), Ensembl v93 gene annotations for both human and mouse **GRCh38_premrna_v1.2.0** (GRCh38_premrna, obsoleted) for 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** (mm10_premrna, obsoleted) for 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** (GRCh38_premrna_and_mm10_premrna, obsoleted) for human and mouse, introns included, built from GRCh38_premrna and mm10_premrna **GRCh38_vdj_v3.1.0** for human 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** for mouse V(D)J sequences, cellranger reference 3.1.0, annotation built from Ensembl *Mus_musculus.GRCm38.94.gtf* **GRCh38_atac_v1.1.0** for scATAC-Seq, human GRCh38, cellranger-atac reference 1.1.0 **mm10_atac_v1.1.0** for scATAC-Seq, mouse mm10, cellranger-atac reference 1.1.0 **GRCh38_v1.2.0** (GRCh38, obsoleted) for human GRCh38, cellranger reference 1.2.0, Ensembl v84 gene annotation **hg19_v1.2.0** (hg19, obsoleted) for human hg19, cellranger reference 1.2.0, Ensembl v82 gene annotation **mm10_v1.2.0** (mm10, obsoleted) for mouse mm10, cellranger reference 1.2.0, Ensembl v84 gene annotation **GRCh38_and_mm10_v1.2.0** (GRCh38_and_mm10, obsoleted) for human and mouse, built from GRCh38 and mm10 cellranger references (1.2.0), Ensembl v84 gene annotations for both human and mouse **GRCh38_vdj_v2.0.0** (GRCh38_vdj, obsoleted) for human 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_v2.0.0** (GRCm38_vdj, obsoleted) for mouse V(D)J sequences, cellranger reference 2.0.0, annotation built from Ensembl *Mus_musculus.GRCm38.90.chr_patch_hapl_scaff.gtf*. |
| **Flowcell** | Indicates the Google bucket URL of uploaded BCL folders. |
| **Lane** | Tells which lanes the sample was pooled into. |
| **Index** | Contains 10x sample index set names (e.g. SI-GA-A12). For cell-hashing/nucleus-hashing/CITE-Seq, this field should be the index |

