tibanna Documentation

Release 4.0.0

4DN DCIC

Sep 15, 2023

Contents

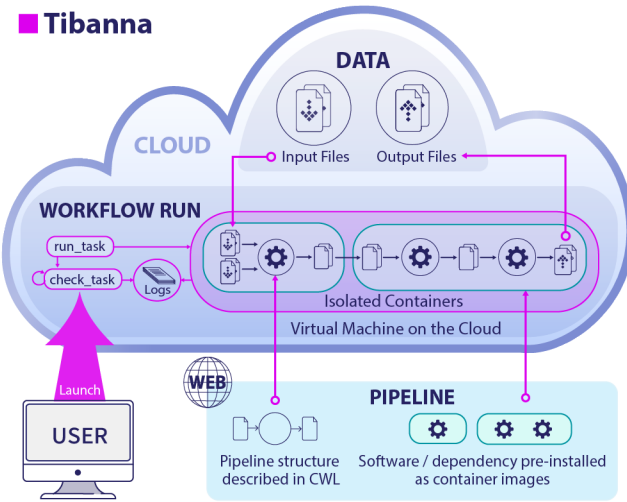Tibanna is a software tool that helps you run genomic pipelines on the Amazon (AWS) cloud. It is also used by 4DN-DCIC (4D Nucleome Data Coordination and Integration Center) to process data.

What do I need to run pipelines using Tibanna?

- Your pipeline
- Your data
- An Amazon Web Services (AWS) cloud account
- Tibanna

1.1 Pipeline

- The commands to run your pipeline must be written in either Common Workflow Language (CWL) (recommended), Workflow Description Language (WDL) (only basic support), Snakemake or a list of shell commands.
- The pipelines may be local CWL/WDL/Snakemake files or they should be downloadable from public urls.
- With the exception of Snakemake, your pipeline and dependencies must be pre-installed as a Docker image (https://www.docker.com/). For Snakemake, conda can be used instead.

1.2 Data

- Your data must be in AWS S3 buckets.

1.3 AWS cloud account

- Confirm that you can log in to AWS.

1.4 Tibanna

- Tibanna is open-source and can be found on github.

- Once installed, Tibanna can be run either as a set of commend-line tools or a set of python modules.

Command-line tools

```
$ tibanna run_workflow --input-json=run1.json
```

Python

```
>>> from tibanna.core import API
>>> API().run_workflow(input_json='run1.json')  # input_json is a json file or a dict
↪object
```

Contents:

1.4.1 News and updates

Publication

- **May 15, 2019** Tibanna paper is out on Bioinformatics now! https://doi.org/10.1093/bioinformatics/btz379

- **Apr 18. 2019** A newer version of the Tibanna paper is out on Biorxiv! https://www.biorxiv.org/content/10.1101/440974v3

- **Oct 11. 2018** Tibanna paper preprint is out on Biorxiv! https://www.biorxiv.org/content/early/2018/10/11/440974

Version updates

For more recent version updates, check out Tibanna releases

Sep 15, 2023 The latest version is now 4.0.0_.

- Support for Python 3.7 has been dropped
- Added support for Python 3.9 and 3.10

Nov 18, 2022 The latest version is now 3.0.0_.

- Tibanna now supports AWS Graviton-based instances.
- The instance type configuration now allows single instances (e.g., `t3.micro`) and lists (e.g., `[t3.micro, t3.small]`). If `spot_instance` is enabled, Tibanna will run the workflow on the instance with the highest available capacity. If `spot_instance` is disabled, it will run the workflow on the cheapest instance in the list.
- The option `other_instance_types` for `behavior_on_capacity_limit` has been removed. It will fall back to `wait_and_retry`.

Mar 10, 2022 The latest version is now 2.0.0.

- The default Python version for Tibanna is now 3.8 (or 3.7). Python 3.6 is no longer supported.

Sep 16, 2019 The latest version is now 0.9.1.

- A new functionality of generating a resource metrics report html is now added! This report includes a graph of CPU/Memory/disk space utilization and usage at 1min interval, as well as a table of summary metrics. - After each run, an html report gets automatically added to the `log_bucket` which can be viewed using a Web Browser. However, for this to take effect, the unicorn must be redeployed. - The new `plot_metrics` function of CLI (`tibanna plot_metrics -h`) allows users to create the resource metrics report before a run it complete. - The same function can be used through Python API (`API().plot_metrics(job_id=<jobid>, ...)`)

- A new functionality `cost` is added to the tibanna CLI/API, to retrieve the cost of a specific run. - `tibanna cost --job-id=<jobid>` - It usually takes a day for the cost to be available. - The cost can also be added to the resource plot, by

```
tibanna cost -j <jobid> --update-tsv
tibanna plot_metrics -j <jobid> --update-html-only --force-upload
```

  - A new dynamoDB-based jobID indexing is enabled! This allows users to search by jobid without specifying step function name and even after the execution expires (e.g. `tibanna log`, `tibanna plot_metrics`) - To use this feature, the unicorn must be redeployed. Only the runs created after the redeployment would be searchable using this feature. When the jobid index is not available, tibanna automatically switches to the old way of searching. - DynamoDB may add to the cost but very minimally (up to $0.01 per month in case of 4DN)

  - `Benchmark 0.5.5` is used now for 4DN pipelines.

  - `run_workflow` now has `--do-not-open-browser` option that disables opening the Step function execution on a Web Browser.

Aug 14, 2019 The latest version is now 0.9.0.

- `root_ebs_size` now supported (default 8) as a config field. (useful for large docker images or multiple docker images, which uses root EBS)

- `TIBANNA_AWS_REGION` and `AWS_ACCOUNT_NUMBER` no longer required as environment variables.

Jul 22, 2019 The latest version is now 0.8.8.

- Fixed installation issue caused by `python-lambda-4dn`

- Input file can now be a directory for `shell` and `snakemake` - e.g. `"file:///data1/shell/somedir" : "s3://bucketname/dirname"`

- Output target can now be a directory for `shell` and `snakemake` - e.g. `"file:///data1/shell/somedir": "dirname"`

Jul 8, 2019 The latest version is now 0.8.7.

- ec2 termination policy is added to usergroup to support `kill` function

- `run_workflow verbose` option is now passed to `dynamodb`

Jun 25, 2019 The latest version is now 0.8.6.

- A newly introduced issue of not reporting `Metric` after the run is now fixed.

- With `tibanna log`, when the log/postrunjson file is not available, it does not raise an error but prints a message.

- Benchmark `0.5.4` is used instead of `0.5.3` for 4DN pipelines.

Jun 14, 2019 The latest version is now 0.8.5.

- A newly introduced bug in the `rerun` cli (not working) now fixed.

Jun 12, 2019 The latest version is now 0.8.4.

- The issue of auto-determined EBS size being sometimes not an integer fixed.

- Now input files in the unicorn input json can be written in the format of `s3://bucket/key` as well as `{'bucket_name': bucket, 'object_key': key}`

- command can be written in the format of a list for aesthetic purpose (e.g. `[command1, command2, command3]` is equivalent to `command1; command2; command3`)

Jun 10, 2019 The latest version is now 0.8.3.

- A newly introduced issue of `--usergroup` not working properly with `deploy_unicorn`/`deploy_core` is now fixed.

- Now one can specify `mem` (in GB) and `cpu` instead of `instance_type`. The most cost-effective instance type will be auto-determined.

- Now one can set `behavior_on_capacity_limit` to `other_instance_types`, in which case tibanna will try the top 10 instance types in the order of decreasing hourly cost.

- EBS size can be specified in the format of `3x`, `5.5x`, etc. to make it 3 (or 5.5) times the total input size.

Jun 3, 2019 The latest version is now 0.8.2.

- One can now directly send in a command and a container image without any CWL/WDL (language = `shell`).

- One can now send a local/remote(http or s3) Snakemake workflow file to awsem and run it (either the whole thing, a step or multiple steps in it). (language = `snakemake`)

- Output target and input file dictionary keys can now be a file name instead of an argument name (must start with `file://`) - input file dictionary keys must be `/data1/input`, `/data1/out` or either `/data1/shell` or `/data1/snakemake` (depending on the language option).

- With shell / snakemake option, one can also `exec` into the running docker container after sshing into the EC2 instance.

- The `dependency` field can be in args, config or outside both in the input json.

May 30, 2019 The latest version is now 0.8.1.

- `deploy_core` (and `deploy_unicorn`) not working in a non-venv environment fixed

- local CWL/WDL files and CWL/WDL files on S3 are supported.

- new issue with opening the browser with `run_workflow` fixed

May 29, 2019 The latest version is now 0.8.0.

- Tibanna can now be installed via `pip install tibanna`! (no need to `git clone`)

- Tibanna now has its own CLI! Instead of `invoke run_workflow`, one should use `tibanna run_workflow`.

- Tibanna's API now has its own class! Instead of `from core.utils import run_workflow`, one should use the following.

```python
from tibanna.core import API
API().run_workflow(...)
```

- The API `run_workflow()` can now directly take an input json file as well as an input dictionary (both through `input_json` parameter).

- The `rerun` CLI now has `--appname_filter` option exposed

- The `rerun_many` CLI now has `--appname-filter`, `--shutdown-min`, `--ebs-size`, `--ebs-type`, `--ebs-iops`, `--key-name`, `--name` options exposed. The API also now has corresponding parameters.

- The `stat` CLI now has API and both has a new parameter *n* (*-n*) that prints out the first n lines only. The option `-v` (`--verbose`) is not replaced by `-l` (`--long`)

May 15, 2019 The latest version is now 0.7.0.

- Now works with **Python3.6** (2.7 is deprecated!)

- newly introduced issue with non-list secondary output target handling fixed

- fixed the issue with top command reporting from ec2 not working any more

- now the *run_workflow* function does not later the original input dictionary

- auto-terminates instance when CPU utilization is zero (inactivity) for an hour (mostly due to aws-related issue but could be others).

- The *rerun* function with a run name that contains a uuid at the end(to differentiate identical run names) now removes it from run_name before adding another uuid.

Mar 7, 2019 The latest version is now 0.6.1.

- Default **public bucket access is deprecated** now, since it also allows access to all buckets in one's own account. The users must specify buckets at deployment, even for public buckets. If the user doesn't specify any bucket, the deployed Tibanna will only have access to the public tibanna test buckets of the 4dn AWS account.

- A newly introduced issue of `rerun` with no `run_name` in `config` fixed.

Feb 25, 2019 The latest version is now 0.6.0.

- The input json can now be simplified.

  - `app_name`, `app_version`, `input_parameters`, `secondary_output_target`, `secondary_files` fields can now be omitted (now optional)

  - `instance_type`, `ebs_size`, `EBS_optimized` can be omitted if benchmark is provided (`app_name` is a required field to use benchmark)

  - `ebs_type`, `ebs_iops`, `shutdown_min` can be omitted if using default ('gp2', '', 'now', respectively)

  - `password` and `key_name` can be omitted if user doesn't care to ssh into running/failed instances

- issue with rerun with a short run name containing uuid now fixed.

Feb 13, 2019 The latest version is now 0.5.9.

- Wrong requirement of `SECRET` env is removed from unicorn installation

- deploy_unicorn without specified buckets also works

- deploy_unicorn now has `--usergroup` option

- cloud metric statistics aggregation with runs > 24 hr now fixed

- `invoke -l` lists all invoke commands

- `invoke add_user`, `invoke list` and `invoke users` added

- `log()` function not assuming default step function fixed

- `invoke log` working only for currently running jobs fixed

Feb 4, 2019 The latest version is now 0.5.8.

- `invoke log` can be used to stream log or postrun json file.

- postrun json file now contains Cloudwatch metrics for memory/CPU and disk space for all jobs.

- `invoke rerun` has config override options such as `--instance-type`, `shutdown-min`, `ebs-size` and `key-name` to rerun a job with a different configuration.

Jan 16, 2019 The latest version is now 0.5.7.

- Spot instance is now supported. To use a spot instance, use `"spot_instance":` `true` in the `config` field in the input execution json.

```
"spot_instance": true,
"spot_duration": 360
```

Dec 21, 2018 The latest version is now 0.5.6.

- CloudWatch set up permission error fixed

- *invoke kill* works with jobid (previously it worked only with execution arn)

```
invoke kill --job-id=<jobid> [--sfn=<stepfunctionname>]
```

- A more comprehensive monitoring using *invoke stat -v* that prints out instance ID, IP, instance status, ssh key and password.

- To update an existing Tibanna on AWS, do the following

```
invoke setup_tibanna_env --buckets=<bucket1>,<bucket2>,...
invoke deploy_tibanna --sfn-type=unicorn --usergroup=<usergroup_name>
```

e.g.

```
invoke setup_tibanna_env --buckets=leelab-datafiles,leelab-tibanna-
↪log
invoke deploy_tibanna --sfn-type=unicorn --usergroup=default_3225
```

Dec 14, 2018 The latest version is now 0.5.5.

- Now memory, Disk space, CPU utilization are reported to CloudWatch at 1min interval from the Awsem instance.

- To turn on Cloudwatch Dashboard (a collective visualization for all of the metrics combined), add `"cloudwatch_dashboard" :` `true` to `"config"` field of the input execution json.

Dec 14, 2018 The latest version is now 0.5.4.

- Problem of EBS mounting with newer instances (e.g. c5, t3, etc) fixed.

- Now a common AMI is used for *CWL v1*, *CWL draft3* and *WDL* and it is handled by *awsf/aws_run_workflow_generic.sh*

    - To use the new features, redeploy *run_task_awsem* lambda.

```
git pull
invoke deploy_core run_task_awsem --usergroup=<usergroup>   # e.g.␣
↪usergroup=default_3046
```

Dec 4, 2018 The latest version is now 0.5.3.

- For WDL workflow executions, a more comprehensive log named `<jobid>.debug.tar.gz` is collected and sent to the log bucket.

- A file named `<jobid>.input.json` is now sent to the log bucket at the start of all Pony executions.

- Space usage info is added at the end of the log file for WDL executions.

- `bigbed` files are registered to Higlass (pony).

- Benchmark for `encode-chipseq` supported. This includes double-nested array input support for Benchmark.

- `quality_metric_chipseq` and `quality_metric_atacseq` created automatically (Pony).

- An empty extra file array can be handled now (Pony).

- When Benchmark fails, now Tibanna returns which file is missing.

Nov 20, 2018 The latest version is now 0.5.2.

- User permission error for setting postrun jsons public fixed

- `--no-randomize` option for `invoke setup_tibanna_env` command to turn off adding random number at the end of usergroup name.

- Throttling error upon mass file upload for md5/fastqc trigger fixed.

Nov 19, 2018 The latest version is now 0.5.1.

- Conditional alternative outputs can be assigned to a global output name (useful for WDL)

Nov 8, 2018 The latest version is now 0.5.0.

- WDL and Double-nested input array is now also supported for Pony.

Nov 7, 2018 The latest version is now 0.4.9.

- Files can be renamed upon downloading from s3 to an ec2 instance where a workflow will be executed.

Oct 26, 2018 The latest version is now 0.4.8.

- Double-nested input file array is now supported for both CWL and WDL.

Oct 24, 2018 The latest version is now 0.4.7.

- Nested input file array is now supported for both CWL and WDL.

Oct 22, 2018 The latest version is now 0.4.6.

- Basic *WDL* support is implemented for Tibanna Unicorn!

Oct 11. 2018 The latest version is now 0.4.5.

- Killer CLIs `invoke kill` is available to kill specific jobs and `invoke kill_all` is available to kill all jobs. They terminate both the step function execution and the EC2 instances.

1.4.2 Simple Example Pipeline

hello

We will run a command that prints *hello world* through Tibanna. To do this, we just need to prepare for a job description json and run tibanna.

Job description

To run the pipeline on a specific input file using Tibanna, we need to create an *job description* file for each execution (or a dictionary object if you're using Tibanna as a python module).

The job description for running shell commands requires `command` and `container_image` fields. The former is a list of commands and the latter is the Docker image name. Here, we use `ubuntu:20.04` image and use an `echo` command. Notice that double-quotes are escaped inside the command string. We're passing an environment variable `$NAME` through the field `input_env`. Also notice that the environment variable's `$` sign is prefixed with an escaped backslash in the `command` string.

In the following example, the output file `hello.txt` in the same directory is copied to the output bucket `my-tibanna-test-bucket` as `some_sub_dirname/my_first_hello.txt`.

This json can be found at https://github.com/4dn-dcic/tibanna/blob/master/examples/hello/hello_shell_input.json

```
{
  "args": {
    "container_image": "ubuntu:20.04",
    "command": ["echo \"Hello world, \\$NAME!\" > hello.txt"],
    "language": "shell",
    "input_files": {},
    "secondary_files": {},
    "input_parameters": {},
    "input_env": {"NAME": "Soo"},
    "output_S3_bucket": "my-tibanna-test-bucket",
    "output_target": {
      "file:///data1/shell/hello.txt": "some_sub_dirname/my_first_hello.txt"
    },
    "secondary_output_target": {}
  },
  "config": {
    "ebs_size": 10,
    "instance_type": "t3.micro",
    "EBS_optimized": true,
    "password": "whateverpasswordworks",
    "log_bucket": "my-tibanna-test-bucket"
  }
}
```

Tibanna run

To run Tibanna,

1. Sign up for AWS

2. Install and configure `awscli`

   see Before_using_Tibanna

3. Install Tibanna on your local machine

   see Installation

4. Deploy Tibanna (link it to the AWS account)

   see Installation

5. Run workflow as below.

```
cd tibanna
tibanna run_workflow --input-json=examples/hello/hello_shell_input.json
```

6. Check status

```
tibanna stat
```

7. Check output file

Let's try downloading the output file to check the content.

```
aws s3 cp s3://my-tibanna-test-bucket/some_sub_dirname/my_first_hello.txt
 ↪.
```

The output file `my_first_hello.txt` would look as below.

```
Hello world, Soo!
```

md5

We will preprare a pipeline that calculated md5sum. To create this pipeline and run it through Tibanna, we will do the following.

1. prepare for a component of the pipeline as a script (it could be a binary program)

2. package the components as a Docker image

3. create the pipeline description using either *CWL* or *WDL*.

4. prepare for a job definition that specifies pipeline, input files, parameters, resources, output target, etc.

5. run Tibanna.

Data

For input data, let's use a file named `somefastqfile.fastq.gz` on a public bucket named `my-tibanna-test-input-bucket`.

(You could also upload your own file to your own bucket and set up Tibanna to access that bucket.)

Pipeline component

Let's try a very simple pipeline that calculates the md5sum of an input file. We'll write a script named `run.sh` that calculates two md5sum values for a gzipped input file, one for the compressed and one for the uncompressed content of the file. The script creates an output file named `report` that contains two md5sum values. If the file is not gzipped, it simply repeats a regular md5sum value twice.

The pipeline/script could look like this:

```
#!/bin/bash

file=$1

if [[ $file =~ \.gz$ ]]
then
```

(continues on next page)

```
  MD_OUT=($(md5sum $file))
  CONTENT_MD_OUT=($(gunzip -c $file | md5sum))
else
  MD_OUT=($(md5sum $file))
  CONTENT_MD_OUT=$MD_OUT
fi

MD=${MD_OUT[0]}
CONTENT_MD=${CONTENT_MD_OUT[0]}
echo "$MD" >> report
echo "$CONTENT_MD" >> report
```

Docker image

We already have a public docker image for this (duplexa/md5:v2) that contains script run.sh. You can find it on Docker Hub: https://hub.docker.com/r/duplexa/md5/. If you want to use this public image, you can skip the following steps.

To create your own, first you need to install docker on your (local) machine.

1. First, create a directory (e.g. named md5)

2. Put the above run.sh script in this directory.

3. Then, inside this directory create a file named Dockerfile with the following content.

```
# start from ubuntu docker image
FROM ubuntu:20.04

# general updates & installing necessary Linux components
RUN apt-get update -y && apt-get install -y unzip

# copy the pipeline script into the image
# (in this case, /usr/local/bin)
WORKDIR /usr/local/bin
COPY run.sh .
RUN chmod +x run.sh

# default command
CMD ["run.sh"]
```

4. Then, build the docker image. You can use the same image name (duplexa/md5:v2) for this step, but it is recommended to replace duplexa with your preferred Docker Hub account name, to be able to push the image to Docker Hub later.

```
docker build -t my_account/md5:v2 .
```

5. Check the image

```
docker images
```

6. Push the image to Docker Hub. You will need an account on Docker Hub.

```
docker login
docker push my_account/md5:v2
```

Pipeline description

CWL

A sample CWL file is below. This CWL file can be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/md5/md5.cwl. To use your own docker image, replace `duplexa/md5:v2` with your docker image name. To use your own CWL file, you'll need to make sure it is accessible via HTTP so Tibanna can download it with `wget`: If you're using github, you could use raw.githubusercontent.com like the link above.

```
---
cwlVersion: v1.0
baseCommand:
  - run.sh
inputs:
  - id: "#gzfile"
    type:
      - File
    inputBinding:
      position: 1
outputs:
  - id: "#report"
    type:
    - File
    outputBinding:
      glob: report
hints:
  - dockerPull: duplexa/md5:v2
    class: DockerRequirement
class: CommandLineTool
```

The pipeline is ready!