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,gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4,1-
↪2,SI-GA-A8,threeprime,rna
sample_2,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4,3-
↪4,SI-GA-B8,SC3Pv3,rna
sample_3,mm10,gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4,5-6,
↪SI-GA-C8,fiveprime,rna
sample_4,mm10,gs://fc-e0000000-0000-0000-0000-000000000000/VK18WBC6Z4,7-8,
↪SI-GA-D8,fiveprime,rna
sample_1,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2,1-
↪2,SI-GA-A8,threeprime,rna
sample_2,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2,3-
↪4,SI-GA-B8,SC3Pv3,rna
sample_3,mm10,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2,5-6,
↪SI-GA-C8,fiveprime,rna
sample_4,mm10,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9Z2,7-8,
↪SI-GA-D8,fiveprime,rna
sample_5,GRCh38_vdj,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK10WBC9ZZ,1,SI-GA-A1,fiveprime,vdj
sample_6,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9ZZ,2,
↪AGATCCTT,SC3Pv3,adt,gs://fc-e0000000-0000-0000-0000-000000000000/
↪antibody_index.csv
sample_7,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/VK10WBC9ZZ,3,
↪TCCGGAGA,threeprime,crispr,gs://fc-e0000000-0000-0000-0000-000000000000/
↪crispr_index.csv
sample_8,GRCh38_atac_v1.1.0,gs://fc-e0000000-0000-0000-0000-000000000000/
↪VK10WBC9YB,*,SI-NA-A1,auto,atac
```

4. Upload your sample sheet to the workspace bucket.

   Example:

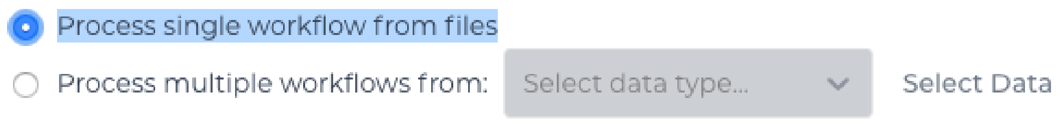
   ```
   gsutil cp /foo/bar/projects/sample_sheet.csv gs://fc-e0000000-0000-0000-
   ↪0000-000000000000/
   ```

5. Import *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.

6. In your workspace, open `cellranger_workflow` in `WORKFLOWS` tab. Select `Process single workflow from files` as below



[✓ Process single workflow from files]
Process multiple workflows from: Select data type... Select Data

and click `SAVE` button.

### 8.3.1 cellranger_workflow inputs:

`cellranger_workflow` takes Illumina outputs as input and runs `cellranger mkfastq`/`cellranger-atac mkfastq` and `cellranger count`/`cellranger vdj`/`cellranger-atac count`/cumulus feature extraction. 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 mkfastq` or `cellranger-atac mkfastq` | true | true |
| run_count | If you want to run steps after `mkfastq`, such as `cellranger count`, `cellranger vdj`, `cellranger-atac count` or `cumulus feature_extraction` | true | true |
| delete_input_bcl_directory | If delete BCL directories after demux. If false, you should delete this folder yourself so as to not incur storage charges | false | false |
| 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 |
| vdj_denovo | Do not align reads to reference V(D)J sequences before de novo assembly | false | false |
| vdj_chain | Force the web summary HTML and metrics summary CSV to only report on a particular chain type. The accepted values are: auto for autodetection based on TR vs IG representation, TR for T cell receptors, IG for B cell receptors, all for all chain types | TR | |
| scaffold_sequence | Scaffold sequence in sgRNA for Purturb-seq, only used for crispr data type | "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 2.2.0, 3.0.2, 3.1.0 | "3.0.2" | "3.0.2" |
| cellranger_atac_version | cellranger-atac version, currently only 1.1.0 | "1.1.0" | "1.1.0" |

Continued on next page

Table 1 – continued from previous page

| Name | Description | Example | Default |
|---|---|---|---|
| cumulus_version | Cumulus version for extracting feature barcode matrix, currently only 0.10.0 | "0.10.0" | "0.10.0" |
| docker_registry | Docker registry to use for cellranger_workflow. Options: <br>• "cumulusprod/" for Docker Hub images; <br>• "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-east1-d us-west1-a us-west1-b" | "us-east1-d us-west1-a us-west1-b" |
| num_cpu | Number of cpus to request for one node | 32 | 32 |
| atac_num_cpu | Number of cpus for cellranger-atac count | 64 | 64 |
| memory | Memory size string | "120G" | "120G" |
| feature_memory | Optional memory string for extracting feature count matrix | "32G" | "32G" |
| atac_memory | Memory string for cellranger-atac count | "57.6G" | "57.6G" |
| mkfastq_disk_space | Optional disk space in GB for mkfastq | 1500 | 1500 |
| count_disk_space | Disk space in GB needed for cellranger count | 500 | 500 |
| vdj_disk_space | Disk space in GB needed for cellranger vdj | 500 | 500 |
| feature_disk_space | Disk space in GB needed for extracting feature count matrix | 100 | 100 |
| atac_disk_space | Disk space in GB needed for cellranger-atac count | 500 | 500 |
| preemptible | Number of preemptible tries | 2 | 2 |

### 8.3.2 cellranger_workflow outputs:

See the table below for important *Cell Ranger mkfastq/count* 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. |
| output_vdj_directory | Array[String] | A list of google bucket urls containing vdj results, one url per sample. |
| output_adt_directory | Array[String] | A list of google bucket urls containing adt count matrices, one url per sample. |
| output_atac_count_directory | Array[String] | A list of google bucket urls containing cellranger-atac count 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. |

### 8.3.3 Only run the count part

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 FASTQS. Each folder should contain all FASTQ files for one sample.

   Example:

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

2. Create a scRNA-Seq formatted sample sheet.

   This sample sheet is the same as the sample sheet described for running mkfastq and count except that the full path to the FASTQ files is `FlowCell/Sample`.

   scRNA-Seq formatted sample sheet description (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 *cellranger count* for each 10x channel.<br>The elements in the *reference* column can be either Google bucket URLs to reference tarballs or keywords such as<br>**GRCh38_v3.0.0** for human GRCh38, cellranger reference 3.0.0, Ensembl v93 gene annotation<br>**hg19_v3.0.0** for human hg19, cellranger reference 3.0.0, Ensembl v87 gene annotation<br>**mm10_v3.0.0** for mouse mm10, cellranger reference 3.0.0, Ensembl v93 gene annotation<br>**GRCh38_and_mm10_v3.1.0** for human and mouse, built from GRCh38 and mm10 cellranger references (3.1.0), Ensembl v93 gene annotations for both human and mouse<br>**GRCh38_premrna_v1.2.0** (GRCh38_premrna, obsoleted) for human, introns included, built from GRCh38 cellranger reference 1.2.0, Ensembl v84 gene annotation, treating annotated transcripts as exons<br>**mm10_premrna_v1.2.0** (mm10_premrna, obsoleted) for mouse, introns included, built from mm10 cellranger reference 1.2.0, Ensembl v84 gene annotation, treating annotated transcripts as exons<br>**GRCh38_premrna_and_mm10_premrna_v1.2.0** (GRCh38_premrna_and_mm10_premrna, obsoleted) for human and mouse, introns included, built from GRCh38_premrna and mm10_premrna<br>**GRCh38_vdj_v3.1.0** for human V(D)J sequences, cellranger reference 3.1.0, annotation built from Ensembl *Homo_sapiens.GRCh38.94.chr_patch_hapl_scaff.gtf*<br>**GRCm38_vdj_v3.1.0** for mouse V(D)J sequences, cellranger reference 3.1.0, annotation built from Ensembl *Mus_musculus.GRCm38.94.gtf*<br>**GRCh38_atac_v1.1.0** for scATAC-Seq, human GRCh38, cellranger-atac reference 1.1.0<br>**mm10_atac_v1.1.0** for scATAC-Seq, mouse mm10, cellranger-atac reference 1.1.0<br>**GRCh38_v1.2.0** (GRCh38, obsoleted) for human GRCh38, cellranger reference 1.2.0, Ensembl v84 gene annotation<br>**hg19_v1.2.0** (hg19, obsoleted) for human hg19, cellranger reference 1.2.0, Ensembl v82 gene annotation<br>**mm10_v1.2.0** (mm10, obsoleted) for mouse mm10, cellranger reference 1.2.0, Ensembl v84 gene annotation<br>**GRCh38_and_mm10_v1.2.0** (GRCh38_and_mm10, obsoleted) for human and mouse, built from GRCh38 and mm10 cellranger references (1.2.0), Ensembl v84 gene annotations for both human and mouse<br>**GRCh38_vdj_v2.0.0** (GRCh38_vdj, obsoleted) for human 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*<br>**GRCm38_vdj_v2.0.0** (GRCm38_vdj, obsoleted) for mouse V(D)J sequences, cellranger reference 2.0.0, annotation built from Ensembl *Mus_musculus.GRCm38.90.chr_patch_hapl_scaff.gtf*. |
| **Flowcell** | Indicates the Google bucket URL of uploaded FASTQ folders. |
| Chemistry | Describes the 10x chemistry used for the sample.<br>This column is optional. The default chemistry for **rna** data (see DataType column below) is **auto**, which will try to detect the chemistry automatically. |

In the following example sample_1 is sequenced on 2 flowcells. The FASTQ files for flowcell_1 are located at gs://fc-e0000000-0000-0000-0000-000000000000/flowcell_1/sample_1 while the FASTQ files for flowcell_2 are located at gs://fc-e0000000-0000-0000-0000-000000000000/flowcell_2_sample1:

```
Sample,Reference,Flowcell
sample_1,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/flowcell_1
sample_1,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/flowcell_2
```

3. Set optional input `run_mkfastq` of cellranger_workflow workflow to `false`.

---

### 8.3.4 Extract feature count matrices from CITE-Seq/Cell-hashing/Nucleus-hashing/Perturb-seq assays

`cellranger_workflow` can optionally extract feature count matrices from *CITE-Seq/Cell-hashing/Nucleus-hashing/Perturb-seq* assays. For *CITE-Seq/Cell-hashing/Nucleus-hashing*, the feature refers to antibody. Note that for *CITE-Seq/Cell-hashing*, only Biolegend TotalSeq-A is supported. For *Perturb-seq*, the feature refers to guide RNA. To extract feature count matrices, please follow the instructions below.

#### Instructions to configure `cellranger_workflow`

1. Prepare one feature barcode file per assay and upload the files to the Google bucket.

   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.

2. Add assay information into the sample sheet.

   See below for an example:

   ```
   Sample,Reference,Flowcell,Lane,Index,Chemistry,DataType,FeatureBarcodeFile
   sample_1_rna,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/
   ↪VK18WBC6Z4,1-2,SI-GA-A8,threeprime,count
   sample_1_adt,GRCh38,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,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,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. Note that for the tag data, the *Index* field is different. The index for tag and crispr data should be Illumina index primer sequence (e.g. `ATTACTCG`, also known as `D701`, in row two). In addition, the *DataType* field is changed to `adt`.
- 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). Note that it is users' responsibility to avoid index collision between 10x genomics' RNA indexes (e.g. SI-GA-A8) and Illumina index sequences for tag and crispr data (e.g. `ATTACTCG`).

3. Fill in the ADT-specific parameters:

| Name | Description | Example | Default |
|---|---|---|---|
| scaffold_sequence | Scaffold sequence in sgRNA for Purturb-seq, only used for crispr data type | "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 |
| feature_memory | Optional memory string for extracting ADT count matrix | "32G" | "32G" |
| feature_disk_space | Optional disk space in GB needed for extracting ADT count matrix | 100 | 100 |

### 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.

### Extracted feature count matrix output

For each antibody tag or crispr tag sample, a folder with the sample ID is generated under `cellranger_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 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`.

## 8.4 bcl2fastq

### 8.4.1 License

bcl2fastq license

### 8.4.2 Docker

Read this tutorial if you are new to Docker and don't know how to write your Dockerfile.

For a Debian based docker, add the lines below into its Dockerfile to install `bcl2fastq`:

```
RUN apt-get install --no-install-recommends -y alien unzip
ADD https://support.illumina.com/content/dam/illumina-support/documents/downloads/
↪software/bcl2fastq/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*
```

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

### 8.4.3 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.

### 8.4.4 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.

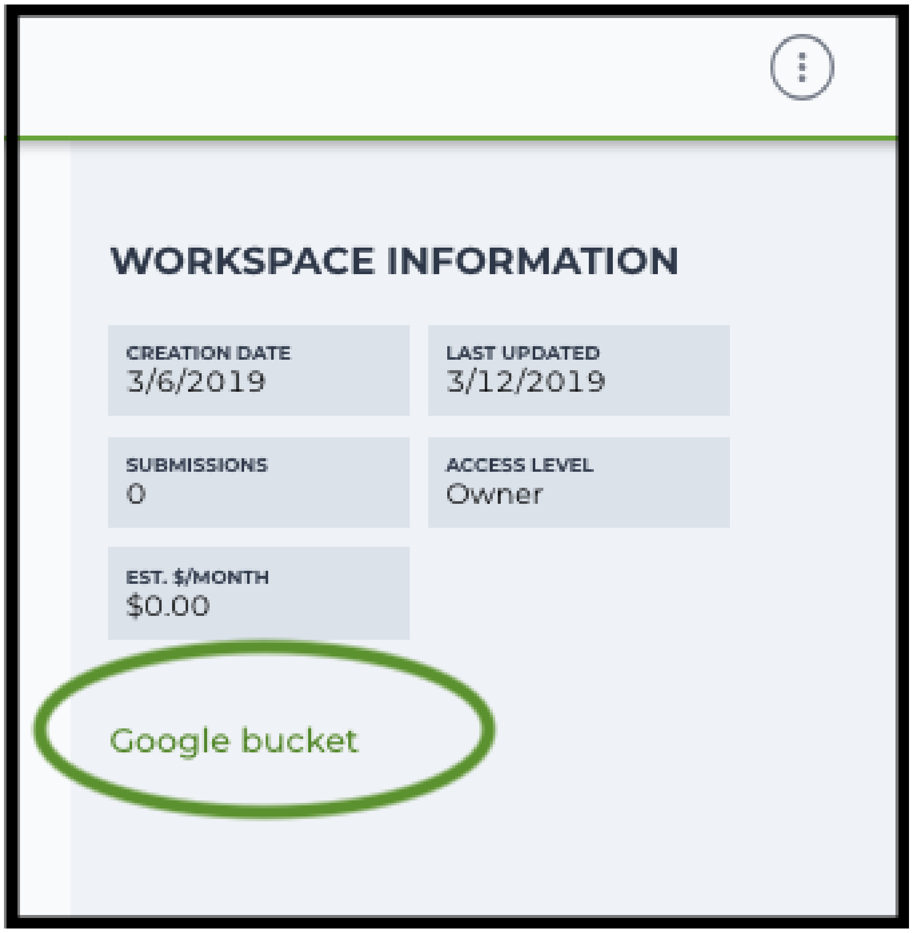
## 8.5 Extract gene-count matrices from plated-based SMART-Seq2 data

### 8.5.1 Run SMART-Seq2 Workflow

Follow the steps below to extract gene-count matrices from SMART-Seq2 data on Terra. This WDL aligns reads using *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.