WDL

Like CWL, WDL describes a pipeline structure. We describe individual runs (jobs) as separate json files.

A sample WDL file is below. This WDL file can be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/md5/md5.wdl. To use your own docker image, replace `duplexa/md5:v2` with your docker image name. To use your own WDL file, you'll need to make sure it is accessible via HTTP so Tibanna can download it with `wget`: If you're using github, you could use raw.githubusercontent.com like the link above. Content-wise, this WDL does exactly the same as the above CWL.

```
workflow md5 {
    call md5_step
}

task md5_step {
    File gzfile
    command {
        run.sh ${gzfile}
    }
    output {
        File report = "report"
```

(continues on next page)

```
    }
    runtime {
        docker: "duplexa/md5:v2"
    }
}
```

The pipeline is ready!

Shell

A list of shell commands can also be used. It could be something like this.

```
run.sh input.gz
```

A shell command doesn't have to be written in a file. The command itself can be passed to Tibanna as part of the job description json.

Job description

To run the pipeline on a specific input file using Tibanna, we need to create an *job description* file for each execution (or a dictionary object if you're using Tibanna as a python module).

Job description for CWL

The example job description for CWL is shown below and it can also be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/md5/md5_cwl_input.json.

```
{
  "args": {
    "app_name": "md5",
    "app_version": "v2",
    "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/tibanna/
→master/examples/md5",
    "cwl_main_filename": "md5.cwl",
    "cwl_child_filenames": [],
    "cwl_version": "v1",
    "input_files": {
      "gzfile": {
        "bucket_name": "my-tibanna-test-input-bucket",
        "object_key": "somefastqfile.fastq.gz"
      }
    },
    "secondary_files": {},
    "input_parameters": {},
    "output_S3_bucket": "my-tibanna-test-bucket",
    "output_target": {
      "report": "some_sub_dirname/my_first_md5_report"
    },
    "secondary_output_target": {}
  },
  "config": {
    "ebs_size": 10,
```

```
      "EBS_optimized": false,
      "instance_type": "t3.micro",
      "password": "whateverpasswordworks",
      "log_bucket": "my-tibanna-test-bucket"
    }
}
```

The json file specifies the input with `gzfile`, matching the name in CWL. In this example it is `somefastqfile.fastq.gz` on bucket `my-tibanna-test-input-bucket`. The output file will be renamed to `some_sub_dirname/my_first_md5_report` in a bucket named `my-tibanna-test-bucket`. In the input json, we specify the CWL file with `cwl_main_filename` and its url with `cwl_directory_url`. Note that the file name itself is not included in the url).

We also specified in `config`, that we need 10GB space total (`ebs_size`) and we're going to run an EC2 instance (VM) of type `t3.micro` which comes with 1 CPU and 1GB memory.

Job description for WDL

The example job description for WDL is shown below and it can also be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/md5/md5_wdl_input.json.

Content-wise, it is exactly the same as the one for CWL above. Notice that the only difference is that 1) you specify fields "wdl_main_filename", "wdl_child_filenames" and "wdl_directory_url" instead of "cwl_main_filename", "cwl_child_filenames", "cwl_directory_url", and "cwl_version" in `args`, that 2) you have to specify `"language" :  "wdl"` in `args` and that 3) when you refer to an input or an output, CWL allows you to use a global name (e.g. `gzfile`, `report`), whereas with WDL, you have to specify the workflow name and the step name (e.g. `md5.md5_step.gzfile`, `md5.md5_step.report`).

```
{
  "args": {
    "app_name": "md5",
    "app_version": "v2",
    "wdl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/tibanna/
↪master/examples/md5",
    "wdl_main_filename": "md5.wdl",
    "wdl_child_filenames": [],
    "language": "wdl",
    "input_files": {
      "md5.md5_step.gzfile": {
        "bucket_name": "my-tibanna-test-input-bucket",
        "object_key": "somefastqfile.fastq.gz"
      }
    },
    "secondary_files": {},
    "input_parameters": {},
    "output_S3_bucket": "my-tibanna-test-bucket",
    "output_target": {
      "md5.md5_step.report": "some_sub_dirname/my_first_md5_report"
    },
    "secondary_output_target": {}
  },
  "config": {
    "ebs_size": 10,
```

```
      "EBS_optimized": false,
      "instance_type": "t3.micro",
      "password": "whateverpasswordworks",
      "log_bucket": "my-tibanna-test-bucket"
   }
}
```

The json file specifies the input with `md5.md5_step.gzfile`, matching the name in WDL. In this example it is `somefastqfile.fastq.gz` on bucket `my-tibanna-test-input-bucket`. The output file will be renamed to `some_sub_dirname/my_first_md5_report` in a bucket named `my-tibanna-test-bucket`. In the input json, we specify the WDL file with `wdl_filename` and its url with `wdl_directory_url`. Note that the file name itself is not included in the url).

The config field is identical to the CWL input json. In `config`, we specify that we need 10GB space total (`ebs_size`) and we're going to run an EC2 instance (VM) of type `t3.micro` which comes with 1 CPU and 1GB memory.

Job description for shell

The job description for running shell commands requires `command` and `container_image` fields. The former is a list of commands and the latter is the Docker image name.

The current working directory for running shell commands is `/data1/shell` and it can be requested that input files be copied from S3 to this directory.

In the following example, input file `s3://my-tibanna-test-input-bucket/somefastqfile.fastq.gz` is copied to `/data1/shell` as `input.gz` which matches the input file in the `command` field (`run.sh input.gz`). The output file `report` in the same directory is copied to the output bucket `my-tibanna-test-bucket` as `some_sub_dirname/my_first_md5_report`.

This json file can be found at https://github.com/4dn-dcic/tibanna/blob/master/examples/md5/md5_shell_input.json

```
{
  "args": {
    "container_image": "duplexa/md5:v2",
    "command": ["run.sh input.gz"],
    "language": "shell",
    "input_files": {
      "file:///data1/shell/input.gz": "s3://my-tibanna-test-input-bucket/
↪somefastqfile.fastq.gz"
    },
    "secondary_files": {},
    "input_parameters": {},
    "output_S3_bucket": "my-tibanna-test-bucket",
    "output_target": {
      "file:///data1/shell/report": "some_sub_dirname/my_first_md5_report"
    },
    "secondary_output_target": {}
  },
  "config": {
    "ebs_size": 10,
    "instance_type": "t3.micro",
    "EBS_optimized": false,
```

```
      "password": "whateverpasswordworks",
      "log_bucket": "my-tibanna-test-bucket"
  }
}
```

Tibanna run

To run Tibanna,

1. Sign up for AWS

2. Install and configure awscli

   see Before_using_Tibanna

3. Install Tibanna on your local machine

   see Installation

4. Deploy Tibanna (link it to the AWS account)

   see Installation

5. Run workflow as below.

   For CWL,

   ```
   cd tibanna
   tibanna run_workflow --input-json=examples/md5/md5_cwl_input.json
   ```

   or for WDL,

   ```
   cd tibanna
   tibanna run_workflow --input-json=examples/md5/md5_wdl_input.json
   ```

   or for shell,

   ```
   cd tibanna
   tibanna run_workflow --input-json=examples/md5/md5_shell_input.json
   ```

6. Check status

```
tibanna stat
```

merge

This pipeline is an example of a nested input file array (e.g. [[f1, f2], [f3, f4]]). It consists of two steps,
paste and cat, the former pastes input files horrizontally and the latter concatenates input files vertically. Since
we're using generic commands, we do not need to create a pipeline software component or a Docker image. We will
use the existing ubuntu:20.04 Docker image. So, we will just do the following three steps.

1. create the pipeline description using either *CWL* or *WDL*.

2. prepare for a job definition that specifies pipeline, input files, parameters, resources, output target, etc.

3. run Tibanna.

Data

For input data, let's use files named `smallfile1`, `smallfile2`, `smallfile3` and `smallfile4` in a public bucket named `my-tibanna-test-input-bucket`. Each of these files contains a letter ('a', 'b', 'c', and 'd', respectively). We feed an array of array of these files in the following format:

```
[[smallfile1, smallfile2], [smallfile3, smallfile4]]
```

(You could also upload your own file to your own bucket and set up Tibanna to access that bucket.)

Pipeline description
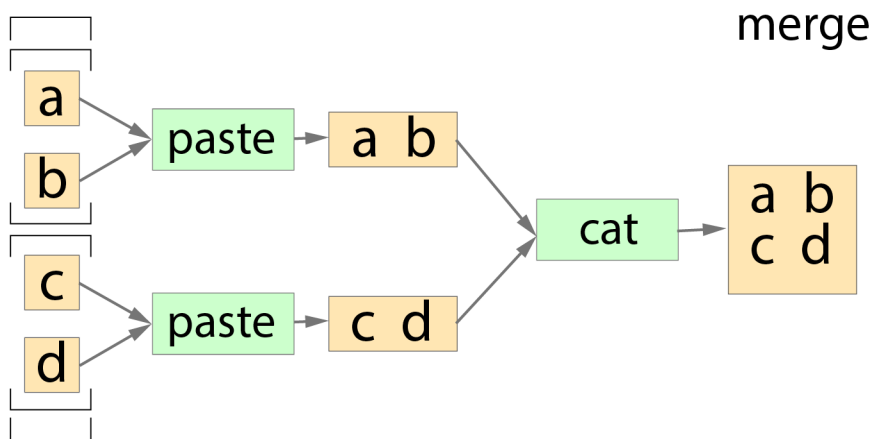
Thie pipeline takes an input 'smallfiles' which is an array of array of files. The input is scattered to the first step `paste`, which means that each element of 'smallfiles' (i.e. each array of files) goes as the input of `paste`, and the outputs will be gathered into an array before it is passed to the next step. From the input data above, there will be two runs of `paste` and one will take in `[smallfile1, smallfile2]` and the other `[smallfile3, smallfile4]`, and the outputs will be combined into an array *[<paste_output1>, <paste_output2>]*. The second step, `cat` takes in this array and concatenates them.

So, the output of the two `paste` runs would look like:

```
a   b
```

```
c   d
```

And the output of the `cat` (or the output of the workflow) would look like:

```
a   b
c   d
```

CWL

Since this is a multi-step pipeline, we use three CWL files, `merge.cwl` (master workflow CWL) and two other CWL files `paste.cwl` and `cat.cwl` that are called by `merge.cwl`.

These CWL files can be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge/merge.cwl, https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge/paste.cwl

and https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge/cat.cwl. To use your own CWL file, you'll need to make sure it is accessible via HTTP so Tibanna can download it with `wget`: If you're using github, you could use raw.githubusercontent.com like the link above. Alternatively, you can have tham as a local file and provide `cwl_directory_local` instead of `cwl_directory_url`.

The following is `merge.cwl`. It is of class 'workflow' and defines inputs, outputs and steps. For the other two CWL files (`paste.cwl` and `cat.cwl`), see the links above.

```
---
class: Workflow
cwlVersion: v1.0
inputs:
  smallfiles:
    type:
      type: array
      items:
        type: array
        items: File
outputs:
  -
    id: "#merged"
    type: File
    outputSource: "#cat/concatenated"
steps:
  -
    id: "#paste"
    run: "paste.cwl"
    in:
    -
      id: "#paste/files"
      source: "smallfiles"
    scatter: "#paste/files"
    out:
    -
      id: "#paste/pasted"
  -
    id: "#cat"
    run: "cat.cwl"
    in:
    -
      id: "#cat/files"
      source: "#paste/pasted"
    out:
    -
      id: "#cat/concatenated"
requirements:
  -
    class: "ScatterFeatureRequirement"
```

The pipeline is ready!

WDL

WDL describes this pipeline in one file and it can be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge/merge.wdl. To use your own WDL file, you'll need to make sure it is accessible via HTTP so Tibanna can download it with `wget`: If you're using github, you could use

raw.githubusercontent.com like the link above. Content-wise, this WDL does exactly the same as the above CWL.

```
workflow merge {
    Array[Array[File]] smallfiles = []
    scatter(smallfiles_ in smallfiles) {
        call paste {input: files = smallfiles_}
    }
    call cat {input: files = paste.pasted}
    output {
        File merged = cat.concatenated
    }
}

task paste {
    Array[File] files = []
    command {
        paste ${sep=" " files} > pasted
    }
    output {
        File pasted = "pasted"
    }
    runtime {
        docker: "ubuntu:20.04"
    }
}

task cat {
    Array[File] files = []
    command {
        cat ${sep=" " files} > concatenated
    }
    output {
        File concatenated = "concatenated"
    }
    runtime {
        docker: "ubuntu:20.04"
    }
}
```

The pipeline is ready!

Job description

To run the pipeline on a specific input file using Tibanna, we need to create an *job description* file for each execution (or a dictionary object if you're using Tibanna as a python module).

Job description for CWL

The example job description for CWL is shown below and it can also be found at https://raw.githubusercontent.com/4dn-dic/tibanna/master/examples/merge/merge_cwl_input.json.

```
{
  "args": {
    "app_name": "merge",
```

```
    "app_version": "",
    "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/tibanna/
↪master/examples/merge",
    "cwl_main_filename": "merge.cwl",
    "cwl_child_filenames": ["paste.cwl", "cat.cwl"],
    "cwl_version": "v1",
    "input_files": {
      "smallfiles": {
        "bucket_name": "my-tibanna-test-input-bucket",
        "object_key": [["smallfile1", "smallfile2"], ["smallfile3",
↪"smallfile4"]]
      }
    },
    "secondary_files": {},
    "input_parameters": {},
    "output_S3_bucket": "my-tibanna-test-bucket",
    "output_target": {
      "merged": "some_sub_dirname/my_first_merged_file"
    },
    "secondary_output_target": {}
  },
  "config": {
    "ebs_size": 10,
    "EBS_optimized": true,
    "instance_type": "t3.micro",
    "password": "whateverpasswordworks",
    "log_bucket": "my-tibanna-test-bucket"
  }
}
```

The json file specifies the input nested file array ("smallfiles") (`[["smallfile1"`, `"smallfile2"]`, `["smallfile3"`, `"smallfile4"]]`), matching the name in CWL. The output file will be renamed to `some_sub_dirname/my_first_merged_file` in a bucket named `my-tibanna-test-bucket`. In the input json, we specify the CWL file with `cwl_main_filename` and its url with `cwl_directory_url`. Note that the file name itself is not included in the url). Note that child CWL files are also specified in this case (`"cwl_child_filenames":  ["paste.cwl", "cat.cwl"]`).

We also specified in `config`, that we need 10GB space total (`ebs_size`) and we're going to run an EC2 instance (VM) of type `t3.micro` which comes with 1 CPU and 1GB memory.

Job description for WDL

The example job description for WDL is shown below and it can also be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge/merge_wdl_input.json.

Content-wise, it is exactly the same as the one for CWL above. Notice that the only difference is that 1) you specify fields "wdl_main_filename", "wdl_child_filenames" and "wdl_directory_url" instead of "cwl_main_filename", "cwl_child_filenames", "cwl_directory_url", and "cwl_version" in `args`, that 2) you have to specify `"language" :  "wdl"` in `args` and that 3) when you refer to an input or an output, CWL allows you to use a global name (e.g. `smallfiles`, `merged`), whereas with WDL, you have to specify the workflow name (e.g. `merge.smallfiles`, `merge.merged`). We omit the step names in this case because we use global variables that are passed to and from the steps.

```
{
  "args": {
    "app_name": "merge",
    "app_version": "",
    "language": "wdl",
    "wdl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/tibanna/
→master/examples/merge",
    "wdl_main_filename": "merge.wdl",
    "wdl_child_filenames": [],
    "input_files": {
      "merge.smallfiles": {
        "bucket_name": "my-tibanna-test-input-bucket",
        "object_key": [["smallfile1", "smallfile2"], ["smallfile3",
→"smallfile4"]]
      }
    },
    "secondary_files": {},
    "input_parameters": {},
    "output_S3_bucket": "my-tibanna-test-bucket",
    "output_target": {
      "merge.merged": "some_sub_dirname/my_first_merged_file"
    },
    "secondary_output_target": {}
  },
  "config": {
    "ebs_size": 10,
    "EBS_optimized": true,
    "instance_type": "t3.micro",
    "password": "whateverpasswordworks",
    "log_bucket": "my-tibanna-test-bucket"
  }
}
```

Tibanna run

To run Tibanna,

1. Sign up for AWS

2. Install and configure `awscli`

   see Before_using_Tibanna

3. Install Tibanna on your local machine

   see Installation

4. Deploy Tibanna (link it to the AWS account)

   see Installation

5. Run workflow as below.

   For CWL,

   ```
   cd tibanna
   tibanna run_workflow --input-json=examples/merge/merge_cwl_input.json
   ```

   or for WDL,

```
cd tibanna
tibanna run_workflow --input-json=examples/merge/merge_wdl_input.json
```

6. Check status

```
tibanna stat
```

merge_and_cut

This pipeline is an example of a double-nested input file array (e.g. `[[[f1, f2], [f3, f4]], [[f5, f6], [f7, f8]]]`). It consists of a subworkflow called `merge` (previous section) and an extra step called `cut`. Merge consists of two steps, `paste` and `cat`, the former pastes input files horrizontally and the latter concatenates input files vertically. `Cut` prints the first letter of every line from a list of files. Since we're using generic commands, we do not need to create a pipeline software component or a Docker image. We will use the existing `ubuntu:20.04` Docker image. So, we will just do the following three steps.

1. create the pipeline description using either *CWL* or *WDL*.

2. prepare for a job definition that specifies pipeline, input files, parameters, resources, output target, etc.

3. run Tibanna.

Data

For input data, let's use files named `smallfile1, smallfile2, ... smallfile8` in a public bucket named `my-tibanna-test-input-bucket`. Each of these files contains a letter ('a', 'b', ... , 'h', respectively). We feed an array of array of array of these files in the following format:

```
[
  [[smallfile1, smallfile2], [smallfile3, smallfile4]],
  [[smallfile5, smallfile6], [smallfile7, smallfile8]]
]
```

(You could also upload your own file to your own bucket and set up Tibanna to access that bucket.)

Pipeline description
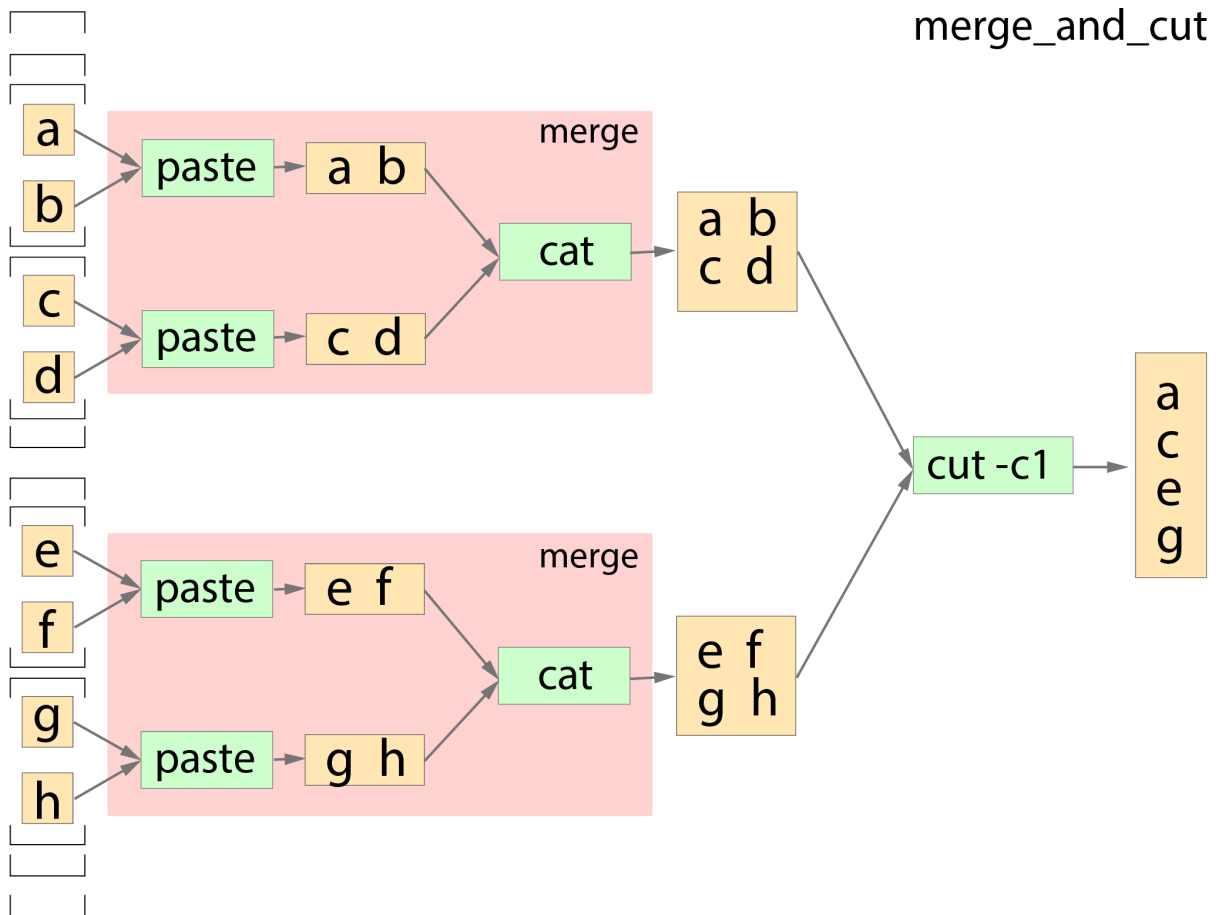
Thie pipeline takes an input 'smallfiles' which is an array of array of array of files. The 3D input is scattered into 2D arrays to the first subworkflow `merge`, which internally scatters each of the 2D arrays into 1D at the step `paste` and put it through another step `cat` which produces a single file. Therefore, each 2D array has an output of a single file. The result of these outputs are combined into an array and fed to the extra step `cut`.

CWL

The structure of this pipeline is a bit complex and we use five CWL files, three of which are identical to the ones from `merge` workflow example (`merge.cwl`, `paste.cwl`, `cat.cwl`) since we're using the `merge` workflow as a subworkflow. There is one master CWL (`merge_and_cut.cwl`) and an extra step CWL, `cut.cwl`.

These CWL files can be found at the following URLs:

- https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge_and_cut/merge_and_cut.cwl
- https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge_and_cut/merge.cwl
- https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge_and_cut/paste.cwl
- https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge_and_cut/cat.cwl

- https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge_and_cut/cut.cwl

To use your own CWL file, you'll need to make sure it is accessible via HTTP so Tibanna can download it with `wget`: If you're using github, you could use raw.githubusercontent.com like the link above.

The following is `merge_and_cut.cwl`.

```
---
class: Workflow
cwlVersion: v1.0
inputs:
  smallfiles:
    type:
      type: array
      items:
        type: array
        items:
          type: array
          items: File
outputs:
  -
    id: "#merged_and_cut"
    type: File
    outputSource: "#cut/cut1"
steps:
  -
    id: "#merge"
    run: "merge.cwl"
    in:
      -
        id: "#merge/smallfiles"
        source: "smallfiles"
    scatter: "#merge/smallfiles"
    out:
      -
        id: "#merge/merged"
  -
    id: "#cut"
    run: "cut.cwl"
    in:
      -
        id: "#cut/files"
        source: "#merge/merged"
    out:
      -
        id: "#cut/cut1"
requirements:
  -
    class: "ScatterFeatureRequirement"
  -
    class: "SubworkflowFeatureRequirement"
```

The pipeline is ready!

WDL

WDL describes this pipeline in two files, one main file and a subworkflow file. The main file can be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge_and_cut/merge_

and_cut.wdl and the subworkflow file is identical to the WDL file used in the example of **merge** (https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge_and_cut/merge.wdl). To use your own WDL file, you'll need to make sure it is accessible via HTTP so Tibanna can download it with `wget`: If you're using github, you could use raw.githubusercontent.com like the link above. Content-wise, this WDL does exactly the same as the above CWL. Below is the main WDL.

```
import "merge.wdl" as sub

workflow merge_and_cut {
    Array[Array[Array[File]]] smallfiles = []
    scatter(smallfiles_ in smallfiles) {
        call sub.merge {input: smallfiles = smallfiles_}
    }
    call cut {input: files = merge.merged}
    output {
        File merged_and_cut = cut.cut1
    }
}

task cut {
    Array[File] files = []
    command {
        cut -c1 ${sep=" " files} > cut1
    }
    output {
        File cut1 = "cut1"
    }
    runtime {
        docker: "ubuntu:20.04"
    }
}
```

The pipeline is ready!

Job description

To run the pipeline on a specific input file using Tibanna, we need to create an *job description* file for each execution (or a dictionary object if you're using Tibanna as a python module).

Job description for CWL

The example job description for CWL is shown below and it can also be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge_and_cut/merge_and_cut_cwl_input.json.

```
{
  "args": {
    "app_name": "merge_and_cut",
    "app_version": "",
    "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/tibanna/
↪master/examples/merge_and_cut",
    "cwl_main_filename": "merge_and_cut.cwl",
    "cwl_child_filenames": ["merge.cwl", "paste.cwl", "cat.cwl", "cut.cwl"],
    "cwl_version": "v1",
```

(continues on next page)

```
    "input_files": {
      "smallfiles": {
        "bucket_name": "my-tibanna-test-input-bucket",
        "object_key": [
            [["smallfile1", "smallfile2"], ["smallfile3", "smallfile4"]],
            [["smallfile5", "smallfile6"], ["smallfile7", "smallfile8"]]
        ]
      }
    },
    "secondary_files": {},
    "input_parameters": {},
    "output_S3_bucket": "my-tibanna-test-bucket",
    "output_target": {
      "merged_and_cut": "some_sub_dirname/my_first_merged_and_cut_file"
    },
    "secondary_output_target": {}
  },
  "config": {
    "ebs_size": 10,
    "EBS_optimized": true,
    "instance_type": "t3.micro",
    "password": "whateverpasswordworks",
    "log_bucket": "my-tibanna-test-bucket"
  }
}
```

The json file specifies the input double-nested file array ("smallfiles"), matching the name in CWL. The output file will be renamed to `some_sub_dirname/my_first_merged_and_cut_file` in a bucket named `my-tibanna-test-bucket`. In the input json, we specify the CWL file with `cwl_main_filename` and its url with `cwl_directory_url`. Note that the file name itself is not included in the url). Note that child CWL files are also specified in this case (`"cwl_child_filenames": ["merge.cwl", "paste.cwl", "cat.cwl", "cut.cwl"]`).

We also specified in `config`, that we need 10GB space total (`ebs_size`) and we're going to run an EC2 instance (VM) of type `t3.micro` which comes with 1 CPU and 1GB memory.

Job description for WDL

The example job description for WDL is shown below and it can also be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/merge_and_cut/merge_and_cut_wdl_input.json.

Content-wise, it is exactly the same as the one for CWL above. Notice that the only difference is that 1) you specify fields "wdl_main_filename", "wdl_child_filenames" and "wdl_directory_url" instead of "cwl_main_filename", "cwl_child_filenames", "cwl_directory_url", and "cwl_version" in `args`, that 2) you have to specify `"language" : "wdl"` in `args` and that 3) when you refer to an input or an output, CWL allows you to use a global name (e.g. `smallfiles`, `merged`), whereas with WDL, you have to specify the workflow name (e.g. `merge_and_cut.smallfiles`, `merge_and_cut.merged_and_cut`). We omit the step names in this case because we use global variables that are passed to and from the steps.

```
{
  "args": {
```

```
      "app_name": "merge_and_cut",
      "app_version": "",
      "language": "wdl",
      "wdl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/tibanna/
↪master/examples/merge_and_cut",
      "wdl_main_filename": "merge_and_cut.wdl",
      "wdl_child_filenames": ["merge.wdl"],
      "input_files": {
        "merge_and_cut.smallfiles": {
          "bucket_name": "my-tibanna-test-input-bucket",
          "object_key": [
              [["smallfile1", "smallfile2"], ["smallfile3", "smallfile4"]],
              [["smallfile5", "smallfile6"], ["smallfile7", "smallfile8"]]
          ]
        }
      },
      "secondary_files": {},
      "input_parameters": {},
      "output_S3_bucket": "my-tibanna-test-bucket",
      "output_target": {
        "merge_and_cut.merged_and_cut": "some_sub_dirname/my_first_merged_and_
↪cut_file"
      },
      "secondary_output_target": {}
    },
  "config": {
    "ebs_size": 10,
    "EBS_optimized": true,
    "instance_type": "t3.micro",
    "password": "whateverpasswordworks",
    "log_bucket": "my-tibanna-test-bucket"
  }
}
```

Tibanna run

To run Tibanna,

1. Sign up for AWS

2. Install and configure `awscli`

   see Before_using_Tibanna

3. Install Tibanna on your local machine

   see Installation

4. Deploy Tibanna (link it to the AWS account)

   see Installation

5. Run workflow as below.

   For CWL,

   ```
   cd tibanna
   tibanna run_workflow --input-json=examples/merge_and_cut/merge_and_cut_
   ↪cwl_input.json
   ```

or for WDL,

```
cd tibanna
tibanna run_workflow --input-json=examples/merge_and_cut/merge_and_cut_
→wdl_input.json
```

6. Check status

```
tibanna stat
```

cond_merge

This pipeline is an example of a conditional output. It chooses between two tasks, `paste` and `cat`, depending on the length of the input array (i.e. the number of input files). The former pastes input files horrizontally and the latter concatenates input files vertically. Since we're using generic commands, we do not need to create a pipeline software component or a Docker image. We will use the existing `ubuntu:20.04` Docker image. So, we will just do the following three steps.

1. create the pipeline description using *WDL*. (*CWL* does not support conditional statements)

2. prepare for a job definition that specifies pipeline, input files, parameters, resources, output target, etc.

3. run Tibanna.

Data

For input data, let's use files named `smallfile1`, `smallfile2`, `smallfile3` and `smallfile4` in a public bucket named `my-tibanna-test-input-bucket`. Each of these files contains a letter ('a', 'b', 'c', and 'd', respectively). We feed an array of these files in the following formats (one with length 4, another with length 2):

```
[smallfile1, smallfile2, smallfile3, smallfile4]
```

```
[smallfile1, smallfile2]
```

(You could also upload your own file to your own bucket and set up Tibanna to access that bucket.)

Pipeline description

This pipeline takes an input file array. If the length of the array is larger than 2 (more than 2 files), it runs `paste`. If it is smaller than or equal to 2, it runs `cat`. The former creates a pasted file and the latter creates a concatenated file.

In the former case (`paste`), the output would look like this:

```
a   b   c   d
```

In the latter case (`cat`), the output would look like:

```
a
b
```

WDL

WDL describes this pipeline in one file and it can be found at https://raw.githubusercontent.com/4dn-dcic/tibanna/master/examples/cond_merge/cond_merge.wdl. To use your own WDL file, you'll need to make sure it is accessible via HTTP so Tibanna can download it with `wget`: If you're using github, you could use raw.githubusercontent.com like the link above. Content-wise, this WDL does exactly the same as the above CWL.

```
workflow cond_merge {
    Array[File] smallfiles = []
    if(length(smallfiles)>2) {
        call paste {input: files = smallfiles}
    }
    if(length(smallfiles)<=2) {
        call cat {input: files = smallfiles}
    }
}

task paste {
    Array[File] files = []
    command {
        paste ${sep=" " files} > pasted
    }
    output {
        File pasted = "pasted"
    }
    runtime {
        docker: "ubuntu:20.04"
    }
}

task cat {
    Array[File] files = []
    command {
        cat ${sep=" " files} > concatenated
    }
    output {
        File concatenated = "concatenated"
    }
    runtime {
        docker: "ubuntu:20.04"
    }
}
```

The pipeline is ready!

Job description

To run the pipeline on a specific input file using Tibanna, we need to create an *job description* file for each execution (or a dictionary object if you're using Tibanna as a python module).

Job description for WDL

If the user does not know (or does not want to manually control) which of the two outputs should be sent to S3, one can specify a global name for this output and associate it with the alternative output

---

names. For example, in this case, we could set up a global name to be `cond_merge.cond_merged`
and associate with two alternative names `cond_merge.paste.pasted` and `cond_merge.cat.`
`concatenated`. This way, either of the two will be recognized as `cond_merge.cond_merged`
and will be treated as if it was not a conditional output from the user's perspective.

An example job description for WDL is shown below and it can also be found at https://raw.
githubusercontent.com/4dn-dcic/tibanna/master/examples/cond_merge/cond_merge_wdl_input.json.
Another example with two input files can be found at https://raw.githubusercontent.com/
4dn-dcic/tibanna/master/examples/cond_merge/cond_merge_wdl_input2.json. Note the field
`alt_cond_output_argnames` under `args`.

```
{
  "args": {
    "app_name": "cond_merge",
    "app_version": "",
    "language": "wdl",
    "wdl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/tibanna/
↪master/examples/cond_merge",
    "wdl_main_filename": "cond_merge.wdl",
    "wdl_child_filenames": [],
    "input_files": {
      "cond_merge.smallfiles": {
        "bucket_name": "my-tibanna-test-input-bucket",
        "object_key": ["smallfile1", "smallfile2", "smallfile3", "smallfile4
↪"]
      }
    },
    "secondary_files": {},
    "input_parameters": {},
    "output_S3_bucket": "my-tibanna-test-bucket",
    "output_target": {
        "cond_merge.cond_merged": "some_sub_dirname/my_first_cond_merged_file
↪"
    },
    "alt_cond_output_argnames": {
        "cond_merge.cond_merged": ["cond_merge.paste.pasted", "cond_merge.
↪cat.concatenated"]
    },
    "secondary_output_target": {}
  },
  "config": {
    "ebs_size": 10,
    "EBS_optimized": true,
    "instance_type": "t3.micro",
    "password": "whateverpasswordworks",
    "log_bucket": "my-tibanna-test-bucket"
  }
}
```

Tibanna run

To run Tibanna,

1. Sign up for AWS

2. Install and configure `awscli`

   see Before_using_Tibanna

3. Install Tibanna on your local machine

   see Installation

4. Deploy Tibanna (link it to the AWS account)

   see Installation

5. Run workflow as below.

```
cd tibanna
tibanna run_workflow --input-json=examples/cond_merge/cond_merge_wdl_
→input.json
tibanna run_workflow --input-json=examples/cond_merge/cond_merge_wdl_
→input2.json
```

6. Check status

```
tibanna stat
```

1.4.3 Check Before using Tibanna

- Before using Tibanna, one must have an **AWS account**.

- An **admin** user with access key and secret key **sets up and deploys Tibanna** for a specific user group and specific buckets.

- A **regular user**, with their own access key and secret key, associated with the user group can upload data to the bucket and **run jobs using Tibanna**.

- In addition, your *workflows* must be written in either *CWL (Common Workflow Language)* or *WDL (Workflow Description Language)* which point to a docker image on *docker hub* or AWS ECR (Elastic Container Registry) on the same AWS account. Alternatively, you can use *Snakemake* workflow to be run as a whole on a single EC2 machine, inside a Snakemake docker image. A CWL/WDL/Snakemake file can be a public url, a local file, or on a (public or private) S3 bucket.

1.4.4 Installation

Tibanna's installation is two-step - installation of the tibanna package on the local machine and deployment of its serverless components to the AWS Cloud. Since the second step is separated from the first step, one may deploy as many copies of Tibanna as one wishes for different projects, with different bucket permissions and users.

Installing Tibanna package

Tibanna works with the following Python and pip versions.

- Python 3.8, 3.9, 3.10

- Pip 9, 10, 18, 19, 20

Install Tibanna on your local machine or server from which you want to send commands to run workflows.

First, create a virtual environment.

```
# create a virtual environment
virtualenv -p python3.8 ~/venv/tibanna
source ~/venv/tibanna/bin/activate
```

Then, install Tibanna.

```
pip install tibanna
```

Alternatively, use `git clone` followed by `make install`

```
# Alternatively installing tibanna package from github repo
 git clone https://github.com/4dn-dcic/tibanna
 cd tibanna
 make install
```

Starting version `1.0.0`, there is also a Docker image that contains the same version of tibanna as the image tag. This image is used on the EC2 AWSEM instances and not for a local use. The image contains many other things including Docker, Singularity, Cromwell, cwltool, etc. in addition to Tibanna and therefore not recommended, but in case the above two somehow didn't work in your environment, and if you have Docker, you could try:

```
docker run -it 4dndcic/tibanna-awsf:1.0.0 bash
# You could use a different version tag instead of 1.0.0
# you can also mount your local directories and files as needed.
```

AWS configuration

To deploy and use Tibanna on the AWS Cloud, you must first have an AWS account.

Deployment requires an admin user credentials. For more details, check out https://aws.amazon.com/.

To only run workflows using Tibanna, you need a regular user credentials.

Once you have the user credentials, we can add that information to the local machine using one of the following three methods:

1) using `awscli`

2) by manually creating two files in `~/.aws`.

3) setting AWS environment variables

Details of each method is described below. Tibanna uses this information to know that you have the permission to deploy to your AWS account.

1) using `awscli`

```
# first install awscli - see below if this fails
pip install awscli

# configure AWS credentials and config through awscli
aws configure
```

Type in your keys, region and output format ('json') as below.

```
AWS Access Key ID [None]: <your_aws_key>
AWS Secret Access Key [None]: <your_aws_secret_key>
Default region name [None]: us-east-1
Default output format [None]: json
```

2) by manually creating two files in `~/.aws`

Alternatively, (in case you can't install `awscli` for any reason (e.g. `PyYAML` version conflict)), do the following manually to set up AWS credentials and config.

```
mkdir ~/.aws
```

Add the following to `~/.aws/credentials`.

```
[default]
aws_access_key_id = <your_aws_key>
aws_secret_access_key = <your_aws_secret_key>
```

Add the following to `~/.aws/config`.

```
[default]
region = us-east-1
output = json
```

3) setting AWS environment variables

Alternatively, you can directly set AWS credentials and config as environment variables (instead of creating `~/.aws/credentials` and `~/.aws/config`).

```
export AWS_ACCESS_KEY_ID=<AWS_ACCESS_KEY>
export AWS_SECRET_ACCESS_KEY=<AWS_SECRET_ACCESS_KEY>
export AWS_DEFAULT_REGION=<AWS_DEFAULT_REGION>
```

Tibanna environment variables

Note: starting `0.9.0`, users do not need to export `AWS_ACCOUNT_NUMBER` and `TIBANNA_AWS_REGION` any more.

Deploying Tibanna Unicorn to AWS

Note: You have to have admin permission to deploy unicorn to AWS and add user to a tibanna permission group

If you're using a forked Tibanna repo or want to use a specific branch, set the following variables as well before deployment. They will be used by the EC2 (VM) instances to grab the right scripts from the *awsf* directory of the right tibanna repo/branch. If you're using default (`4dn-dcic/tibanna`, `master`), no need to set these variables.

```
# only if you're using a forked repo
export TIBANNA_REPO_NAME=<git_hub_repo_name>  # (default: 4dn-dcic/tibanna)
export TIBANNA_REPO_BRANCH=<git_hub_branch_name>  # (default: master)
```

If you're using an external bucket with a separate credential, you can give the permission to this bucket to tibanna unicorn during deployment by setting the following additional environment variables before deploying. This credential will be added as profile `user1` on the EC2 instances to run. This profile name can be added to input file specifications for the files that require this external credential. For most cases, this part can be ignored.

```
# only if you're using an external bucket with a separate credential
export TIBANNA_PROFILE_ACCESS_KEY=<external_profile_access_key>
export TIBANNA_PROFILE_SECRET_KEY=<external_profile_secret_key>
```

Then, deploy a copy of Tibanna as below.

If you want to operate multiple copies of Tibanna (e.g. for different projects), you can try to name each copy of Tibanna using `--usergroup` option (by default the name is `default_<random_number>`).

Here, we're naming it `hahaha` - come up with a better name if you want to.

```
tibanna deploy_unicorn --usergroup=hahaha
# This will give permission to only public tibanna test buckets.
# To add permission to other private or public buckets, use --buckets option.
```

Run a test workflow

The above command will first create a usergroup that shares the permission to use a single tibanna environment. Then, it will deploy a tibanna instance (step function / lambda). The name of the tibanna step function is added to your `~/.bashrc` file. Check that you can see the following line in the `~/.bashrc` file.

```
# check your ~/.bashrc file
tail -1 ~/.bashrc
```

You should be able to see the following.

```
export TIBANNA_DEFAULT_STEP_FUNCTION_NAME=tibanna_unicorn_hahaha
```

To set this environmental variable,

```
source ~/.bashrc
```

You can run a workflow using Tibanna if you're an admin user or if you are a user that belongs to the user group. The following command launches a workflow run. See below for what to feed as input json, which contains information about what buckets to use, where to find the workflow CWL/WDL or what command to run inside a docker container, what the output file names should be, etc.

```
tibanna run_workflow --input-json=<input_json_for_a_workflow_run>
```

As an example you can try to run a test workflow as below. This one uses only public buckets `my-tibanna-test-bucket` and `my-tibanna-test-input-bucket`. The public has permission to these buckets - the objects will expire in 1 day and others may have access to the same bucket and read/overwrite/delete your objects. Please use it only for initial testing of Tibanna.