WORKSPACE INFORMATION

CREATION DATE
3/6/2019

LAST UPDATED
3/12/2019

SUBMISSIONS
0

ACCESS LEVEL
Owner

EST. $/MONTH
$0.00

Google bucket

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 | Location of the FASTQ file for read1 in the cloud (gsurl). |

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

3. Upload your sample sheet to the workspace bucket.

   Example:

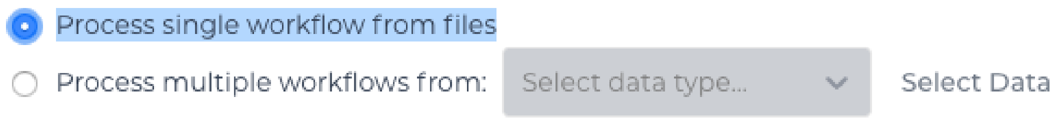
```
gsutil cp /foo/bar/projects/sample_sheet.csv gs://fc-e0000000-0000-0000-
↪0000-000000000000/
```

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 `Process single workflow from files` as below


Process single workflow from files
Process multiple workflows from:
Select data type...
Select Data

   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:<br>• Pre-created genome references: "GRCh38" for human; "GRCm38" and "mm10" for mouse.<br>• Create a custom genome reference using smartseq2_create_reference workflow, and specify its Google bucket URL here. | "GRCh38", or<br>"gs://fc-e0000000-0000-0000-0000-000000000000/rsem_ref.tar.gz" | |
| smartseq2_version | SMART-Seq2 version to use. Versions available: 1.0.0. | "1.0.0" | "1.0.0" |
| docker_registry | Docker registry to use. Options:<br>• "cumulusprod/" for Docker Hub images;<br>• "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-east1-d us-west1-a us-west1-b" |
| num_cpu | Number of cpus to request for one node | 4 | 4 |
| memory | Memory size string | "3.60G" | "3.60G" |
| disk_space | Disk space in GB | 10 | 10 |
| preemptible | Number of preemptible tries | 2 | 2 |

### Outputs:

See the table below for important 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`. |

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.

---

### 8.5.2 Custom Genome

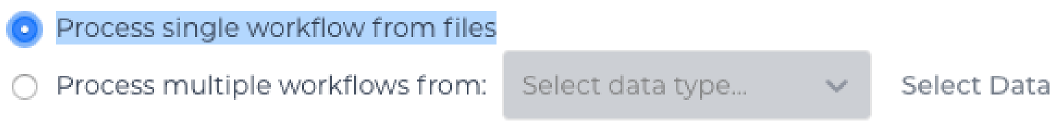
We also provide a way of generating user-customized Genome references for SMART-Seq2 workflow.

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 `Process single workflow from files` as below

   
   Process single workflow from files
   Process multiple workflows from:
   Select data type...
   Select Data

   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.<br>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.<br>For example, "gs://fc-e0000000-0000-0000-0000-000000000000/Homo_sapiens.GRCh38.83.gtf" | |
| smartseq2_version | SMART-Seq2 version to use.<br>Versions available: 1.0.0. | String | "1.0.0" |
| docker_registry | Docker registry to use. Options:<br>• "cumulusprod/" for Docker Hub images;<br>• "quay.io/cumulus/" for backup images on Red Hat registry. | String | "cumulusprod/" |
| zones | Google cloud zones | String | "us-east1-b us-east1-c us-east1-d" |
| cpu | Number of CPUs | Integer | 8 |
| memory | Memory size string | String | "7.2G" |
| extra_disk_space | Extra disk space in GB | Integer | 15 |
| preemptible | Number of preemptible tries | Integer | 2 |

### Outputs

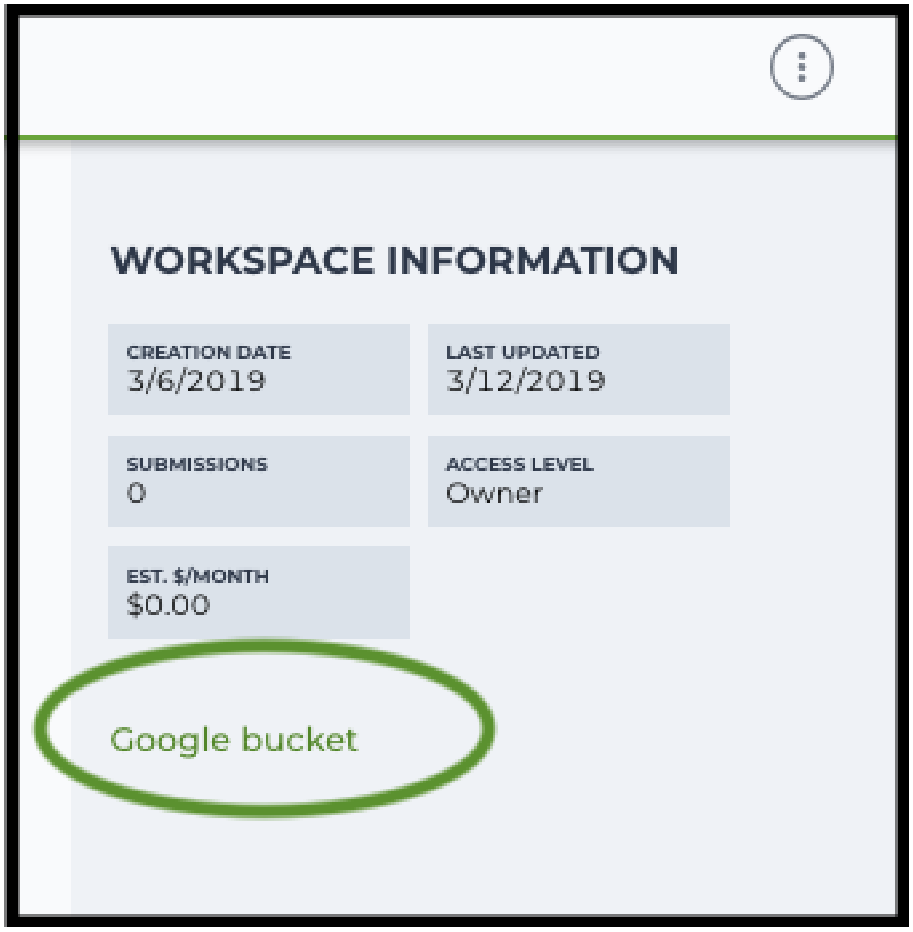
| Name | Type | Description |
|---|---|---|
| reference | File | The custom Genome reference generated. Its default file name is `rsem_ref.tar.gz`. |