First, create the input json file `my_test_tibanna_input.json` as below.

```
{
  "args": {
    "app_name": "md5",
    "app_version": "0.2.6",
    "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/pipelines-cwl/0.
→2.6/cwl_awsem_v1/",
    "cwl_main_filename": "md5.cwl",
    "cwl_version": "v1",
    "input_files": {
      "input_file": {
        "bucket_name": "my-tibanna-test-input-bucket",
        "object_key": "somefastqfile.fastq.gz"
      }
    },
    "output_S3_bucket": "my-tibanna-test-bucket",
    "output_target": {
      "report": "my_outdir/report"
    }
  },
  "config": {
```

(continues on next page)

```
    "run_name": "md5-public-test",
    "log_bucket": "my-tibanna-test-bucket"
  }
}
```

```
tibanna run_workflow --input-json=my_test_tibanna_input.json
```

Deploying Tibanna Unicorn with private buckets

Note: You have to have admin permission to deploy unicorn to AWS and add user to a tibanna permission group

Creating a bucket

You can skip this section if you want to use existing buckets for input/output/logs.

If you are an admin or have a permission to create a bucket, you can either use the AWS Web Console or use the following command using *awscli*. For example, a data (input/output) bucket and a tibanna log bucket may be created. You could also separate input and output buckets, or have multiple input buckets, etc. Bucket names are globally unique.

```
aws s3api create-bucket --bucket <bucketname>
```

Example

```
aws s3api create-bucket --bucket montys-data-bucket   # choose your own data bucket␣
↪name
aws s3api create-bucket --bucket montys-tibanna-log-bucket   # choose your own log␣
↪bucket name
```

Upload your files to the data bucket by using the following

```
aws s3 cp <filename> s3://<bucketname>/<filename>
aws s3 cp --recursive <dirname> s3://<bucketname>/<dirname>
```

Example

```
aws s3 cp somebamfile.bam s3://montys-data-bucket/somebamfile.bam
aws s3 cp --recursive montys-input-data-folder s3://montys-data-bucket/montys-input-
↪data-folder
```

Deploying Tibanna

Let's try setting up Tibanna that uses private buckets. As you deploy your tibanna, add your private bucket names. Again, you can name this new copy of Tibanna by specifying a new user group (e.g. `lalala`.)

```
tibanna deploy_unicorn --buckets=<bucket1>,<bucket2>,... --usergroup=lalala
```

Example

```
tibanna deploy_unicorn --buckets=montys-data-bucket,montys-tibanna-log-bucket \
                    --usergroup=lalala

# no space between bucket names!
```

Export the environmental variable for Tibanna step function name.

```
source ~/.bashrc
```

Create an input json using your buckets.

Then, run workflow.

```
tibanna run_workflow --input-json=<input_json>
```

Now we have two different copies of deployed Tibanna. According to your *~/.bashrc*, the latest deployed copy is your default copy. However, if you want to run a workflow on a different copy of Tibanna, use `--sfn` option. For example, now your default copy is `lalala` (the latest one), but you want to run our workflow on `hahaha`. Then, do the following.

```
tibanna run_workflow --input-json=<input_json> --sfn=tibanna_unicorn_hahaha
```

User permission

To deploy Tibanna, one must be an admin for an AWS account. To run a workflow, the user must be either an admin or in the IAM group `tibanna_<usergroup>`. To add a user to a user group, you have to be an admin. To do this, use the `tibanna` command.

```
tibanna users
```

You will see the list of users.

Example

```
user        tibanna_usergroup
soo
monty
```

The following command will add a user to a specific user group.

```
tibanna add_user --user=<user> --usergroup=<usergroup>
```

For example, if you have a user named `monty` and you want to give permission to this user to user Tibanna `lalala`. This will give this user permission to run and monitor the workflow, access the buckets that Tibanna usergroup `lalala` was given access to through `tibanna deploy_unicorn --buckets=<b1>,<b2>,...`

```
tibanna add_user --user=monty --usergroup=lalala
```

Check users again.

```
tibanna users
```

```
user        tibanna_usergroup
soo
monty       lalala
```

Now `monty` can use `tibanna_unicorn_lalala` and access buckets `montys-data-bucket` and `montys-tibanna-log-bucket`

1.4.5 Command-line tools

Listing all commands

To list all available commands, use `tibanna -h`

```
tibanna -h
```

Checking Tibanna version

To check Tibanna version,

```
tibanna -v
```

Admin only commands

The following commands require admin previlege to one's AWS account.

deploy_unicorn

To create an instance of tibanna unicorn (step function + lambdas)

```
tibanna deploy_unicorn [<options>]
```

Options

`-b|--buckets=<bucket1,bucket2,...>`	List of buckets to use **for** tibanna runs. The associated **lambda** functions, EC2 instances **and** user group will be given permission to these buckets.
`-S|--no-setup`	Skip setup buckets/permissions **and** just redeploy tibanna step function **and** lambdas. This **is** useful when upgrading the existing tibanna that's already set up.
`-E|--no-setenv` ↪FUNCTION_NAME	Do **not** overwrite TIBANNA_DEFAULT_STEP_ environmental variable **in** your bashrc.
`-s|--suffix=<suffixname>` ↪version	Using suffix helps deploying various dev- tibanna. The step function **and lambda**
↪functions	will have the suffix. Having a different
↪suffix	does **not** create a new user group **with** a
↪different	permission (**for** this purpose use --
↪usergroup).	

(continues on next page)

(continued from previous page)

```
-g|--usergroup=<usergroup>            Tibanna usergroup to share the permission to␣
↪access
                                      buckets and run jobs

-P|--do-not-delete-public-access-block  Do not delete public access block from␣
↪buckets
                                      (this way postrunjson and metrics reports␣
↪will
                                      not be public)
```

-C|–deploy-costupdater Deploys an additional step function that will periodically check, if the cost for a workflow run can be retrieved from AWS. If it is available, it will automatically update the metrics report.

Note: starting `0.9.0`, users do not need to export `AWS_ACCOUNT_NUMBER` and `TIBANNA_AWS_REGION` any more.

deploy_core

Deploy/update only a single lambda function

```
tibanna deploy_core -n <lambda_name> [<options>]
```

where `<lambda_name>` would be either `run_task_awsem` or *check_task_awsem'*.

Options

```
-s|--suffix=<suffixname>              Using suffix helps deploying various dev-version␣
↪tibanna.
                                      The step function and lambda functions will have␣
↪the suffix.

-g|--usergroup=<usergroup>            Tibanna usergroup to share the permission to␣
↪access
                                      buckets and run jobs
```

users

To list users

```
tibanna users
```

add_user

To add users to a tibanna user group

```
tibanna add_user -u <user> -g <usergroup>
```

cleanup

To remove Tibanna components on AWS.

```
tibanna cleanup -g <usergroup> ...
```

Options

```
-s|--suffix=<suffixname>              If suffix was used to deploy a tibanna, it should␣
↪be added
                                      here. The step function and lambda functions will␣
↪have the
                                      suffix at the end.

-E|--do-not-ignore-errors             By default, if any of the components does not␣
↪exist (e.g.
                                      already removed), it does not throw an error and␣
↪keeps on
                                      to remove the other components. Using this option␣
↪turns off
                                      this feature and will throw an error.

-G|--do-not-remove-iam-group          if set, it does not remove the IAM permissions.␣
↪This option
                                      is recommended if various suffices are used to␣
↪share the
                                      same usergroup.

-p|--purge-history                    if set, remove all the job logs and other job-
↪related files
                                      from S3 bucket and dynamoDB. Please use with␣
↪caution.

-q|--quiet                            run quietly
```

setup_tibanna_env

- Advanced user only

To set up environment on AWS without deploying tibanna, use *tibanna setup_tibanna_env*.

```
tibanna setup_tibanna_env <options>
```

Options

```
-g|--usergroup-tag=<usergrouptag>      an identifier for a usergroup that shares
                                       a tibanna permission

-R|--no-randomize                      do not add a random number to generate a
                                       usergroup name (e.g. the usergroup name used
                                       will be identical to the one specified using
                                       the ``--usergrou-tag`` option.  By default,
                                       a random number will be added at the end
                                       (e.g. default_2721).

-b|--buckets=<bucket_list>             A comma-delimited list of bucket names - the
                                       buckets to which Tibanna needs access to
                                       through IAM role (input, output, log).

-P|--do-not-delete-public-access-block  Do not delete public access block from␣
↪buckets
```

```
                                    (this way postrunjson and metrics reports␣
↪will

                                    not be public)
```

Non-admin user commands

The following commands can be used by a non-admin user, as long as the user belongs to the right user group.

run_workflow

To run workflow

```
tibanna run_workflow --input-json=<input_json_file> [<options>]
```

Options

```
-s|--sfn=<stepfunctionname>        An example step function name may be
                                   'tibanna_unicorn_defaut_3978'. If not specified,␣
↪default

                                   value is taken from environmental variable
                                   TIBANNA_DEFAULT_STEP_FUNCTION_NAME.
-j JOBID, --jobid JOBID            specify a user-defined job id (randomly generated␣
↪if

                                   not specified)
-B, --do-not-open-browser          Do not open browser
-S SLEEP, --sleep SLEEP            Number of seconds between submission, to avoid␣
↪drop-

                                   out (default 3)
```

run_batch_workflows

To run multiple workflows in a batch. This command does not open browser and job ids are always automatically assigned. This function is available for Tibanna versions >= 1.0.0.

```
tibanna run_batch_workflows -i <input_json_file> [<input_json_file2>] [...] [<options>
↪]
```

Options

```
-s|--sfn=<stepfunctionname>        An example step function name may be
                                   'tibanna_unicorn_defaut_3978'. If not specified,␣
↪default

                                   value is taken from environmental variable
                                   TIBANNA_DEFAULT_STEP_FUNCTION_NAME.
-S SLEEP, --sleep SLEEP            Number of seconds between submission, to avoid␣
↪drop-

                                   out (default 3)
```

stat

To check status of workflows,

```
tibanna stat [<options>]
```

Options

```
-t|--status=<status>                filter by run status (all runs if not␣
↪specified).

                                    Status must be one of the following values:
                                    RUNNING|SUCCEEDED|FAILED|TIMED_OUT|ABORTED

-s|--sfn=<stepfunctionname>         An example step function name may be
                                    'tibanna_unicorn_defaut_3978'. If not specified,
↪ default

                                    value is taken from environmental variable
                                    TIBANNA_DEFAULT_STEP_FUNCTION_NAME. If the␣
↪environmental

                                    variable is not set, it uses name 'tibanna_pony
↪' (4dn

                                    default, works only for 4dn).

-n|--nlines=<number_of_lines>       print out only the first n lines

-j|--job-ids <job_id> [<job_id2>] ... job ids of the specific jobs to display,␣
↪separated by

                                    space. This option cannot be combined with
                                    --nlines(-n), --status(-t) or --sfn(-s). This␣
↪option is

                                    available only for version >= ``1.0.0``.
```

The output is a table (an example below)

```
jobid         status   name     start_time        stop_time
2xPih7reR6FM         RUNNING md5   2018-08-15 17:45        2018-08-15 17:50
3hbkJB3hv92S         SUCCEEDED    hicprocessingbam        2018-08-15 16:04         ␣
↪2018-08-15 16:09
UlkvH3gbBBA2         FAILED  repliseq-parta  2018-08-09 18:26        2018-08-09 19:01
j7hvisheBV27         SUCCEEDED    bwa-mem 2018-08-09 18:44        2018-08-09 18:59
```

log

To check the log or postrun json (summary) of a workflow run

```
tibanna log --exec-arn=<stepfunctionrun_arn>|--job-id=<jobid> [<options>]
```

or

```
tibanna log --exec-name=<exec_name> --sfn=<stepfunctionname> [<options>]
```

Options

```
-p|--postrunjson     The -p option streams out a postrun json file instead of a log␣
↪file.

                     A postrun json file is available only after the run finishes.
                     It contains the summary of the job including input, output, EC2␣
↪config and

                     Cloudwatch metrics on memory/CPU/disk space.
```

(continues on next page)

```
-r|--runjson          print out run json instead, which is the json file tibanna␣
→sends to the instance
                      before the run starts. (new in ``1.0.0``)

-t|--top              prints out top file (log file containing top command
                      output) instead. This top file contains all the top batch␣
→command output
                      at a 1-minute interval. (new in ``1.0.0``)

-T|--top-latest       prints out the latest content of the top file. This one␣
→contains only the latest
                      top command output (latest 1-minute interval). (new in ``1.0.
→0``)
```

rerun

To rerun a failed job with the same input json on a specific step function.

```
tibanna rerun --exec-arn=<execution_arn>|--job-id=<jobid> --sfn=<target_stepfunction_
→name> [<options>]
```

Options

```
-i|--instance-type=<instance_type>  Override instance type for the rerun

-d|--shutdown-min=<shutdown_min>    Override shutdown minutes for the rerun

-b|--ebs-size=<ebs_size>            Override EBS size for the rerun

-T|--ebs-type=<ebs_size>            Override EBS type for the rerun

-p|--ebs-iops=<ebs_iops>            Override EBS IOPS for the rerun

-k|--key-name=<key_name>            Override key name for the rerun

-n|--name=<run_name>               Override run name for the rerun

-a|--appname-filter=<appname>       Rerun only if the app name matches the specified␣
→app name.
```

rerun_many

To rerun many jobs that failed after a certain time point

```
tibanna rerun_many [<options>]
```

Options

```
-s|--sfn=<stepfunctionname>         An example step function name may be
                                    'tibanna_unicorn_defaut_3978'. If not specified,␣
→default
                                    value is taken from environmental variable
```

```
                                      TIBANNA_DEFAULT_STEP_FUNCTION_NAME. If the␣
↪environmental

                                      variable is not set, it uses name 'tibanna_pony'␣
↪(4dn

                                      default, works only for 4dn).

-D|--stopdate=<stopdate>              e.g. '14Feb2018'

-H|--stophour=<stophour>              e.g. 14 (24-hour format, same as system time zone␣
↪by default)

-M|--stopminute=<stopminute>          e.g. 30 (default 0)

-r|--sleeptime=<sleeptime>            seconds between reruns (eefault 5)

-o|--offset=<offset>                  offset between AWS time zone and system time zone␣
↪(default 0)

                                      e.g. if 17:00 by AWS time zone corresponds to␣
↪12:00 by system

                                      time zone, offset must be 5.

-t|--status=<status>                  filter by status. default 'FAILED', i.e. rerun␣
↪only failed

                                      jobs

-i|--instance-type=<instance_type>  Override instance type for the rerun

-d|--shutdown-min=<shutdown_min>    Override shutdown minutes for the rerun

-b|--ebs-size=<ebs_size>             Override EBS size for the rerun

-T|--ebs-type=<ebs_size>             Override EBS type for the rerun

-p|--ebs-iops=<ebs_iops>             Override EBS IOPS for the rerun

-k|--key-name=<key_name>             Override key name for the rerun

-n|--name=<run_name>                 Override run name for the rerun

-a|--appname-filter=<appname>        Rerun only if the app name matches the specified␣
↪app name.
```

Example

```
tibanna rerun_many --stopdate=14Feb2018 --stophour=15
```

This example will rerun all the jobs of default step function that failed after 3pm on Feb 14 2018.

kill

To kill a specific job through its execution arn or a jobid

```
tibanna kill --exec-arn=<execution_arn>|--job-id=<jobid>
```

If the execution id or job id is not found in the current RUNNING executions (e.g. the execution has already been aborted), then only the EC2 instance will be terminated.

Example

For example, let's say we run the following job by mistake.

```
$ tibanna run_workflow --input-json=fastqc.json
```

The following message is printed out

```
about to start run fastqc_85ba7f41-daf5-4f82-946f-06d31d0cd293
response from aws was:
{u'startDate': datetime.datetime(2018, 10, 11, 20, 15, 0, 71000, tzinfo=tzlocal()),
→'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId':
→'54664dcc-cd92-11e8-a2c0-51ce6ca6c6ea', 'HTTPHeaders': {'x-amzn-requestid':
→'54664dcc-cd92-11e8-a2c0-51ce6ca6c6ea', 'content-length': '161', 'content-type':
→'application/x-amz-json-1.0'}}, u'executionArn': u'arn:aws:states:us-east-
→1:643366669028:execution:tibanna_unicorn_default3537:fastqc_85ba7f41-daf5-4f82-946f-
→06d31d0cd293'}
url to view status:
https://console.aws.amazon.com/states/home?region=us-east-1#/executions/details/
→arn:aws:states:us-east-1:643366669028:execution:tibanna_unicorn_default3537:fastqc_
→85ba7f41-daf5-4f82-946f-06d31d0cd293
JOBID jLeL6vMbhL63 submitted
EXECUTION ARN = arn:aws:states:us-east-1:643366669028:execution:tibanna_unicorn_
→default3537:fastqc_85ba7f41-daf5-4f82-946f-06d31d0cd293
```

To kill this job, use the execution arn in the above message ('EXECUTION_ARN') (it can also be found on the Step Function Console)

```
$ tibanna kill --exec-arn=arn:aws:states:us-east-1:643366669028:execution:tibanna_
→unicorn_default3537:fastqc_85ba7f41-daf5-4f82-946f-06d31d0cd293
```

or

```
$ tibanna kill --job-id jLeL6vMbhL63
```

kill_all

To kill all currently running jobs for a given step function

```
tibanna kill_all
```

Options

```
-s|--sfn=<stepfunctionname>         An example step function name may be
                                    'tibanna_unicorn_defaut_3978'. If not specified,␣
→default
                                    value is taken from environmental variable
                                    TIBANNA_DEFAULT_STEP_FUNCTION_NAME. If the␣
→environmental
                                    variable is not set, it uses name 'tibanna_pony'␣
→(4dn
                                    default, works only for 4dn).
```

list_sfns

To list all step functions

```
tibanna list_sfns [-n]
```

Options

```
-n      show stats of the number of jobs for per status (using this option could slow␣
→down the
        process)
```

plot_metrics

To collect, save and visualize the resources metrics from Cloud Watch

```
tibanna plot_metrics --job-id=<jobid> [<options>]
```

Options

```
-s|--sfn=<stepfunctionname>         An example step function name may be
                                    'tibanna_unicorn_defaut_3978'. If not specified,␣
→default

                                    value is taken from environmental variable
                                    TIBANNA_DEFAULT_STEP_FUNCTION_NAME. If the␣
→environmental

                                    variable is not set, it uses name 'tibanna_pony'␣
→(4dn

                                    default, works only for 4dn).

-f|--force-upload                   This flag force the upload of the metrics reports
                                    to the S3 bucket, even if there is a lock (upload
                                    is blocked by default by the lock)

-u|--update-html-only               This flag specify to only update the html file
                                    for metrics visualization,
                                    metrics reports are not updated

-B|--do-not-open-browser            Do not open the browser to visualize the metrics␣
→html

                                    after it has been created/updated

-i|--instance-id=<instance_id>      Manually provide instance ID in case Tibanna␣
→somehow

                                    can't find the information. This field is not␣
→required normally.
```

cost

To retrieve the cost and update the metrics report file created with plot_metrics. The cost is typically available 24 hours after the job finished. This function is available to non-admin users from version 1.0.6.

```
tibanna cost --job-id=<jobid> [<options>]
```

Options

```
-s|--sfn=<stepfunctionname>            An example step function name may be
                                       'tibanna_unicorn_defaut_3978'. If not specified,␣
↪default
                                       value is taken from environmental variable
                                       TIBANNA_DEFAULT_STEP_FUNCTION_NAME. If the␣
↪environmental
                                       variable is not set, it uses name 'tibanna_pony'␣
↪(4dn
                                       default, works only for 4dn).

-u|--update-tsv                        Update with the cost the tsv file that stores␣
↪metrics
                                       information on the S3 bucket
```

cost_estimate

To retrieve cost estimates and update the metrics report file created with plot_metrics. In contrast to the exact costs, the estimated costs are available immediately after the job has completed. The cost estimate will also indicate if it is an immediate estimate (i.e., the exact cost is not yet available), the actual cost or the retrospective estimate (i.e., the exact cost is not available anymore). In case the estimate returns the actual cost and the *-u* parameter is set, the cost row in the metrics file will be automatically updated. This function requires a (deployed) Tibanna version >=1.0.6.

```
tibanna cost_estimate --job-id=<jobid> [<options>]
```

Options

```
-u|--update-tsv                        Update with the cost the tsv file that stores␣
↪metrics
                                       information on the S3 bucket

-f|--force                             Return the estimate, even if the actual cost is␣
↪available
```

1.4.6 Python API

All the API functions are in the `API` class in `tibanna.core`. Note that the class must be *instantiated* first (`API().run_workflow` rather than `API.run_workflow`).

General Usage

```python
from tibanna.core import API
API().method(...)
```

Example

```python
from tibanna.core import API
API().run_workflow(input_json='myrun.json')  # json file or dictionary object
```

Admin only commands

The following commands require admin previlege to one's AWS account.