## 8.6 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.

   WORKSPACE INFORMATION

   CREATION DATE
   3/6/2019

   LAST UPDATED
   3/12/2019

   SUBMISSIONS
   0

   ACCESS LEVEL
   Owner

   EST. $/MONTH
   $0.00

   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:

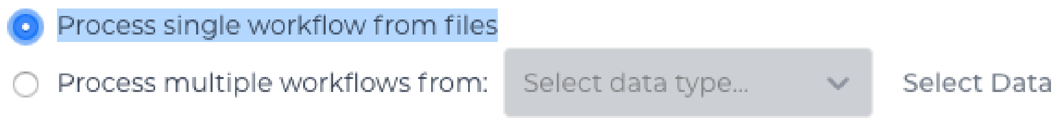
   ```
   gsutil cp /foo/bar/projects/sample_sheet.csv gs://fc-e0000000-0000-0000-
   ↪0000-000000000000/
   ```

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 `Process single workflow from files` as below


Process single workflow from files
Process multiple workflows from:
Select data type...
Select Data

and click the `SAVE` button.

### 8.6.1 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_fraction | Threshold for the merge procedure (default 0.2) |
| dropest_max_cb_merge_edit_distance | Max edit distance between barcodes (default 2) |
| dropest_max_umi_merge_edit_distance | 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_precise | Use precise 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",
```

(continues on next page)

(continued from previous page)

```
        "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.

### 8.6.2 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 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 cumulus 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

## 8.7 Demultiplex cell-hashing/nuclei-hashing data using demuxEM or prepare for CITE-Seq analysis

Follow the steps below to run **cumulus** for cell-hashing/nuclei-hashing/CITE-Seq data on Terra.

1. Run Cell Ranger tool to generate raw gene count matrices and antibody hashtag data.

   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_be_matrices_h5.h5`) and ADT count matrix (e.g. `sample_1_ADT.csv`) for each sample.

2. Create a sample sheet, **sample_sheet_hashing.csv**, which describes the metadata for each pair of RNA and antibody hashtag data. The sample sheet should contain 4 columns — *OUTNAME*, *RNA*, *ADT*, and *TYPE*. *OUTNAME* is the output name for one pair of RNA and ADT data. *RNA* and *ADT* are the raw gene count matrix and the ADT count matrix generated in Step 1, respectively. *TYPE* is the assay type, which can be `cell-hashing`, `nuclei-hashing`, or `cite-seq`.

   Example:

   ```
   OUTNAME,RNA,ADT,TYPE
   sample_1,gs://fc-e0000000-0000-0000-0000-000000000000/my_dir/sample_1/raw_
   ↪gene_bc_matrices_h5.h5,gs://fc-e0000000-0000-0000-0000-000000000000/my_
   ↪dir/sample_1_ADT/sample_1_ADT.csv,cell-hashing
   sample_2,gs://fc-e0000000-0000-0000-0000-000000000000/my_dir/sample_2/raw_
   ↪feature_bc_matrices.h5,gs://fc-e0000000-0000-0000-0000-000000000000/my_
   ↪dir/sample_2_ADT/sample_2_ADT.csv,nuclei-hashing
   ```

   Note that in the example above, sample_2 is 10x genomics' v3 chemistry. Cumulus can automatically detect v2/v3 chemistry when loading hdf5 files.

3. Create an additional antibody-control sheet **antibody_control.csv** if you have CITE-Seq data. This sheet contains 2 columns — *Antibody* and *Control*.

   Example:

   ```
   Antibody,Control
   CD8,Mouse-IgG1
   HLA-ABC,Mouse-IgG2a
   CD45RA,Mouse-IgG2b
   ```

4. Upload your sample sheets to the Google bucket of your workspace.

   Example:

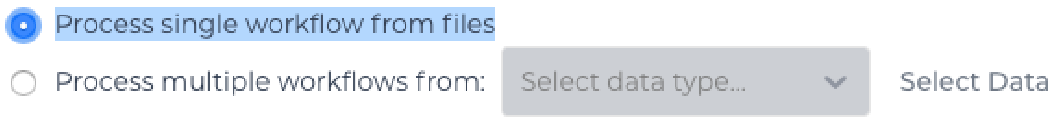
   ```
   gsutil cp /foo/bar/projects/my_sample_sheet_hashing.csv gs://fc-e0000000-
   ↪0000-0000-0000-000000000000/
   gsutil cp /foo/bar/projects/antibody_control.csv gs://fc-e0000000-0000-
   ↪0000-0000-000000000000/
   ```

5. Import *cumulus_hashing_cite_seq* to your workspace.

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

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

6. In your workspace, open `cumulus_hashing_cite_seq` in `WORKFLOWS` tab. Select `Process single workflow from files` as below

   
   Process single workflow from files
   Process multiple workflows from:
   Select data type...
   Select Data

   and click the `SAVE` button.

### 8.7.1 cumulus_hashing_cite_seq inputs:

| Name | Description | Example | Default |
|---|---|---|---|
| **input_sample_sheet** | Input CSV file describing metadata of RNA and ADT data pairing | "gs://fc-e0000000-0000-0000-0000-000000000000/sample_sheet_hashing.csv" | |
| **output_directory** | This is the output directory (gs url + path) for all results. There will be one folder per RNA-ADT data pair under this directory | "gs://fc-e0000000-0000-0000-0000-000000000000/my_demux_dir" | |
| genome | Reference genome name. If not provided, cumulus will infer the genome name from data | "GRCh38" | |
| demuxEM_min_num_genes | demuxEM parameter. Only demultiplex cells/nuclei with at least <demuxEM_min_num_genes> expressed genes | 200 | 100 |
| demuxEM_alpha_on_samples | demuxEM parameter. The Dirichlet prior concentration parameter (alpha) on samples. An alpha value < 1.0 will make the prior sparse. | 2.0 | 0.0 |
| demuxEM_min_num_umis | demuxEM parameter. Only demultiplex cells/nuclei with at least <demuxEM_min_num_umis> of UMIs. | 200 | 100 |
| demuxEM_min_signal_hashtag | demuxEM parameter. Any cell/nucleus with less than <count> hashtags from the signal will be marked as unknown. [default: 10.0] | 10.0 | 10.0 |
| demuxEM_random_state | 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" | |
| antibody_control_csv | merge_rna_adt parameter. This is a CSV file containing the IgG control information for each antibody. | "gs://fc-e0000000-0000-0000-0000-000000000000/antibody_control.csv" | |
| cumulus_version | cumulus version to use. Versions available: 0.10.0. | "0.10.0" | "0.10.0" |
| docker_registry | Docker registry to use. Options: <br>• "cumulusprod/" for Docker Hub images; <br>• "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-east1-d us-west1-a us-west1-b" |
| num_cpu | Number of CPUs per cumulus_hashing_cite_seq job | 8 | 8 |
| memory | Memory size string | "10G" | "10G" |
| disk_space | Total disk space in GB | 20 | 20 |
| preemptible | Number of preemptible tries | 2 | 2 |