---

deploy_unicorn

To create an instance of tibanna unicorn (step function + lambdas)

```
API().deploy_unicorn(...)
```

Parameters

```
buckets=<bucket1,bucket2,...>       List of buckets as a string to use for tibanna␣
↪runs.

                                    The associated lambda functions, EC2 instances
                                    and user group will be given permission to these␣
↪buckets.

no_setup                            Skip setup buckets/permissions and just redeploy␣
↪tibanna

                                    step function and lambdas.
                                    This is useful when upgrading the existing␣
↪tibanna that's

                                    already set up.

no_setenv                           Do not overwrite TIBANNA_DEFAULT_STEP_FUNCTION_
↪NAME

                                    environmental variable in your bashrc.

suffix                              Using suffix helps deploying various dev-version␣
↪tibanna.

                                    The step function and lambda functions will have␣
↪the suffix.

usergroup                           Tibanna usergroup to share the permission to␣
↪access

                                    buckets and run jobs

do_not_delete_public_access_block   If set True, Tibanna does not delete public
                                    access block from the specified buckets
                                    (this way postrunjson and metrics reports will
                                    not be public). Default False.
```

Note: starting `0.9.0`, users do not need to export `AWS_ACCOUNT_NUMBER` and `TIBANNA_AWS_REGION` any more.

deploy_core

Deploy/update only a single lambda function

```
API().deploy_core(name=<lambda_name>, ...)
```

where `<lambda_name>` would be either `run_task_awsem` or *check_task_awsem'*.

Options

```
suffix=<suffixname>                 Using suffix helps deploying various dev-version␣
↪tibanna.

                                    The step function and lambda functions will have the␣
↪suffix.
```

(continues on next page)

```
usergroup=<usergroup>            Tibanna usergroup to share the permission to access
                                 buckets and run jobs
```

users

To list users

```
API().users()
```

add_user

To add users to a tibanna user group

```
API().add_user(user=<user>, usegroup=<usergroup>)
```

cleanup

To remove Tibanna components on AWS.

```
API().cleanup(user_group_name=<usergroup>, ...)
```

Options

```
suffix=<suffixname>              If suffix was used to deploy a tibanna, it should be␣
→added
                                 here. The step function and lambda functions will have␣
→the
                                 suffix at the end.

ignore_errors=<True|False>       If True, if any of the components does not exist (e.g.
                                 already removed), it does not throw an error and keeps␣
→on
                                 to remove the other components. (default True)

do_not_remove_iam_group<True|False>
                                 If True, does not remove the IAM permission. This␣
→option is
                                 recommended if various suffices are used to share the␣
→same
                                 usergroup. (default False)

purge_history=<True|False>       If True, remove all the job logs and other job-related␣
→files
                                 from S3 bucket and dynamoDB. Please use with caution.
                                 (default False)

verbose=<True|False>             Verbose if True. (default False)
```

setup_tibanna_env

- Advanced user only

To set up environment on AWS without deploying tibanna, use *tibanna setup_tibanna_env*.

```
API().setup_tibanna_env(...)
```

Options

```
usergroup_tag=<usergrouptag>         an identifier for a usergroup that shares
                                     a tibanna permission

no_randomize                         If set True, Tibanna does not add a random
                                     number to generate a usergroup name (e.g. the
                                     usergroup name used will be identical to the
                                     one specified using the ``usergrou_tag`` option.
                                     By default, a random number will be added at the
                                     end (e.g. default_2721). Default False.

buckets=<bucket_list>                A comma-delimited list of bucket names - the
                                     buckets to which Tibanna needs access to
                                     through IAM role (input, output, log).

do_not_delete_public_access_block    If set True, Tibanna does not delete public
                                     access block from the specified buckets
                                     (this way postrunjson and metrics reports will
                                     not be public). Default False.
```

Non-admin commands

The following commands can be used by a non-admin user, as long as the user belongs to the right user group.

run_workflow

To run workflow

```
API().run_workflow(input_json=<input_json_file|input_dict>, ...)
```

Options

```
sfn=<stepfunctionname>            An example step function name may be
                                  'tibanna_unicorn_defaut_3978'. If not specified,
→default
                                  value is taken from environmental variable
                                  TIBANNA_DEFAULT_STEP_FUNCTION_NAME.
jobid=<JOBID>                     specify a user-defined job id (randomly generated if
                                  not specified)
open_browser=<True|False>         Open browser (default True)
sleep=<SLEEP>                     Number of seconds between submission, to avoid drop-
                                  out (default 3)
```

run_batch_workflows

To run multiple workflows in a batch. This function does not open browser and job ids are always automatically assigned. This function is available for Tibanna versions >= `1.0.0`.

```
API().run_batch_workflows(input_json_list=<list_of_input_json_files_or_dicts>, ...)
```

Options

```
sfn=<stepfunctionname>          An example step function name may be
                                'tibanna_unicorn_defaut_3978'. If not specified,␣
→default
                                value is taken from environmental variable
                                TIBANNA_DEFAULT_STEP_FUNCTION_NAME.
sleep=<SLEEP>                   Number of seconds between submission, to avoid drop-
                                out (default 3)
```

stat

To check status of workflows,

```
API().stat(...)
```

Options

```
status=<status>                 filter by run status (all runs if not specified).
                                Status must be one of the following values:
                                RUNNING|SUCCEEDED|FAILED|TIMED_OUT|ABORTED

sfn=<stepfunctionname>          An example step function name may be
                                'tibanna_unicorn_defaut_3978'. If not specified,␣
→default
                                value is taken from environmental variable
                                TIBANNA_DEFAULT_STEP_FUNCTION_NAME.

n=<number_of_lines>             print out only the first n lines

job_ids=<list_of_job_ids>       filter by a list of job ids. This option is
                                available only for version >= ``1.0.0``.
```

The output is a table (an example below)

```
jobid        status    name      start_time        stop_time
2xPih7reR6FM          RUNNING md5     2018-08-15 17:45        2018-08-15 17:50
3hbkJB3hv92S          SUCCEEDED     hicprocessingbam        2018-08-15 16:04          ␣
→2018-08-15 16:09
UlkvH3gbBBA2          FAILED  repliseq-parta 2018-08-09 18:26        2018-08-09 19:01
j7hvisheBV27          SUCCEEDED       bwa-mem 2018-08-09 18:44        2018-08-09 18:59
```

log

To check the log or postrun json (summary) of a workflow run

```
API().log(exec_arn=<stepfunctionrun_arn>|job_id=<jobid>, ...)
```

Options

```
postrunjson=<True|False>          The postrunjson option streams out a postrun json file␣
↪instead of a log file.
                                  A postrun json file is available only after the run␣
↪finishes.
                                  It contains the summary of the job including input,␣
↪output, EC2 config and
                                  Cloudwatch metrics on memory/CPU/disk space.

runjson=<True|False>              prints out run json instead, which is the json file␣
↪tibanna sends to the instance
                                  before the run starts. (new in ``1.0.0``)

top=<True|False>                  prints out top file (log file containing top command␣
↪output) instead. This top file
                                  contains all the top batch command output at a 1-
↪minute interval. (new in ``1.0.0``)

top_latest=<True|False>           prints out the latest content of the top file. This␣
↪one contains only the latest
                                  top command output (latest 1-minute interval). (new in␣
↪``1.0.0``)
```

rerun

To rerun a failed job with the same input json on a specific step function.

```
API().rerun(exec_arn=<execution_arn>|job_id=<jobid>, sfn=<target_stepfunction_name>, .
↪..)
```

Options

```
instance_type=<instance_type>  Override instance type for the rerun

shutdown_min=<shutdown_min>    Override shutdown minutes for the rerun

ebs_size=<ebs_size>            Override EBS size for the rerun

ebs_type=<ebs_size>            Override EBS type for the rerun

ebs_iops=<ebs_iops>            Override EBS IOPS for the rerun

ebs_throughput=<ebs_throughput>        Override EBS GP3 throughput for the rerun

key_name=<key_name>            Override key name for the rerun

name=<run_name>                Override run name for the rerun

appname_filter=<appname>       Rerun only if the app name matches the specified app␣
↪name.
```

rerun_many

To rerun many jobs that failed after a certain time point

```
API().rerun_many(...)
```

Options

sfn=<stepfunctionname> default	An example step function name may be `'tibanna_unicorn_defaut_3978'`. If **not** specified, value **is** taken **from environmental** variable TIBANNA_DEFAULT_STEP_FUNCTION_NAME.
stopdate=<stopdate>	e.g. `'14Feb2018'`
stophour=<stophour> default)	e.g. `14` (24-hour `format`, same **as** system time zone by
stopminute=<stopminute>	e.g. `30` (default `0`)
sleeptime=<sleeptime>	seconds between reruns (eefault `5`)
offset=<offset> (default `0`) system time zone, offset must be `5`.	offset between AWS time zone **and** system time zone e.g. **if** `17:00` by AWS time zone corresponds to `12:00` by
status=<status> failed	`filter` by status. default `'FAILED'`, i.e. rerun only jobs
instance_type=<instance_type>	Override instance `type` **for** the rerun
shutdown_min=<shutdown_min>	Override shutdown minutes **for** the rerun
ebs_size=<ebs_size>	Override EBS size **for** the rerun
ebs_type=<ebs_size>	Override EBS `type` **for** the rerun
ebs_iops=<ebs_iops>	Override EBS IOPS **for** the rerun
ebs_throughput=<ebs_throughput>	Override EBS GP3 throughput **for** the rerun
key_name=<key_name>	Override key name **for** the rerun
name=<run_name>	Override run name **for** the rerun
appname_filter=<appname> name.	Rerun only **if** the app name matches the specified app

Example

```
API().rerun_many(stopdate='14Feb2018', stophour=15)
```

This example will rerun all the jobs of default step function that failed after 3pm on Feb 14 2018.

kill

To kill a specific job through its execution arn or a jobid

```
API().kill(exec_arn=<execution_arn>)
```

or

```
API().kill(job_id=<jobid>, sfn=<stepfunctionname>)
```

If `jobid` is specified but not `stepfunctionname`, then by default it assumes `TIBANNA_DEFAULT_STEP_FUNCTION_NAME`. If the job id is not found in the executions on the default or specified step function, then only the EC2 instance will be terminated and the step function status may still be RUNNING.

Example

For example, let's say we run the following job by mistake.

```
API().run_workflow(input_json='fastqc.json')
```

The following message is printed out

```
about to start run fastqc_85ba7f41-daf5-4f82-946f-06d31d0cd293
response from aws was:
{u'startDate': datetime.datetime(2018, 10, 11, 20, 15, 0, 71000, tzinfo=tzlocal()),
→'ResponseMetadata': {'RetryAttempts': 0, 'HTTPStatusCode': 200, 'RequestId':
→'54664dcc-cd92-11e8-a2c0-51ce6ca6c6ea', 'HTTPHeaders': {'x-amzn-requestid':
→'54664dcc-cd92-11e8-a2c0-51ce6ca6c6ea', 'content-length': '161', 'content-type':
→'application/x-amz-json-1.0'}}, u'executionArn': u'arn:aws:states:us-east-
→1:643366669028:execution:tibanna_unicorn_default3537:fastqc_85ba7f41-daf5-4f82-946f-
→06d31d0cd293'}
url to view status:
https://console.aws.amazon.com/states/home?region=us-east-1#/executions/details/
→arn:aws:states:us-east-1:643366669028:execution:tibanna_unicorn_default3537:fastqc_
→85ba7f41-daf5-4f82-946f-06d31d0cd293
JOBID jLeL6vMbhL63 submitted
EXECUTION ARN = arn:aws:states:us-east-1:643366669028:execution:tibanna_unicorn_
→default3537:fastqc_85ba7f41-daf5-4f82-946f-06d31d0cd293
```

To kill this job, use the execution arn in the above message ('EXECUTION_ARN') (it can also be found on the Step Function Console)

```
API().kill(exec_arn='arn:aws:states:us-east-1:643366669028:execution:tibanna_unicorn_
→default3537:fastqc_85ba7f41-daf5-4f82-946f-06d31d0cd293')
```

or

```
API().kill(job_id='jLeL6vMbhL63')
```

kill_all

To kill all currently running jobs for a given step function

```
API().kill_all(...)
```

Options

```
sfn=<stepfunctionname>            An example step function name may be
                                  'tibanna_unicorn_defaut_3978'. If not specified,␣
→default
                                  value is taken from environmental variable
                                  TIBANNA_DEFAULT_STEP_FUNCTION_NAME.
```

list_sfns

To list all step functions

```
API().list_sfns(...)
```

Options

```
n       show stats of the number of jobs for per status (using this option could slow␣
→down the
        process)
```

plot_metrics

To collect, save and visualize the resources metrics from Cloud Watch

```
API().plot_metrics(job_id=<jobid>, ...)
```

Options

```
sfn=<stepfunctionname>            An example step function name may be
                                  'tibanna_unicorn_defaut_3978'. If not specified,␣
→default
                                  value is taken from environmental variable
                                  TIBANNA_DEFAULT_STEP_FUNCTION_NAME. If the␣
→environmental
                                  variable is not set, it uses name 'tibanna_pony'␣
→(4dn
                                  default, works only for 4dn).

force_upload                      This flag force the upload of the metrics reports
                                  to the S3 bucket, even if there is a lock (upload
                                  is blocked by default by the lock)

update_html_only                  This flag specify to only update the html file for
                                  metrics visualization,
                                  metrics reports are not updated

open_browser                      This flag specify to not open the browser to␣
→visualize
                                  the metrics html after it has been created/updated

filesystem=<filesystem>           Define the filesystem of the EC2 instance, default
                                  value is '/dev/nvme1n1'

endtime=<endtime>                 End time of the interval to be considered
                                  to retrieve the data
```

(continues on next page)

```
instance_id=<instance_id>            Manually provide instance ID in case Tibanna␣
↪somehow
                                     can't find the information. This field is not␣
↪required
                                     normally.
```

cost

To retrieve the cost and update the metrics report file created with plot_metrics

```
API().cost(job_id=<jobid>, ...)
```

Options

```
sfn=<stepfunctionname>               An example step function name may be
                                     'tibanna_unicorn_defaut_3978'. If not specified,␣
↪default
                                     value is taken from environmental variable
                                     TIBANNA_DEFAULT_STEP_FUNCTION_NAME. If the␣
↪environmental
                                     variable is not set, it uses name 'tibanna_pony'␣
↪(4dn
                                     default, works only for 4dn).

update_tsv                           This flag specifies wether to update the cost in␣
↪the tsv file that
                                     stores metrics information on the S3 bucket
```

cost_estimate

Retrieve a cost estimate for a specific job. This will be available as soon as the job finished. This function will return the exact cost, if it is available. The cost estimate will also indicate if it is an immediate estimate (i.e., the exact cost is not yet available), the actual cost or the retrospective estimate (i.e., the exact cost is not available anymore). In case the estimate returns the actual cost and the *-u* parameter is set, the cost row in the metrics file will be automatically updated.

```
API().cost_estimate(job_id=<jobid>, ...)
```

Options

```
update_tsv                           This flag specifies wether to update the cost in␣
↪the tsv file that
                                     stores metrics information on the S3 bucket

force                                Return the estimate, even if the actual cost is␣
↪available
```

1.4.7 Job Description JSON Schema

The Job Description json (input of Tibanna) defines an individual execution. It has two parts, *args* and *config*. *args* contains information about pipeline, input files, output bucket, input parameters, etc. *config* has parameters about AWS such as instance type, EBS size, ssh password, etc.

Example job description for CWL

```
{
  "args": {
    "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/pipelines-cwl/0.
↪2.0/cwl_awsem/",
    "cwl_main_filename": "pairsam-parse-sort.cwl",
    "cwl_version": "v1",
    "input_files": {
      "bam": {
        "bucket_name": "montys-data-bucket",
        "object_key": "dataset1/sample1.bam"
      },
      "chromsize": {
        "bucket_name": "montys-data-bucket",
        "object_key": "references/hg38.chrom.sizes"
      }
    },
    "input_parameters": {
      "nThreads": 16
    },
    "input_env": {
      "TEST_ENV_VAR": "abcd"
    },
    "output_S3_bucket": "montys-data-bucket",
    "output_target": {
      "out_pairsam": "output/dataset1/sample1.sam.pairs.gz"
    },
    "secondary_output_target": {
      "out_pairsam": "output/dataset1/sample1.sam.pairs.gz.px2"
    }
  },
  "config": {
    "instance_type": "t3.micro",
    "ebs_size": 10,
    "EBS_optimized": true,
    "log_bucket": "montys-log-bucket"
  }
}
```

args

The `args` field describe pipeline, input and output.

Pipeline specification

CWL-specific

cwl_directory_url

- <url_that_contains_cwl_file(s)>
- (e.g. 'https://raw.githubusercontent.com/4dn-dcic/pipelines-cwl/0.2.0/cwl_awsem')
- (e.g. 's3://bucketname/dirname/dirname2')

- The http url must be public.

- For the s3 url, the bucket must have been included during the `deploy_unicorn` run (accessible by tibanna)

cwl_directory_local

- <local_directory_that_contains_cwl_file(s)>

- If this is set, `cwl_directory_url` can be skipped.

cwl_main_filename

- <main_cwl_file> (e.g. 'pairsam-parse-sort.cwl')

- This file must be in the cwl url given by `cwl_directory_url`.

- The actual cwl link would be `cwl_directory_url` + '' + `cwl_main_file_name`

cwl_child_filenames

- <list_of_cwl_files> or `[]` (e.g. ['step1.cwl', 'step2.cwl'])

- An array of all the other cwl files that are called by the main cwl file. If the main CWL file is of 'workflow' type, the other CWL files corresponding to steps or subworkflows should be listed here.

cwl_version

- either `v1` or `draft-3` (starting with tibanna version `1.0.0`, `draft-3` is no longer supported.)

singularity

- This option uses Singularity to run Docker images internally (slower). This option does NOT support native Singularity images, since CWL does not support native Singularity images.

- either `true` or `false`

- This is an optional field. (default `false`)

WDL-specific

language

- This field must be set to `wdl` to run a WDL pipeline.

- To run an old version (draft2) of WDL, set it to `wdl_draft2`. This will direct Tibanna to specifically use an older version of Cromwell. Some draft2 WDLs may be supported by the later version of Cromwell. Use the `wdl_draft2` option only if the old WDL does not work with the later version of Cromwell.

wdl_directory_url

- <url_that_contains_wdl_file(s)>

- (e.g. 'https://raw.githubusercontent.com/4dn-dcic/pipelines-cwl/master/wdl')

- (e.g. 's3://bucketname/dirname/dirname2')

- The http url must be public.

- For the s3 url, the bucket must have been included during the `deploy_unicorn` run (accessible by tibanna)

wdl_directory_local

- <local_directory_that_contains_wdl_file(s)>

- If this is set, `wdl_directory_url` can be skipped.

wdl_main_filename

- <main_wdl_file> (e.g. 'pairsam-parse-sort.wdl')

- This file must be in the wdl url given by `wdl_directory_url`.

- The actual wdl link would be `wdl_directory_url` + '' + `wdl_file_name`

wdl_child_filenames

- <list_of_wdl_files> or `[]` (e.g. ['subworkflow1.wdl', 'subworkflow2.wdl'])

- An array of all the other wdl files that are called by the main wdl file. This could happen if there are the main WDL file is using another WDL file as a subworkflow.

Shell command-specific

language

- This field must be set to `shell` to run a shell command without CWL/WDL.

container_image

- <Docker image name>

command

- <shell command to be executed inside the Docker container>

- a pair of nested double quotes are allowed

- (e.g.

```
"command": "echo \"haha\" > outfile"
```

Snakemake-specific

language

- This field must be set to `snakemake` to run a Snakemake pipeline.

container_image

- This is a required field.

- It is highly recommended to use the official Snakemake Docker image (`snakemake/snakemake`)

command

- This is a required field.

- Most likely it will be `snakemake` but it can be run with other `snakemake` otions.

- (e.g.

```
"command": "snakemake <target> --use-conda"
```

- a pair of nested double quotes are allowed

- (e.g.

```
"command": "snakemake <target> --config=region=\"22:30000000-40000000\""
```

snakemake_main_filename

- This is a required field.

- Most likely it will be `Snakefile` (do not include directory name).

snakemake_child_filenames

- This is an optional field.

- This may include other workflow-related files including `env.yml`, `config.json`, etc. (Do not include directory name).

snakemake_directory_local

- The location (directory path) of the *snakemake_main_filename* and `snake_child_filenames`.

- Use this if the workflow files are local.

snakemake_directory_url

- The url (directory only) of the *snakemake_main_filename* and `snake_child_filenames`.

- Use this if the worlfow files are accessible through a url (either `http://` or `s3://`.

Other pipeline-related fields

app_name

- <name of the app> (e.g. 'pairsam-parse-sort')

- A alphanumeric string that can identify the pipeline/app. May contain '-' or '_'.

- This field is optional and is used only by `Benchmark` which auto-termines instance type and EBS size based on input size and parameters. If the workflow doesn't have an associated Benchmark function, this field can be omitted, but `instance_type` (or `mem` and `cpu`), `ebs_size` (unless using default 10GB), `EBS_optimized` (unless using default `False`) must be specified in `config`.

app_version

- optional

- <version of the app> (e.g. 0.2.0)

- Version of the pipeline/app, for the user to keep in track.

language

- 'cwl_v1', 'cwl_draft3' (tibanna < `1.0.0` only) or 'wdl' (='wdl_v1' for backward compatibility) or 'wdl_draft2' or 'wdl_v1' (tibanna >= `1.0.0`)

- For WDL, it is a required field. For CWL, the language field can be omitted.

run_args

- Optional, available for tibanna > `1.9.1`

- Additional command line arguments that are passed to the `cwltool`/`cromwell.jar run`/`caper run` commands.

Input data specification

input_files

- A dictionary that contains input files. The keys must match the input argument names of the CWL/WDL.

- It contains `bucket_name`, `object_key` as required fields.

- Optionally, it may contain the following fields:

  - `profile` if the bucket can only be accessed through profile (profile can be set during Tibanna deployment)

  - `rename` if the file name must be changed upon download to the EC2 instance. This could be useful if your files are organized in certain names on S3 but the pipeline requires it to have a different name.

  - `unzip` to unzip the file during the upload to the EBS volume. Supported compression types are "gz" and "bz2".

  - `mount` to mount the input instead of downloading. This saves downloading time but may slow down the file reading slightly. The mounting is done at the bucket level to the EBS. We have tested up to 50 instances concurrently mounting the same bucket with no problem - if you're running 10,000 jobs, we cannot guarantee if this would still work. `mount` and `rename` cannot be used together. If another input file is specified without mount but from the same bucket, this other input file will be downloaded to the running instance even though the bucket is mounted.

- `object_key` and `rename` can be a singleton, an array, an array of arrays or an array of arrays of arrays.

- (e.g.

```
{
    "bam": {
        "bucket_name": "montys-data-bucket",
        "object_key": "dataset1/sample1.bam",
        "mount": true
    },
    "chromsize": {
        "bucket_name": "montys-data-bucket",
        "object_key": "references/JKGFALIFVG.chrom.sizes"
        'rename': 'some_dir_on_ec2/hg38.chrom.sizes'
    }
}
```

)

- key can be a target file path (to be used inside container run environment) starting with `file:/` `/` instead of CWL/WDL argument name.

  - Input data can only be downloaded to `/data1/input` or `/data1/` `<language_name>` where `<language_name` is `cwl|wdl|shell|snakemake`. The latter `/data1/<language_name>` is the working directory for `snakemake` and `shell`.

  - It is highly recommended to stick to using only argument names for CWL/WDL for pipeline reproducibility, since they are already clearly defined in CWL/WDL (especially for CWL).

  - (e.g.

```
{
    "file:///data1/shell/mysample1.bam": {
        "bucket_name": "montys-data-bucket",
        "object_key": "dataset1/sample1.bam"
    }
}
```

secondary_files

- A dictionary of the same format as *input_file* but contains secondary files.

- The keys must match the input argument name of the CWL/WDL where the secondary file belongs.

- (e.g.

```
{
    "bam": {
        "bucket_name": "montys-data-bucket",
        "object_key": "dataset1/sample1.bam.bai"
    }
}
```

)

input_parameters

- A dictionary that contains input parameter values. Default parameters don't need to be included. The keys must match the input argument name of the CWL/WDL.

- (e.g.

```
{
    'nThreads': 16
}
```

)

input_env

- A dictionary that specifies environment variables to be passed.

- Do not use this feature to pass in AWS_ACCESS_KEY and/or AWS_SECRET_KEY or AWS_REGION - it will interfere with the bucket permission of the instance.

- (e.g.

```
{
    "TEST_ENV_VAR": "abcd"
}
```

)

Output target specification

output_S3_bucket

- The name of the bucket where output files will be sent to.

output_target

- A dictionary that contains a desired object keys to be put inside output bucket. This can be useful if, for example, the pipeline always generates an output file of the same name (e.g. report, output.txt, etc) but the user wants to distinguish them by sample names in the output bucket. If not set, the original output file names will be used as object key.

- (e.g.

```
{
  "out_pairsam": "output/dataset1/sample1.sam.pairs.gz"
}
```

)

- key can be a source file path (to be used inside container run environment) starting with `file:/` `/` instead of CWL/WDL argument name.

- (e.g.

```
{
  "file:///data1/out/some_random_output.txt": "output/some_random_output.
↪txt"
}
```

- It is highly recommended to stick to using only argument names for CWL/WDL for pipeline reproducibility, since they are already clearly defined in CWL/WDL (especially for CWL).

- Starting with version `1.0.0`, a dictionary format is also accepted for individual target, with keys `object_key` `bucket_name`, `object_prefix` and/or `unzip`. For a regular file output, `object_key` and `bucket_name` can be used. The use of `bucket_name` here allows using a different output bucket for specific output files. For a directory, `object_prefix` can be used instead which will be used as if it is the directory name on S3. `object_prefix` may or may not have the trailing `/`. `unzip` is boolean (either `true` or `false`) and can be applied to a case when the output file is a `zip` file and you want the content to be extracted into a directory on an S3 bucket.

- (e.g.

```
{
    "out_pairsam": {
        "object_key": "output/renamed_pairsam_file"
    }
}
```

```
{
    "out_pairsam": {
        "object_key": "output/renamed_pairsam_file",
        "bucket_name" : "some_different_bucket"
    }
}
```

```
{
    "some_output_as_dir": {
        "object_prefix": "some_dir_output/",
        "bucket_name": "some_different_bucket"
    }
}
```

```
{
    "out_zip": {
        "object_prefix": "zip_output/",
        "unzip": true
}
```

- One or multiple tags can be automatically added to each ouput file by specifying the `tag` key. In the following example, two (object-level) tags are added to the result file. Note that the tag-set must be encoded as URL Query parameters. In case the `unzip` key is specified in addition to the `tag` key, each file in the output directory will be tagged.

```
{
    "out_zip": {
        "object_key": "result.txt",
        "tag": "Key1=Value1&Key2=Value2"
}
```

secondary_output_target

- Similar to `output_target` but for secondary files.

- (e.g.

```
{
    "out_pairsam": "output/dataset1/sample1.sam.pairs.gz.px2"
}
```

)

alt_cond_output_argnames

- In case output argnames are conditional (see an example in simple_example_cond_merge), specify a global output name that can point to one of the conditional outputs.

- This applies only to WDL since CWL does not support conditional statements.

- (e.g.

```
'alt_cond_output_argnames' : {
  'merged' : ['cond_merged.paste.pasted', 'cond_merged.cat.concatenated']
},
'output_target': {
  'merged' : 'somedir_on_s3/somefilename'
}
```

Dependency specification

dependency

- List of other jobs that should finish before the job starts

- Currently, only execution arns are accepted. An execution arn of a given run is printed out after running the `tibanna run_workflow` command. It can also be retrieved from the response of the `run_workflow` function (`response['_tibanna']['exec_arn']`).

```
{
    "exec_arn": ["arn:aws:states:us-east-
→1:643366669028:execution:tibanna_unicorn_default_7927:md5_test"]
}
```

Custom error handling

custom_errors

- List of dictionaries describing custom error types

- This field allows users to define workflow-specific errors based on a string pattern in log. Tibanna CheckTask step will parse the logs and detect this error.

- This does not serve as error detection - it serves as error identification once the run has failed.

- If the matching error happens, you'll see the error type and the corresponding line(s) of the error in the log file printed as the Exception in Step function.

- `error_type` is a short tag that defines the name of the error.

- `pattern` is the regex pattern to be detected in the log.

- `multiline` (optional) should be set True if `pattern` is multi-line (e.g. contains `\n`).

```
[
    {
        "error_type": "Unmatching pairs in fastq"
        "pattern": "paired reads have different names: .+",
        "multiline": False
    }
]
```

config

The `config` field describes execution configuration.

log_bucket

- <log_bucket_name>

- This is where the logs of the Tibanna runs are sent to.

- required

instance_type

- <instance_type>

- This or `mem` and `cpu` are required if Benchmark is not available for a given workflow.

- `instance_type` can be a string (e.g., `t3.micro`) or a list (e.g., `[t3.micro, t3.small]`). If `spot_instance` is enabled, Tibanna will run the workflow on the instance with the highest available capacity. If `spot_instance` is disabled, it will run the workflow on the cheapest instance in the list.

- If both `instance_type` and `mem` & `cpu` are specified, Tibanna internally creates a list of instances that are directly specified in `instance_type` and instances that satisfy the `mem` & `cpu` requirement. One instance is chosen according to the rules above to run the workflow.

mem

- <memory_in_gb>

- required is Benchmark is not available for a given workflow and if `instance_type` is not specified.

- `mem` specifies memory requirement - instance_type is auto-determined based on `mem` and `cpu`.

- Starting version 1.2.0, 1GB is added to mem when choosing an instance type by default. To turn off This automatic increase in memory, set `mem_as_is` to be `true`.

mem_as_is

- <true|false>

- If true, the value set in `mem` is used as it is when choosing an instance type. If false, 1GB is added by default, to accommodate the memory consumption of the house-keeping processes.

- This field is available for >=1.2.0

cpu

- <number_of_cores>

- required is Benchmark is not available for a given workflow and if `instance_type` is not specified.

- `cpu` specifies number of cores required to run a given workflow - instance_type is auto-determined based on `mem` and `cpu`.

ebs_size

- <ebs_size_in_gb>

- The EBS volume size used for data (input, output, or any intermediary files). This volume is mounted as /data1 on the EC2 instance and as /data1 inside Docker image when running in the `shell` or `snakemake` mode.

- 10 is minimum acceptable value.

- set as 10 if not specified and if Benchmark is not available for a given workflow.

- It can be provided in the format of <s>x (e.g. 3x, 5.5x) to request <s> times total input size. (or 10 is smaller than 10)

- Starting version 1.2.0, 5GB is added to ebs_size by default. To turn off This automatic increase in EBS size, set `ebs_size_as_is` to be `true`.

ebs_size_as_is

- <true|false>

- If true, the value set in `ebs_size` is used as it is. If false, 5GB is added by default, to accommodate the disk usage of house-keeping processes and docker image/containers.

- This field is available for >=1.2.0

EBS_optimized

- <ebs_optimized> `true`, `false` or '' (blank)

- required if Benchmark is not available for a given workflow.

- Whether the specific instance type should be EBS_optimized. It can be True only for an instance type that can be EBS optimized. If instance type is unspecified, leave this as blank.

root_ebs_size

- <root_ebs_size_in_gb>

- default 8

- For versions < `1.0.0`, Tibanna uses two separate EBS volumes, one for docker image, another for data. Most of the times, the 8GB root EBS that is used for docker images has enough space. However, if the docker image is larger than 5GB or if multiple large docker images are used together, one may consider increasing root ebs size. Any directory that is used inside a docker image (e.g. `/tmp` when running in the `shell` mode) that is not mounted from the data EBS could also cause a `no space left in device` error on the root EBS volume. It is recommended to use a directory under `/data1` as a temp directory when running in the `shell` mode, which is mounted from data EBS.

- This field is supported in version `0.9.0` or higher. If an older version has been used, redeploy `run_task_awsem` to enable this feature, after installing `0.9.0` or higher, as below.

  ```
  tibanna deploy_core -n run_task_awsem -g <usergroup> [-s <suffix>]
  ```

- For versions >= `1.0.0`, this field is no longer needed (though still supported) since the docker image also uses the data EBS and not the root EBS starting `1.0.0`. This means for a large docker image, it is recommended to increase `ebs_size` rather than `root_ebs_size`. It takes effect only if `run_task_awsem` is redeployed as above. For consistency, when you redeploy `run_task_awsem` from version < `1.0.0` to version >= `1.0.0`, it is also recommended to redeploy `check_task_awsem` with the same version.

shutdown_min

- either number of minutes or string 'now'

- 'now' would make the EC2 instance to terminate immediately after a workflow run. This option saves cost if the pipeline is stable. If debugging may be needed, one could set shutdown_min to be for example, 30, in which case the instance will keep running for 30 minutes after completion of the workflow run. During this time, a user could ssh into the instance.

- optional (default : "now")

password

- <password_for_ssh> or '' (blank)

- One can use either password or key_name (below) as ssh mechanism, if the user wants an option to ssh into the instance manually for monitoring/debugging purpose. Tibanna itself does not use ssh.

- The password can be any string and anyone with the password and the ip address of the EC2 instance can ssh into the machine.

- optional (default : no password-based ssh)

key_name

- <key_pair_name> or '' (blank)

- One can use either password (above) or key_name as ssh mechanism, if the user wants an option to ssh into the instance manually for monitoring/debugging purpose. Tibanna itself does not use ssh.

- The key pair should be an existing key pair and anyone with the key pair `.pem` file and the ip address of the EC2 instance can ssh into the machine.

- optional (default : no key-based ssh)

ebs_iops

- IOPS of the io1, io2 or gp3 type EBS

- optional (default: unset)

ebs_throughput

- Provisioned throughput of the gp3 type EBS (MiB/s). Must be an integer between 125 and 1000.

- optional (default: unset)

ebs_type

- type of EBS (e.g. `gp3`, `gp2`, `io1`, `io2`)

- optional (default: gp3 (version >= `1.0.0`) or gp2 (version < `1.0.0`))

cloudwatch_dashboard

- **This option is now depricated.**

- if true, Memory Used, Disk Used, CPU Utilization Cloudwatch metrics are collected into a single Cloudwatch Dashboard page. (default `false`)

- Warning: very expensive - Do not use it unless absolutely neessary. Cloudwatch metrics are collected for every awsem EC2 instances even if this option is turned off. The Dashboard option makes it easier to look at them together.

- There is a limit of 1,000 CloudWatch Dashboards per account, so do not turn on this option for more than 1,000 runs.

spot_instance

- if true, request spot instance instead of an On-Demand instance

- optional (default `false`)

spot_duration

- Max duration of spot instance in min (no default). If set, request a fixed-duration spot instance instead of a regular spot instance. `spot_instance` must be set `true`.

- optional (no default)

behavior_on_capacity_limit

- behavior when a requested instance type (or spot instance) is not available due to instance limit or unavailability.

- available options :

  - `fail` (default)

  - `wait_and_retry` (wait and retry with the same instance type again.),

  - `retry_without_spot` (try with the same instance type but not a spot instance) : this option is applicable only when `spot_instance` is set to `` `True ``

availability_zone

- specify availability zone (by default, availability zone is randomly selected within region by AWS)

- e.g. `us-east-1a`

- optional (no default)

security_group

- specify security group. This feature may be useful to launch an instance to a specific VPC.

- e.g. `sg-00151073fdf57305f`

- optional (no default)

- This feature is supported in version *0.15.6* or higher. If an older version has been used, redeploy `run_task_awsem` to enable this feature, after installing `0.15.6` or higher, as below.

```
tibanna deploy_core -n run_task_awsem -g <usergroup> [-s <suffix>]
```

subnet

- specify subnet ID. This feature may be useful to launch an instance to a specific VPC. If you don't have default VPC, subnet must be specified.

- e.g. `subnet-efb1b3c4`

- optional (no default)

- This feature is supported in version *0.15.6* or higher. If an older version has been used, redeploy `run_task_awsem` to enable this feature, after installing `0.15.6` or higher, as below.

```
tibanna deploy_core -n run_task_awsem -g <usergroup> [-s <suffix>]
```

1.4.8 Monitoring a workflow run

Monitoring can be done either from the Step Function Console through a Web Browser, or through command-line.

Command-line

General stats

```
tibanna stat [--sfn=<stepfunctioname>] [--status=RUNNING|SUCCEEDED|FAILED|TIMED_
↪OUT|ABORTED] [-l] [-n <number_of_lines>] [-j <job_id> [<job_id2>] [...]]
```

The output is a table (an example below)

```
jobid        status     name     start_time      stop_time
2xPih7reR6FM        RUNNING md5        2018-08-15 17:45        2018-08-15 17:50
3hbkJB3hv92S        SUCCEEDED     hicprocessingbam  2018-08-15 16:04        2018-
↪08-15 16:09
UlkvH3gbBBA2        FAILED   repliseq-parta   2018-08-09 18:26        2018-08-09
↪19:01
j7hvisheBV27        SUCCEEDED       bwa-mem   2018-08-09 18:44        2018-08-09
↪18:59
```

To print out more information, use the `-l` (long) option. The additional information includes the ID, type, status and public ip of the EC2 instance. Keyname and Password information is shown for ssh.

```
jobid        status     name      start_time       stop_time       instance_id       instance_
↪type   instance_status ip      key      password
O37462jD9Kf7        RUNNING bwa-mem 2018-12-14 23:37        2018-12-14 23:40        i-
↪009880382ee22a5b1    t2.large        running 3.25.66.32      4dn-encode     ␣
↪somepassword
```

(continues on next page)

```
jN4ubJNlNKIi          ABORTED bwa-mem 2018-12-14 23:33          2018-12-14 23:36          i-
↪0df66d22d485bbc05    c4.4xlarge      shutting-down   -       -       -
dWBRxy0R8LXi          SUCCEEDED       bwa-mem 2018-12-14 22:44       2018-12-14 22:59 ␣
↪       i-00f222fe5e4580007    t3.medium       terminated      -       -       -
```

Using `-n` limits the number of lines to be printed. (the most recent `n` items will be printed)

Execution logs

Log

Using your job ID, you can also check your S3 bucket to see if you can find a file named <jobid>.log. This will happen
5~10min after you start the process, because it takes time for an instance to be ready and send the log file to S3. The
log file gets updated, so you can re-download this file and check the progress. Checking the log file can be done
through the `tibanna log` command. For example, to view the last 60 lines of the log for job `lSbkdVIQ6VtX`,

```
tibanna log --job-id=lSbkdVIQ6VtX | tail  -60
```

The output looks as below for version `1.0.0` or higher (much better organized / formatted than older version logs).

```
## job id: tvfZLFlt3PBz
## instance type: t3.micro
## instance id: i-0be6e6be5723ecd24
## instance region: us-east-1
## tibanna lambda version: 1.0.0
## awsf image: duplexa/tibanna-awsf:1.0.0
## ami id: ami-0a7ddfc7e412ab6e0
## availability zone: us-east-1f
## security groups: default
## log bucket: my-tibanna-test-bucket
## shutdown min: 30

## Starting...
Tue Nov  3 20:47:19 UTC 2020


...


## Running CWL/WDL/Snakemake/Shell commands

## workflow language: wdl
## Operating System: Ubuntu 20.04.1 LTS (containerized)
## Docker Root Dir: /mnt/data1/docker
## CPUs: 16
## Total Memory: 40.18GiB


...


INFO /usr/local/bin/cwltool 3.0.20201017180608
INFO Resolved 'workflow_gatk-GenotypeGVCFs_plus_vcf-integrity-check.cwl' to 'file:///
↪mnt/data1/cwl/workflow_gatk-GenotypeGVCFs_plus_vcf-integrity-check.cwl'
INFO [workflow ] start
INFO [workflow ] starting step gatk-GenotypeGVCFs
```

```
INFO [step gatk-GenotypeGVCFs] start

...

22:12:34.599 WARN  InbreedingCoeff - Annotation will not be calculated, must provide
→at least 10 samples
22:12:34.599 WARN  InbreedingCoeff - Annotation will not be calculated, must provide
→at least 10 samples
22:12:34.600 WARN  InbreedingCoeff - Annotation will not be calculated, must provide
→at least 10 samples
22:12:34.601 WARN  InbreedingCoeff - Annotation will not be calculated, must provide
→at least 10 samples
22:12:35.852 INFO  ProgressMeter -      chr14:106769920               50.4        ␣
→79043000        1567469.6
22:12:36.890 INFO  ProgressMeter -      chr14:106882957               50.4        ␣
→79071726        1567501.5
22:12:36.890 INFO  ProgressMeter - Traversal complete. Processed 79071726 total␣
→variants in 50.4 minutes.
22:12:36.999 INFO  GenotypeGVCFs - Shutting down engine
[November 3, 2020 10:12:37 PM UTC] org.broadinstitute.hellbender.tools.walkers.
→GenotypeGVCFs done. Elapsed time: 50.48 minutes.
Runtime.totalMemory()=1915224064
Using GATK jar /miniconda3/share/gatk4-4.1.2.0-1/gatk-package-4.1.2.0-local.jar
```

To Download the log file manually, the following command also works.

```
aws s3 cp s3://<tibanna_lob_bucket_name>/<jobid>.log .
```

Top and Top_latest

As of version `1.0.0`, the top command output is sent to `<jobid>.top` and `<jobid>.top_latest` in the log bucket. The top command output used to be mixed in the log file (`<jobid>.log`) in previous versions. With `tibanna log` command and option `-t` (all top output) and `-T` (latest only), one can print out the top command output from the running instance. The data is collected at 1-minute intervals and only while the command is running (e.g. not while the input data are downloaded to the EC2 instance or ssh is being configured etc).

To use this feature, the tibanna unicorn must be deployed with tibanna >= `1.0.0` and the locally installed version must be >= `1.0.0` as well.