### 8.7.2 cumulus_hashing_cite_seq outputs

See the table below for important *cumulus_hashing_cite_seq* outputs:

| Name | Type | Description |
|---|---|---|
| output_folder | Array[String] | A list of google bucket urls containing results for every RNA-ADT data pairs. |

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

| Name | Description |
|---|---|
| output_name_demux.h5ad | Demultiplexed RNA count matrix in h5ad format. |
| output_name_demux.h5sc | RNA expression matrix with demultiplexed sample identities in cumulus hdf5 (h5sc) format. |
| output_name_ADTs.h5ad | Antibody tag matrix in h5ad format. |
| 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.bar.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 folder of each CITE-Seq RNA-ADT data pair, you can find the following file:

| Name | Description |
|---|---|
| output_name.h5sc | A Cumulus hdf5 format (h5sc) file containing both RNA and ADT count matrices. |

---

### 8.7.3 Load demultiplexing results into Python and R

To load demultiplexing results into Python, you need to install Python package anndata first. Then follow the codes below:

```
import anndata
data = anndata.read_h5ad('output_name_demux.h5ad')
```

Once you load the data object, you can find predicted droplet types (singlet/doublet/unknown) in `data.obs['demux_type']`. You can find predicted sample assignments in `data.obs['assignment']`. You can find estimated sample fractions (sample1, sample2, . . . , samplen, background) for each droplet in `data.obsm['raw_probs']`.

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

```
library(reticulate)
ad <- import("anndata", convert = FALSE)
data <- ad$read_h5ad("output_name_demux.h5ad")
```

Results are in `data$obs['demux_type']`, `data$obs['assignment']`, and `data$obsm['raw_probs']`.

## 8.8 Run Cumulus for sc/snRNA-Seq data analysis

### 8.8.1 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 10x channel. 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/filtered_gene_bc_matrices_h5.h5`.
- 10x format with v3 chemistry. For example, `gs://fc-e0000000-0000-0000-0000-000000000000/my_dir/sample_1/filtered_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`.