Below is an example command and the output, executed twice with a 1-minute interval. In this example, the user can see that around 20:49:01, `unpigz` was running and around 20:50:01, many `java` processes were running (they depend on the command / workflow).

```
tibanna log -j OiHYCN1QoEiP -T
```

```
Timestamp: 2021-01-20-20:49:01
top - 20:49:01 up 1 min,  0 users,  load average: 2.11, 0.75, 0.27
Tasks:  15 total,   2 running,  13 sleeping,   0 stopped,   0 zombie
%Cpu(s): 13.1 us,  6.4 sy,  0.0 ni, 80.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  41139.5 total,  32216.5 free,    675.9 used,   8247.1 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.  39951.0 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
```

```
    54 root      20    0 2928856 102488    48260 S 186.7    0.2    0:44.95 dockerd
   858 root      20    0   28904    1228     1128 R 153.3    0.0    0:09.18 unpigz
   859 root      20    0 1673140   80084    44464 S  46.7    0.2    0:02.91 exe
     1 root      20    0    7104    3692     3348 S   0.0    0.0    0:00.02 run.sh
    94 root      20    0 1781488   45328    25740 S   0.0    0.1    0:00.12 contain+
   319 root      20    0 1792992   14660     9056 S   0.0    0.0    0:00.10 goofys-+
   325 root      20    0 1571284   14136     9080 S   0.0    0.0    0:00.08 goofys-+
   382 root      20    0    6812    2076     1868 S   0.0    0.0    0:00.00 cron
```

If we run the command again in ~1 min, we may get a different snapshot. This way, we can monitor in near-real time what kind of programs are running and how much resources they are using.

```
tibanna log -j OiHYCN1QoEiP -T
```

```
Timestamp: 2021-01-20-20:50:01
top - 20:50:01 up 2 min,  0 users,  load average: 18.06, 4.84, 1.67
Tasks:  45 total,   1 running,  44 sleeping,   0 stopped,   0 zombie
%Cpu(s): 93.6 us,  6.4 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  41139.5 total,  16099.9 free,  16978.6 used,   8061.1 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.  23657.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2085 root      20   0 7984200   1.1g  31356 S 253.3   2.8   0:28.85 java
   2114 root      20   0 7984200   1.2g  31512 S 206.7   2.9   0:25.40 java
   2095 root      20   0 7984200   1.2g  31328 S 186.7   3.0   0:24.46 java
   2208 root      20   0 7984200   1.1g  31356 S 133.3   2.8   0:27.61 java
   2121 root      20   0 7984200   1.2g  31480 S 120.0   2.9   0:26.81 java
   2189 root      20   0 7984200   1.2g  31372 S 120.0   3.0   0:30.18 java
   2122 root      20   0 7984200   1.1g  31232 S 100.0   2.8   0:28.88 java
   2148 root      20   0 7984200   1.0g  31284 S 100.0   2.5   0:29.71 java
```

Postrun.json

Once the job is finished, you should be able to find the `<jobid>.postrun.json` file as well. This file can be viewed likewise using the `tibanna log` command, but with the `-p` option. The postrun json file contains the summary of the run, including the input / output / EC2 configuration and Cloudwatch metrics for memory/CPU/disk space usage.

Starting version `1.0.0`, you can get an incomplete postrun.json before the job is finished, in addition to a complete postrun.json that you get at the end of the run. The incomplete postrun.json will not have the metrics, job status, end time, etc, but will include instance ID and file system.

```
tibanna log -p --job-id=lSbkdVIQ6VtX
```

```
{
    "Job": {
        "status": "0",
        "Metrics": {
            "max_cpu_utilization_percent": 86.4,
            "max_mem_used_MB": 14056.421875,
            "max_mem_utilization_percent": 45.124831006539534,
            "max_disk_space_utilization_percent": 72.0912267060547,
            "total_mem_MB": 31150.08203125,
            "max_mem_available_MB": 17093.66015625,
```

```
            "max_disk_space_used_GB": 64.4835815429688
        },
        "total_tmp_size": "4.0K",
        "Log": {
            "log_bucket_directory": "tibanna-output"
        },
        "App": {
            "main_wdl": "atac.wdl",
            "other_cwl_files": "",
            "App_name": "encode-atacseq-postaln",
            "language": "wdl",
            "other_wdl_files": "",
            "main_cwl": "",
            "cwl_url": "",
            "wdl_url": "https://raw.githubusercontent.com/4dn-dcic/atac-seq-pipeline/
↪master/",
            "App_version": "1.1.1"
        },
        "filesystem": "/dev/nvme1n1",
        "JOBID": "lSbkdVIQ6VtX",
        "instance_id": "i-06fc45b29b47a1703",
        "end_time": "20190204-17:11:01-UTC",
        "total_input_size": "829M",
        "Input": {
            "Input_files_data": {
                "atac.chrsz": {
                    "profile": "",
                    "path": "9866d158-da3c-4d9b-96a9-1d59632eabeb/4DNFIZJB62D1.chrom.
↪sizes",
                    "rename": "",
                    "class": "File",
                    "dir": "elasticbeanstalk-fourfront-webprod-files"
                },
                "atac.blacklist": {
                    "profile": "",
                    "path": "9562ffbd-9f7a-4bd7-9c10-c335137d8966/4DNFIZ1TGJZR.bed.gz
↪",
                    "rename": "",
                    "class": "File",
                    "dir": "elasticbeanstalk-fourfront-webprod-files"
                },
                "atac.tas": {
                    "profile": "",
                    "path": [
                        "b08d0ea3-2d95-4306-813a-f2e956a705a9/4DNFIZYWOA3Y.bed.gz",
                        "0565b17b-4012-4d4d-9914-a4a993717db8/4DNFIZDSO341.bed.gz"
                    ],
                    "rename": [
                        "4DNFIZYWOA3Y.tagAlign.gz",
                        "4DNFIZDSO341.tagAlign.gz"
                    ],
                    "class": "File",
                    "dir": "elasticbeanstalk-fourfront-webprod-wfoutput"
                }
            },
            "Secondary_files_data": {
                "atac.tas": {
```

```
                "profile": "",
                "path": [
                    null,
                    null
                ],
                "rename": [
                    "4DNFIZYWOA3Y.tagAlign.gz",
                    "4DNFIZDSO341.tagAlign.gz"
                ],
                "class": "File",
                "dir": "elasticbeanstalk-fourfront-webprod-wfoutput"
            }
        },
        "Env": {},
        "Input_parameters": {
            "atac.pipeline_type": "atac",
            "atac.paired_end": true,
            "atac.enable_xcor": false,
            "atac.disable_ataqc": true,
            "atac.gensz": "hs"
        }
    },
    "Output": {
        "output_target": {
            "atac.conservative_peak": "b8a245d2-89c3-44d3-886c-4cd895f9d535/
→4DNFICOQGQSK.bb",
            "atac.qc_json": "2296ea28-d09a-41ba-afb9-1cbfafb1898b/atac.qc_
→json16152683435",
            "atac.report": "2296ea28-d09a-41ba-afb9-1cbfafb1898b/atac.
→report34127308390",
            "atac.optimal_peak": "65023676-be5c-4497-927c-a796a4c302fe/
→4DNFIY43X8IO.bb",
            "atac.sig_fc": "166659d9-2d6f-440f-b404-b7fe0109e8c5/4DNFI5BWWMR7.bw"
        },
        "secondary_output_target": {},
        "output_bucket_directory": "elasticbeanstalk-fourfront-webprod-wfoutput",
        "Output files": {
            "atac.conservative_peak": {
                "path": "/data1/wdl/cromwell-executions/atac/14efe06b-a010-42c9-
→be0f-82f33f4d877c/call-reproducibility_overlap/execution/glob-
→c12e49ae1deb87ae04019b575ae1ffe9/conservative_peak.narrowPeak.bb",
                "target": "b8a245d2-89c3-44d3-886c-4cd895f9d535/4DNFICOQGQSK.bb"
            },
            "atac.qc_json": {
                "path": "/data1/wdl/cromwell-executions/atac/14efe06b-a010-42c9-
→be0f-82f33f4d877c/call-qc_report/execution/glob-3440f922973abb7a616aaf203e0db08b/qc.
→json",
                "target": "2296ea28-d09a-41ba-afb9-1cbfafb1898b/atac.qc_
→json16152683435"
            },
            "atac.report": {
                "path": "/data1/wdl/cromwell-executions/atac/14efe06b-a010-42c9-
→be0f-82f33f4d877c/call-qc_report/execution/glob-eae855c82d0f7e2185388856e7b2cc7b/qc.
→html",
                "target": "2296ea28-d09a-41ba-afb9-1cbfafb1898b/atac.
→report34127308390"
            },
```

Chapter 1. What do I need to run pipelines using Tibanna?

```
                "atac.optimal_peak": {
                    "path": "/data1/wdl/cromwell-executions/atac/14efe06b-a010-42c9-
→be0f-82f33f4d877c/call-reproducibility_overlap/execution/glob-
→6150deffcc38df7a1bcd007f08a547cd/optimal_peak.narrowPeak.bb",
                    "target": "65023676-be5c-4497-927c-a796a4c302fe/4DNFIY43X8IO.bb"
                },
                "atac.sig_fc": {
                    "path": "/data1/wdl/cromwell-executions/atac/14efe06b-a010-42c9-
→be0f-82f33f4d877c/call-macs2_pooled/execution/glob-8876d8ced974dc46a0c7a4fac20a3a95/
→4DNFIZYWOA3Y.pooled.fc.signal.bigwig",
                    "target": "166659d9-2d6f-440f-b404-b7fe0109e8c5/4DNFI5BWWMR7.bw"
                }
            },
            "alt_cond_output_argnames": []
        },
        "total_output_size": "232K",
        "start_time": "20190204-15:28:30-UTC"
    },
    "config": {
        "ebs_size": 91,
        "cloudwatch_dashboard": true,
        "ami_id": "ami-0f06a8358d41c4b9c",
        "language": "wdl",
        "json_bucket": "4dn-aws-pipeline-run-json",
        "json_dir": "/tmp/json",
        "EBS_optimized": true,
        "ebs_iops": "",
        "userdata_dir": "/tmp/userdata",
        "shutdown_min": "now",
        "instance_type": "c5.4xlarge",
        "public_postrun_json": true,
        "ebs_type": "gp2",
        "script_url": "https://raw.githubusercontent.com/4dn-dic/tibanna/master/awsf/
→",
        "job_tag": "encode-atacseq-postaln",
        "log_bucket": "tibanna-output"
    },
    "commands": []
}
```

To Download the postrun json file manually, the following command also works.

```
aws s3 cp s3://<tibanna_lob_bucket_name>/<jobid>.postrun.json .
```

EC2 Spot failure detection

From Tibanna version 1.6.0, a cron job on the EC2 will regularly check for Spot Instance interruption notices issued by AWS (in case the workflow on a Spot instance). In such an event, the EC2 spot instance is going to be terminated by AWS and the workflow run will most likely fail. In this case Tibanna creates a file called `<jobid>.spot_failure` in the log bucket.

DEBUG tar ball

For WDL, a more comprehensive log is provided as `<jobid>.debug.tar.gz` in the same log bucket, starting from version 0.5.3. This file is a tar ball created by the following command on the EC2 instance:

```
cd /data1/wdl/
find . -type f -name 'stdout' -or -name 'stderr' -or -name 'script' -or \
-name '*.qc' -or -name '*.txt' -or -name '*.log' -or -name '*.png' -or -name '*.pdf' \
| xargs tar -zcvf debug.tar.gz
```

You can download this file using a `aws s3 cp` command.

```
aws s3 cp s3://<tibanna_lob_bucket_name>/<jobid>.debug.tar.gz .
```

Detailed monitoring through ssh

You can also ssh into your running instance to check more details. The IP of the instance can be found using `tibanna stat -v`

```
ssh ubuntu@<ip>
```

if `keyname` was provided in the input execution json,

```
ssh -i <keyfilename>.pem ubuntu@<ip>
```

The keyname (and/or password) can also be found using `tibanna stat -v`.

Alternatively, the Step Function execution page of AWS Web Console contains details of the ssh options. `keyname` and `password` can be found inside the input json of the execution. The IP can be found inside the output json of the `RunTaskAwsem` step or the input json of the `CheckTaskAwsem` step.

The purpose of the ssh is to monitor things, so refrain from doing various things there, which could interfere with the run. It is recommended, unless you're a developer, to use the log file than ssh.

The instance may be set to run for some time after the run finishes, to allow debugging time with the ssh option. This parameter (in minutes) can be set in the `shutdown_min` field inside the `config` field of the input execution json.

On the instance, one can check the following, for example.

For CWL,

- `/data1/input/` : input files
- `/data1/tmp*` : temp/intermediate files (need sudo access)
- `/data1/output/` : output files (need sudo access)
- `top` : to see what processes are running and how much cpu/memory is being used
- `ps -fe` : to see what processes are running, in more detail

For WDL,
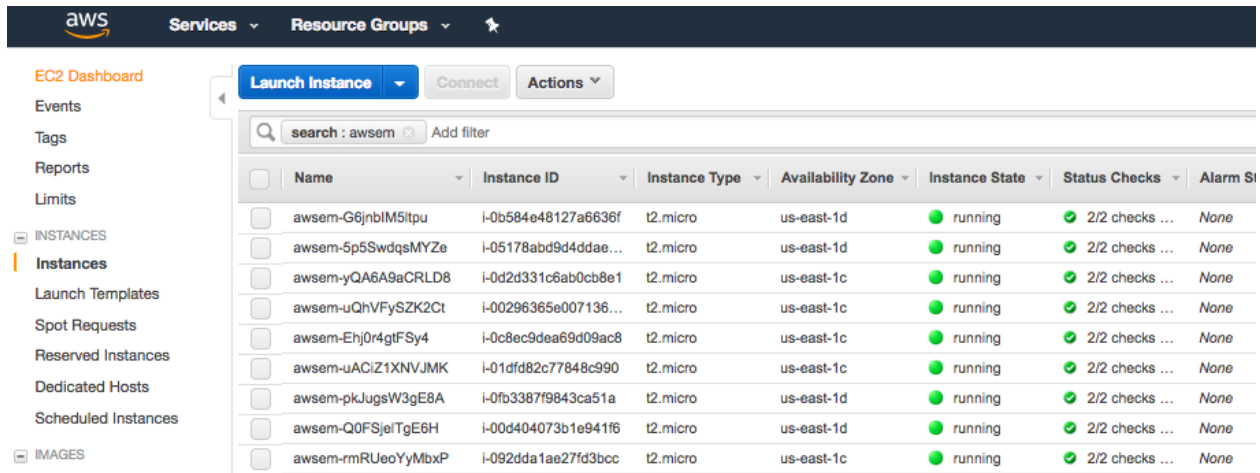
- `/data1/input/` : input files
- `/data1/wdl/cromwell-execution/*` : temp/intermediate files, output files and logs
- `top` : to see what processes are running and how much cpu/memory is being used
- `ps -fe` : to see what processes are running, in more detail

Console

EC2 instances

You can also check from the Console the instance that is running which has a name awsem-<jobid>. It will terminate itself when the run finishes. You won't have access to terminate this or any other instance, but if something is hanging for too long, please contact the admin to resolve the issue.

Step functions

When the run finishes successfully, you'll see in your bucket a file <jobid>.success. If there was an error, you will see a file <jobid>.error instead. The step functions will look green on every step, if the run was successful. If one of the steps is red, it means it failed at that step.
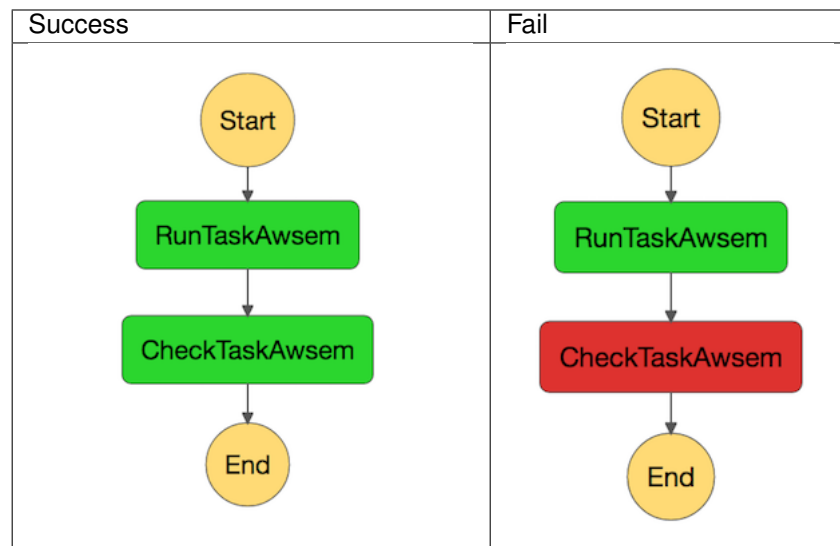
Cloud Watch

Cloudwatch dashboard option is now disabled and replaced by the resource metric report that is generated by the `plot_metrics` command (below `Resource Metrics Report` section).

Resource Metrics Report

Tibanna can collect Cloud Watch metrics on used resources in real time for each run. The metrics are saved as tsv files together with an html report automatically created for visualization. The metrics are collected by 1 minute interval or 5 minute interval depending on the availability on Cloud Watch. The metrics and html files created are uploaded to an S3 bucket.

plot_metrics

This command allows to save Cloud Watch data collected in the required time interval and creates an html report for the visualization.

By default the command will retrieve the data from cloud watch, and creates several files:

- a metrics.tsv file containing all the data points

- a metrics_report.tsv containing the average statistics and other information about the EC2 instance

- a metrics.html report for visualization

All the files are eventually uploaded to a folder named `<jobid>.metrics` inside the log S3 bucket specified for tibanna output. To visualize the html report the URL structure is: `https://<log-bucket>.s3.amazonaws.com/<jobid>.metrics/metrics.html`

Starting with `1.0.0`, the metrics plot will include per-process CPU and memory profiles retrieved from the top command reports at a 1-minute interval. Additional files *top_cpu.tsv* and *top_mem.tsv* will also be created under the same folder `<jobid>.metrics`.

From version `2.1.0`, the metrics that are send to cloud watch are defined here: `https://raw.githubusercontent.com/4dn-dcic/tibanna/master/awsf3/cloudwatch_agent_config.json`. Note that not every metric that is available in cloud watch is displayed in the report created by `plot_metrics`.

Basic Command

```
tibanna plot_metrics --job-id=<jobid> [<options>]
```

Options

```
-s|--sfn=<stepfunctionname>         An example step function name may be
                                    'tibanna_unicorn_defaut_3978'. If not specified,
↪default
                                    value is taken from environmental variable
                                    TIBANNA_DEFAULT_STEP_FUNCTION_NAME. If the
↪environmental
                                    variable is not set, it uses name 'tibanna_pony'
↪(4dn
                                    default, works only for 4dn).

-f|--force-upload                   Upload the metrics reports to the S3 bucket even
                                    if there is a lock file (upload is blocked by
↪default
                                    by the lock)

-u|--update-html-only               Update only the html file for metrics
↪visualization
```

(continues on next page)

```
-B|--do-not-open-browser               Do not open the browser to visualize the metrics␣
↪html
                                       after it has been created/updated

-e|--endtime=<end_time>                Endtime (default job end time if the job has␣
↪finished
                                       or the current time)
-i|--instance-id=<instance_id>         Manually provide instance_id if somehow tibanna␣
↪fails
                                       to retrieve the info
```

When metrics are collected for a run that is complete, a lock file is automatically created inside the same folder. The command will not update the metrics files if a lock file is present. To override this behavior the `--force-upload` flag allows to upload the metrics files ignoring the lock. The `--update-html-only` allows to only update the metrics.html file without modifying the other tsv files. By default the command will open the html report in the browser for visualization when execution is complete, `--do-not-open-browser` can be added to prevent this behavior.