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/filtered_gene_bc_matrices_h5.h5
sample_2,bone_marrow,NextSeq,2,GRCh38,gs://fc-e0000000-0000-0000-0000-
↪000000000000/my_dir/sample_2/filtered_gene_bc_matrices_h5.h5
sample_3,pbmc,NextSeq,1,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/
↪my_dir/sample_3/filtered_feature_bc_matrices.h5
sample_4,pbmc,NextSeq,2,GRCh38,gs://fc-e0000000-0000-0000-0000-000000000000/
↪my_dir/sample_4/filtered_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/
```

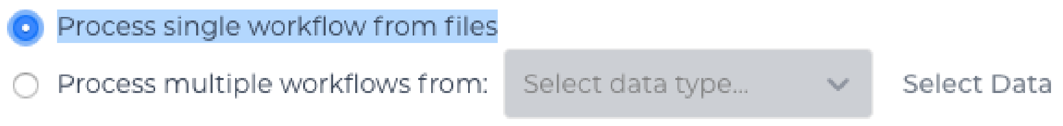
where `/foo/bar/projects/my_count_matrix.csv` is the path to your sample sheet in local machine, and `gs://fc-e0000000-0000-0000-0000-000000000000/` is the location on Google bucket to hold it.

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 `Process single workflow from files` as below



Process single workflow from files
Process multiple workflows from:
Select data type...
Select Data

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 formats (hdf5 or mtx);
- HCA DCP mtx and loom formats;
- Drop-seq dge 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). Notice that for dge and loom files, the `genome` field in global inputs is required.

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

---

### 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. **organize_results**. Copy analysis results from execution environment to destination location on Google bucket.

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_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" | |
| genome | A string contains comma-separated genome names. Cumulus will read all groups associated with genome names in the list from the hdf5 file. If genome is None, all groups will be considered. | "GRCh38" | |
| cumulus_version | cumulus version to use. Versions available: 0.10.0. | "0.10.0" | "0.10.0" |
| docker_registry | Docker registry to use. Options: <br>• "cumulusprod/" for Docker Hub images; <br>• "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-east1-d us-west1-a us-west1-b" |
| 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" | |
| minimum_number_of_genes | Only keep barcodes with at least this number of expressed genes | 100 | 100 |
| is_dropseq | If inputs are dropseq data | true | false |

#### aggregate_matrices output

| Name | Type | Description |
|---|---|---|
| output_h5sc | File | Aggregated count matrix in Cumulus hdf5 (h5sc) format |

### cluster

#### cluster inputs

| Name | Description | Example | Default |
|---|---|---|---|
| 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_on_raw | If input are raw 10x matrix, which include all barcodes, perform a pre-filtration step to keep the data size small. In the pre-filtration step, only keep cells with at least <min_genes_on_raw> of genes | 100 | 100 |
| cite_seq | Data are CITE-Seq data. cumulus will perform analyses on RNA count matrix first. Then it will attach the ADT matrix to the RNA matrix with all antibody names changing to 'AD-' + antibody_name. Lastly, it will embed the antibody expression using FIt-SNE (the basis used for plotting is 'citeseq_fitsne') | false | false |

Continued on next page

Table 2 – continued from previous page

| Name | Description | Example | Default |
|---|---|---|---|
| cite_seq_capping | For CITE-Seq surface protein expression, make all cells with expression > <percentile> to the value at <percentile> to smooth outlier. Set <percentile> to 100.0 to turn this option off. | 10.0 | 99.99 |
| 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 |
| output_filtration_results | If write cell and gene filtration results to a spreadsheet | true | true |
| plot_filtration_results | 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_seurat_compatible | Generate Seurat-compatible h5ad file. Caution: File size might be large, do not turn this option on for large data sets. | false | false |
| output_loom | If generate loom-formatted file | false | false |
| output_parquet | If generate parquet-formatted file | false | false |
| min_genes | Only keep cells with at least <min_genes> of genes | 500 | 500 |
| 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 | 100 | 100 |
| max_umis | Only keep cells with less than <max_umis> of UMIs | 600000 | 600000 |
| mito_prefix | Prefix of mitochondrial gene names. This is to identify mitochondrial genes. | “mt-“ | “MT-“ |
| percent_mito | Only keep cells with mitochondrial ratio less than <percent_mito>% of total counts | 50 | 10.0 |
| 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:<br>• “pegasus”: New selection method proposed in Pegasus, the analysis module of Cumulus workflow.<br>• “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 2 – continued from previous page

| Name | Description | Example | Default |
|---|---|---|---|
| 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 |
| nPC | Number of principal components | 50 | 50 |
| 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 |

Continued on next page

Table 2 – continued from previous page

| Name | Description | Example | Default |
|---|---|---|---|
| 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 | true | true |
| run_umap | Run UMAP for visualization | false | false |
| 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 |
| net_down_sample_fraction | 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 | Basis name 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_h5ad** | File | Output file in h5ad format (output_name.h5ad).<br>To load this file in Python, you need to first install Pegasus on your local machine. Then use `import pegasus as pg; data = pg.read_input('output_name.h5ad')` in Python interpreter.<br>The log-normalized expression matrix is stored in `data.X` as a CSR-format sparse matrix.<br>The `obs` field contains cell related attributes, including clustering results.<br>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;<br>`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 clustring algorithms;<br>The `var` field contains gene related attributes.<br>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.<br>The `obsm` field records embedding coordinates.<br>For example, `data.obsm['X_pca']` records PCA coordinates, `data.obsm['X_tsne']` records t-SNE coordinates,<br>`data.obsm['X_umap']` records UMAP coordinates, `data.obsm['X_diffmap']` records diffusion map coordinates,<br>`data.obsm['X_diffmap_pca']` records the first 3 PCs by projecting the diffusion components using PCA,<br>and `data.obsm['X_fle']` records the force-directed layout coordinates from the diffusion components.<br>The `varm` field records DE analysis results if performed: `data.varm['de_res']`.<br>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_seurat_h5ad | File | h5ad file in seurat-compatible manner. This file can be loaded into R and converted into a Seurat object (see here for instructions) |
| output_filt_xlsx | File | Spreadsheet containing filtration results (output_name.filt.xlsx).<br>This file has two sheets — Cell filtration stats and Gene filtration stats.<br>The first sheet records cell filtering results and it has 10 columns:<br>Channel, channel name; kept, number of cells kept; median_n_genes, median number of expressed genes in kept cells; median_n_umis, median number of UMIs in kept cells;<br>median_percent_mito, median mitochondrial rate as UMIs between mitochondrial genes and all genes in kept cells;<br>filt, number of cells filtered out; total, total number of cells before filtration, if the input contain all barcodes, this number is the cells left after |

### de_analysis

#### de_analysis inputs

| Name | Description | Example | Default |
|---|---|---|---|
| perform_de_analysis | If perform differential expression (DE) analysis | 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 |
| auc | Calculate area under ROC (AUROC) | true | true |
| fisher | Calculate Fisher's exact test | true | true |
| t_test | Calculate Welch's t-test. | true | true |
| mwu | Calculate Mann-Whitney U test | false | false |
| 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: "t", "fisher", or "mwu" | "t" | "t" |
| organism | Organism, could either be "human_immune", "mouse_immune", "human_brain", "mouse_brain" 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 | File | h5ad-formatted results with DE results updated (output_name.h5ad) |
| output_de_xlsx | File | Spreadsheet reporting DE results (output_name.de.xlsx) |
| output_markers_xlsx | File | An excel spreadsheet containing detected markers. 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_name.markers.xlsx) |
| output_anno_file | File | Annotation file (output_name.anno.txt) |

#### 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_diffmap | Takes the format of "attr,attr,. . . ,attr". If non-empty, generate attr colored 3D interactive plot. The 3 coordinates are the first 3 PCs of all diffusion components | "louvain_labels,Donor" | None |
| plot_citeseq_fitsne | plot cells based on FIt-SNE coordinates estimated from antibody expressions. Takes the format of "attr,attr,. . . ,attr". If non-empty, plot attr colored FIt-SNEs 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". | "leiden_labels,Donor" | None |

#### plot outputs

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

---

#### Generate SCP Output

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

#### scp_output inputs

| Name | Description | Example | Default |
|---|---|---|---|
| generate_scp_outputs | 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. |

---

### 8.8.2 Run CITE-Seq analysis

To run CITE-Seq analysis, turn on `cite_seq` option 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.

---

### 8.8.3 Run subcluster analysis

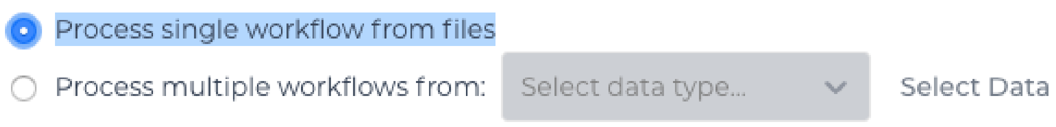
Once we have **cumulus** outputs, we could further analyze a subset of cells by running **cumulus_subcluster**. To run **cumulus_subcluster**, follow the following steps:

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 `Process single workflow from files` as below

   
   Process single workflow from files
   Process multiple workflows from:
   Select data type...
   Select Data

   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_loom_file | File | Generated loom file (output_name.loom) |
| output_parquet_file | File | Generated PARQUET file that contains metadata and expression levels for every gene (output_name.parquet) |
| 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 |

### 8.8.4 Load Cumulus results into Seurat

First, you need to set `output_seurat_compatible` field to `true` in cumulus cluster inputs to generate a Seurat-compatible output file `output_name.seurat.h5ad`, in addition to the normal result `output_name.h5ad`.

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

Execute the R code below to load the results 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.seurat.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.

### 8.8.5 Visualize Cumulus results in Python

Ensure you have Pegasus installed.

Download your analysis result data, say `output_name.h5ad`, from Google bucket to your local machine.

Load the output:

```
import pegasus as pg
adata = pg.read_input("output_name.h5ad")
```

Violin plot of the computed quality measures:

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

t-SNE plot colored by genes of interest:

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

## 8.9 Run Terra pipelines via command line

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

### 8.9.1 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
```

### 8.9.2 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
   git clone https://github.com/klarman-cell-observatory/altocumulus.git
   cd altocumulus
   pip install -e .
   ```

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

### 8.9.3 Run Terra workflows via `alto fc_run`

**alto fc_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 (e.g. fastq directories) 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.<br><*METHOD*> is of format *Namespace/Name* (e.g. `cumulus/cellranger_workflow`).<br>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.<br><*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.<br>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. |

## Example

This example shows how to use `alto fc_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 fc_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 fc_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 fc_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.

## 8.10 Examples

### 8.10.1 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 `/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 option `-m` means copy in parallel, `-r` means copy the directory recursively, and `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`, on your local machine with content below:

```
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/
```

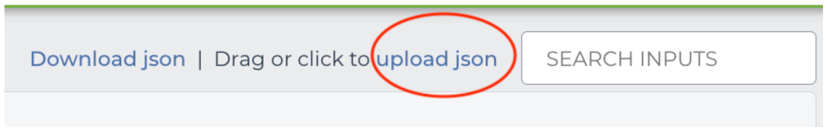
Now we are ready to set up **cellranger_workflow** workflow for this phase. If your workspace doesn't have this workflow, import it to your workspace by following Step 5 and 6 of cellranger_workflow documentation.

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

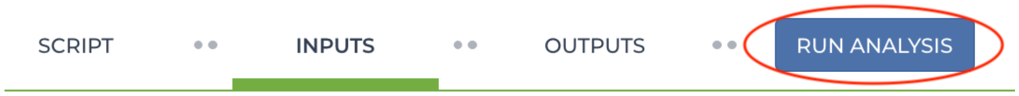
```
{
        "cellranger_workflow.input_csv_file" : "gs://fc-e0000000-0000-0000-0000-
↪000000000000/my-dir/cellranger_sample_sheet.csv",
        "cellranger_workflow.output_directory" : "gs://fc-e0000000-0000-0000-0000-
↪000000000000/my-dir"
}
```

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, `cell_hashing_sample_sheet.csv`, with the following content:

```
OUTNAME,RNA,ADT,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 **ADT** columns specify the RNA and ADT meta-data of samples, and **TYPE** is `cell-hashing` for this phase.

Then upload it to Google bucket:

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

2. If your workspace doesn't have **cumulus_hashing_cite_seq** workflow, import it to your workspace by following Step 5 and 6 of cumulus_hashing_cite_seq documentation.

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

```
{
        "cumulus_hashing_cite_seq.input_sample_sheet" : "gs://fc-e0000000-0000-
↪0000-0000-000000000000/my-dir/cell_hashing_sample_sheet.csv",
        "cumulus_hashing_cite_seq.output_directory" : "gs://fc-e0000000-0000-0000-
↪0000-000000000000/my-dir/",
        "cumulus_hashing_cite_seq.demuxEM_min_num_genes" : 500,
        "cumulus_hashing_cite_seq.demuxEM_generate_diagnostic_plots" : true
}
```

For the details on these options, please refer to cell-hashing/nuclei-hashing inputs.

4. On the page of cumulus_hashing_cite_seq workflow, upload `cell_hashing_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.h5sc`, 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,TYPE
exp_raw,gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/exp/exp_demux.h5sc,
↪gs://fc-e0000000-0000-0000-0000-000000000000/my-dir/sample_cite_seq.csv,cite-seq
```

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, and the sample **TYPE** is now `cite-seq`.

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_hashing_cite_seq.input_sample_sheet" : "gs://fc-e0000000-0000-
↪0000-0000-000000000000/my-dir/cite_seq_sample_sheet.csv",
        "cumulus_hashing_cite_seq.output_directory" : "gs://fc-e0000000-0000-0000-
↪0000-000000000000/my-dir/",
        "cumulus_hashing_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 cell-hashing/nuclei-hashing inputs.

3. On **cumulus_hashing_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.h5sc`, 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.h5sc
```

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_name" : "gs://fc-e0000000-0000-0000-0000-000000000000/my-
↪dir/results/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,
```

(continues on next page)

(continued from previous page)

```
        "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.h5sc",
        ... ...
}
```

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/` to check:

- `exp_merged_out.h5sc`: 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 fc_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 `cell_hashing_sample_sheet.csv` file, prepare JSON file `cell_hashing_inputs.json` as below:

```
{
        "cumulus_hashing_cite_seq.input_sample_sheet" : "cell_hashing_sample_
↪sheet.csv",
        ... ...
}
```

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

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

```
alto fc_run -m cumulus/cumulus_hashing_cite_seq -w ws-lab/ws-01 --bucket-folder␣
↪my-dir -i cell_hashing_inputs.json -o cell_hashing_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_hashing_cite_seq.input_sample_sheet" : "cite_seq_sample_sheet.csv
↪",
        ... ...
}
```

where all the rest parameters remain the same as in Phase 3.

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

```
alto fc_run -m cumulus/cumulus_hashing_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.h5sc",
        ... ...
}
```

where all the rest parameters remain the same.

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

```
alto fc_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.

## 8.11 Contribution