Summary metrics collected as a table

Some summary metrics are collected and shown in the table of at the beginning of the metrics report. They are:

- EC2 Instance type

- Memory, Disk, and CPU utilization as a percentage of the maximum resources available for the EC2 instance
- Memory used in Mb
- Memory available in Mb
- Disk used in Gb

- Start time, end time, and total elapsed time

html report example

Tibanna Metrics

General Information	
EC2 Instance Type	c5n.18xlarge

Metrics	
Maximum Memory Used [Mb]	139116.3046875
Minimum Memory Available [Mb]	50030.13671875
Maximum Disk Used (/data1) [Gb]	64.5787124633789
Maximum Memory Utilization [%]	73.54952260968265
Maximum CPU Utilization [%]	100.0
Maximum Disk Utilization (/data1) [%]	56.8077005677776
Cost	---

Start Time [UTC]	End Time [UTC]	Total Time
2021-01-20 15:43:26	2021-01-20 17:57:59	2:14:33

Resources Utilization

Memory Usage

Memory Available

Disk Usage (/data1)
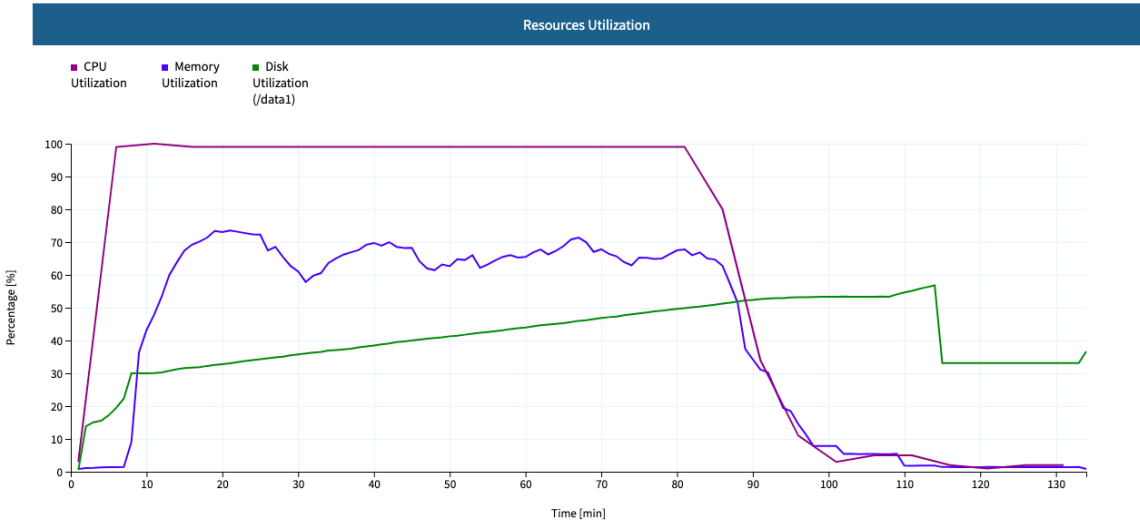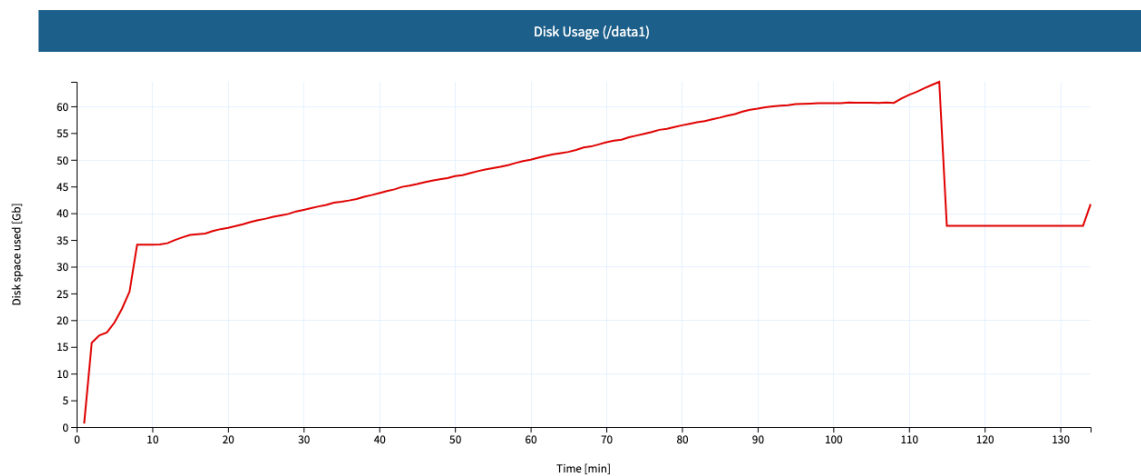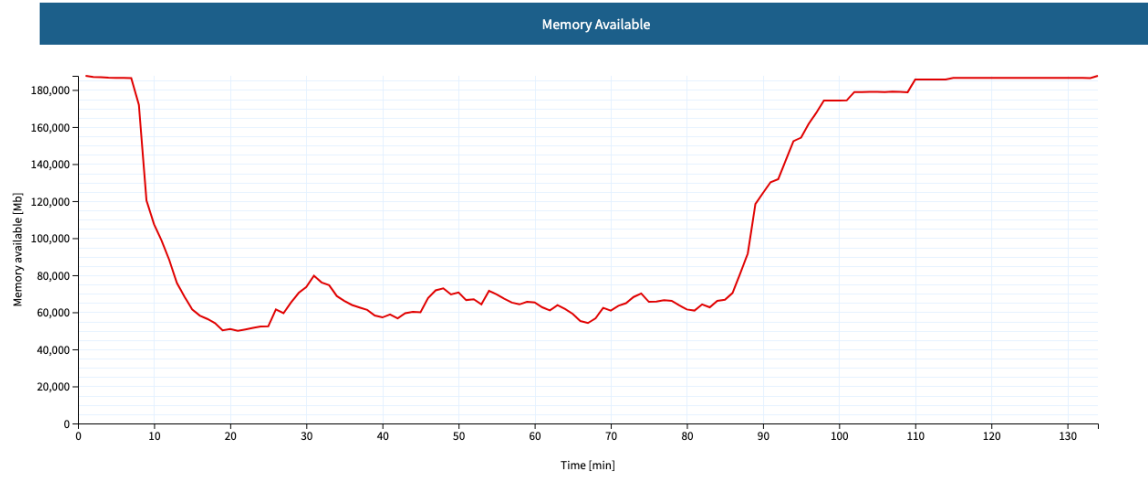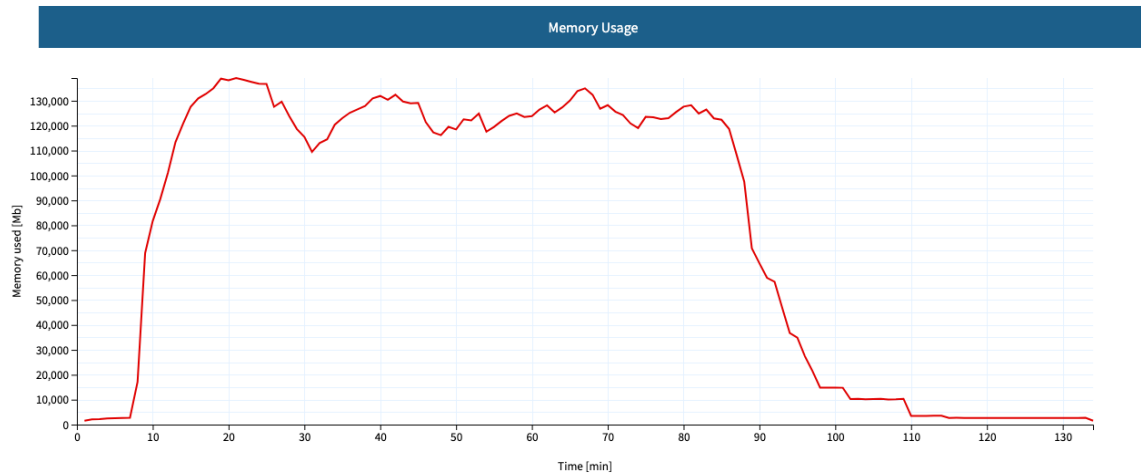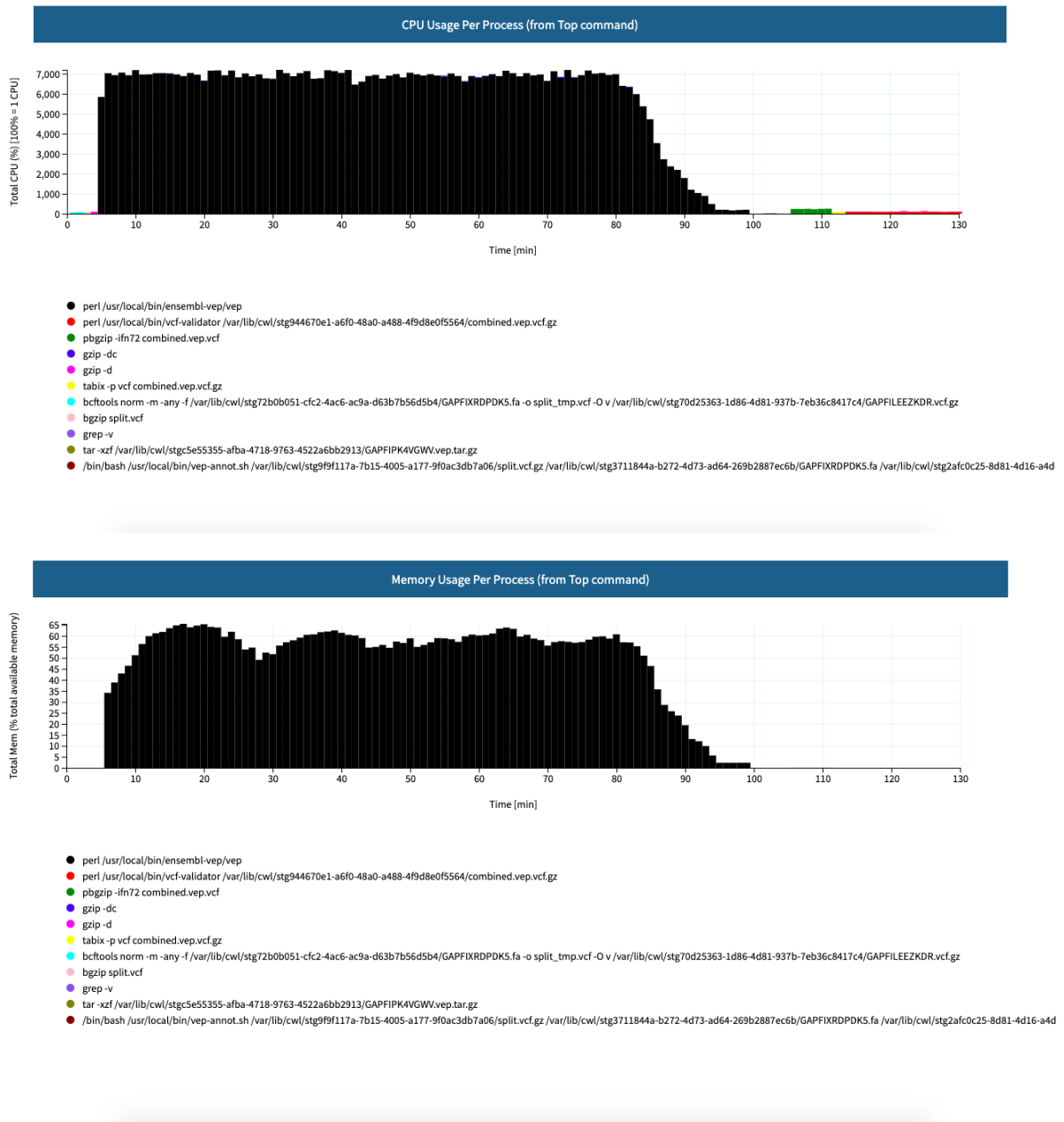
cost

This command allows to retrieve the cost for the run. The cost is not immediately ready and usually requires few days to become available. The command eventually allows to update the information obtained with plot_metrics by adding the cost.

```
tibanna cost --job-id=<jobid> [<options>]
```

Options

```
-s|--sfn=<stepfunctionname>          An example step function name may be
                                     'tibanna_unicorn_defaut_3978'. If not specified,␣
↪default
                                     value is taken from environmental variable
                                     TIBANNA_DEFAULT_STEP_FUNCTION_NAME. If the␣
↪environmental
                                     variable is not set, it uses name 'tibanna_pony'␣
↪(4dn
                                     default, works only for 4dn).

-u|--update-tsv                      Update with the cost the tsv file that stores␣
↪metrics
                                     information on the S3 bucket
```

1.4.9 Common Workflow Language (CWL)

Tibanna supports CWL version 1.0 (https://www.commonwl.org/). Starting with Tibanna version `1.0.0`, CWL draft-3 is no longer supported.

1.4.10 Workflow Description Language (WDL)

Tibanna version < `1.0.0` supports WDL draft-2, through Cromwell binary version 35. Tibanna version >= `1.0.0` supports both WDL draft-2 and v1.0, through Cromwell binary version 35 and 53, respectively. This is because some of our old WDL pipelines written in draft-2 version no longer works with the new Cromwell version and we wanted to ensure the backward compatibility. But if you want to use WDL draft-2, specify `"language": "wdl_draft2"` instead of `"language": "wdl"` which defaults to WDL v1.0.

Tibanna version >= `1.7.0` supports (Caper) in addition to Cromwell. If you would like to use Caper, add `"workflow_engine": "caper"` to the Tibanna job description. Cromwell is the default. If you want to pass additional parameters to Cromwell or Caper, you can specify `run_args` in the Tibanna job description, e.g., `"run_args": "--docker"` will add the docker flag to the Cromwell/Caper `run` command.

1.4.11 Snakemake

Tibanna supports Snakemake pipelines through the snakemake interface (`snakemake --tibanna`). Check out the Snakemake documentation for more details.

1.4.12 Amazon Machine Image

Tibanna now uses the Amazon Machine Images (AMI) `ami-06e2266f85063aabc (x86)` and `ami-0f3e90ad8e76c7a32 (Arm)`, which are made public for `us-east-1`. One can find them among Community AMIs. (Tibanna automatically finds and uses them, so no need to worry about it.)

For regions that are not `us-east-1`, copies of these AMIs are publicly available (different AMI IDs) and are auto-detected by Tibanna.

If you want to use your own AMI, you can overwrite the default values in the `config` field of the Job Description JSON:

```
{
  "args": {
    ...
```

(continues on next page)

```
  },
  "config": {
    ...
    "ami_per_region": {
      "x86": {
        "us-east-1": "my_x86_ami_ue1",
        "us-east-2": "my_x86_ami_ue2",
        ...
      },
      "Arm": {
        "us-east-1": "my_arm_ami_ue1",
        "us-east-2": "my_arm_ami_ue2",
        ...
      }
    },
  }
}
```

1.4.13 Running 4DN pipelines using Tibanna

- For 4DN pipelines, benchmark functions are pre-implemented in Tibanna through the Benchmark package. This means that the user does not have to choose EC2 instance type or EBS size (they are auto-determined). However, if the user wants to specify them, the following fields can be used. `EBS_optimized` makes IO slightly faster, but it is not supported by all isntance types. If you're not sure, choose `false`.

Example

```
"config": {
  "instance_type": "instance_type",
  "ebs_size": 10,
  "EBS_optimized": false,
```

General Quality Control

md5

- Description : calculates two md5sum values (one the file itself, one for ungzipped) for an input file. If the input file is not gzipped, it reports only the first one.

- CWL : https://github.com/4dn-dcic/pipelines-cwl/blob/0.2.6/cwl_awsem_v1/md5.cwl

- Docker : `duplexa/md5:v2`

- 4DN workflow metadata : https://data.4dnucleome.org/workflows/c77a117b-9a58-477e-aaa5-291a109a99f6/

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/75ce5f66-f98f-4222-9d1c-3daed262856b/#graph

- Example input execution json template :

Use the following as a template and replace <YOUR....> with your input/output/log bucket/file(object) information.

```
{
  "args": {
    "app_name": "md5",
    "app_version": "0.2.6",
    "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/pipelines-cwl/0.
→2.6/cwl_awsem_v1/",
    "cwl_version": "v1",
    "cwl_main_filename": "md5.cwl",
    "input_files": {
      "input_file": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_INPUT_FILE_NAME_IN_INPUT_BUCKET>"
      }
    },
    "output_S3_bucket": "<YOUR_OUTPUT_BUCKET>",
    "output_target": {
      "report": "<YOUR_OUTPUT_FILE_NAME_IN_OUTPUT_BUCKET>"
    }
  },
  "config": {
    "log_bucket": "<YOUR_LOG_BUCKET>",
    "key_name": "<YOUR_KEY_NAME>"
  }
}
```

fastqc

- Description : run fastqc on a fastq file

- CWL : https://github.com/4dn-dcic/pipelines-cwl/blob/0.2.6/cwl_awsem_v1/fastqc-0-11-4-1.cwl

- Docker : duplexa/4dn-hic:v32

- 4DN workflow metadata : https://data.4dnucleome.org/workflows/2324ad76-ff37-4157-8bcc-3ce72b7dace9/

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/be8edc0a-f74a-4fae-858e-2915af283ee3/
  #details

- Example input execution json template :

Use the following as a template and replace <YOUR....> with your input/output/log bucket/file(object) information.

```
{
    "args" {
      "app_name": "fastqc-0-11-4-1",
      "app_version": "0.2.0",
      "cwl_version": "v1",
      "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/pipelines-cwl/
→0.2.6/cwl_awsem_v1/",
```

<div align="right">(continues on next page)</div>

```
        "cwl_main_filename": "fastqc-0-11-4-1.cwl",
        "cwl_child_filenames": ["fastqc-0-11-4.cwl"],
        "input_files": {
            "input_fastq": {
                "bucket_name": "<YOUR_INPUT_BUCKET>",
                "object_key": "<YOUR_INPUT_FILE>"
            }
        },
        "output_S3_bucket": "<YOUR_OUTPUT_BUCKET>",
        "output_target": {
            "report_zip": "<YOUR_OUTPUT_REPORT_NAME>.zip"
        }
    },
    "config": {
      "log_bucket" : "<YOUR_LOG_BUCKET>",
      "key_name": "<YOUR_KEY_NAME>"
    }
}
```

Hi-C data processing & QC

- Example full pipeline run
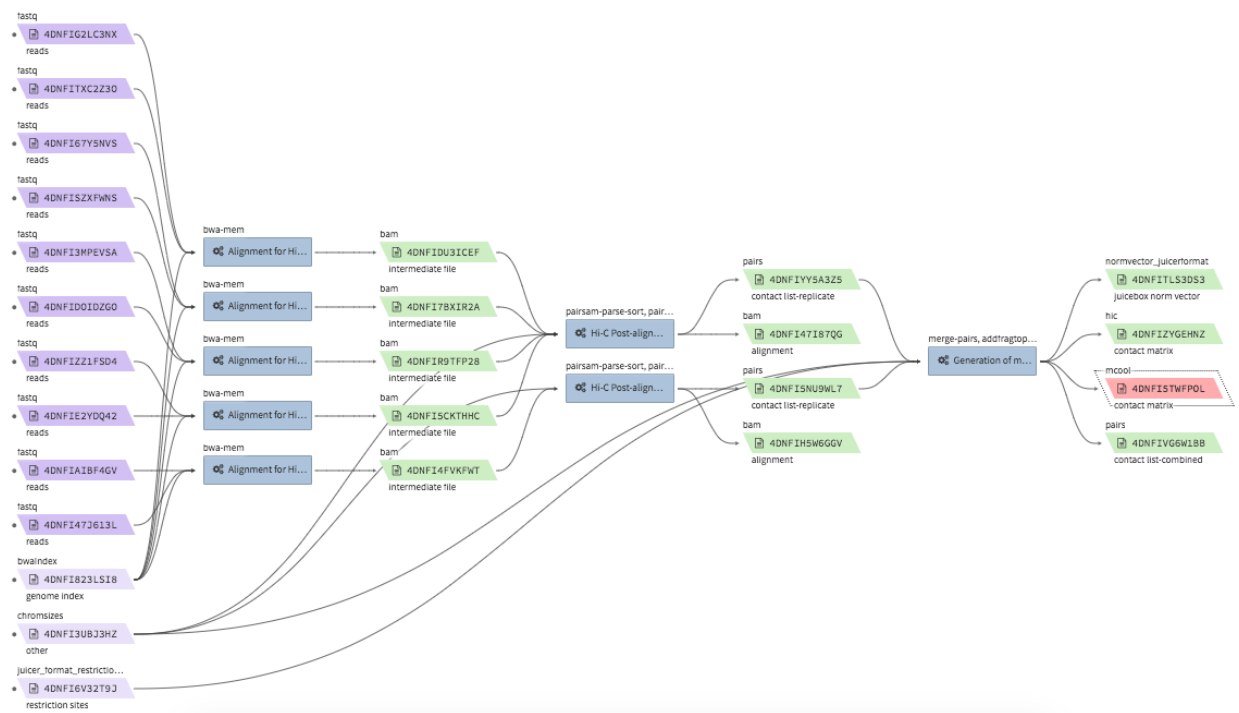
bwa-mem

- Description : aligns Hi-C fastq files to a reference genome using `bwa mem -SP5M`. The output is a single bam file. The bam file is not resorted, and does not accompany a `.bai` index file. The bwa reference genome index must be bundled in a `.tgz` file.

- CWL : https://github.com/4dn-dcic/pipelines-cwl/blob/0.2.6/cwl_awsem_v1/bwa-mem.cwl

- Docker : `duplexa/4dn-hic:v42.2`

- 4DN workflow metadata : https://data.4dnucleome.org/workflows/3feedadc-50f9-4bb4-919b-09a8b731d0cc/

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/14fd752d-ede1-4cc2-bb69-6fae5726e173/

- 4DN reference files: https://data.4dnucleome.org/search/?file_format.file_format=bwaIndex&file_type=genome+index&type=FileReference
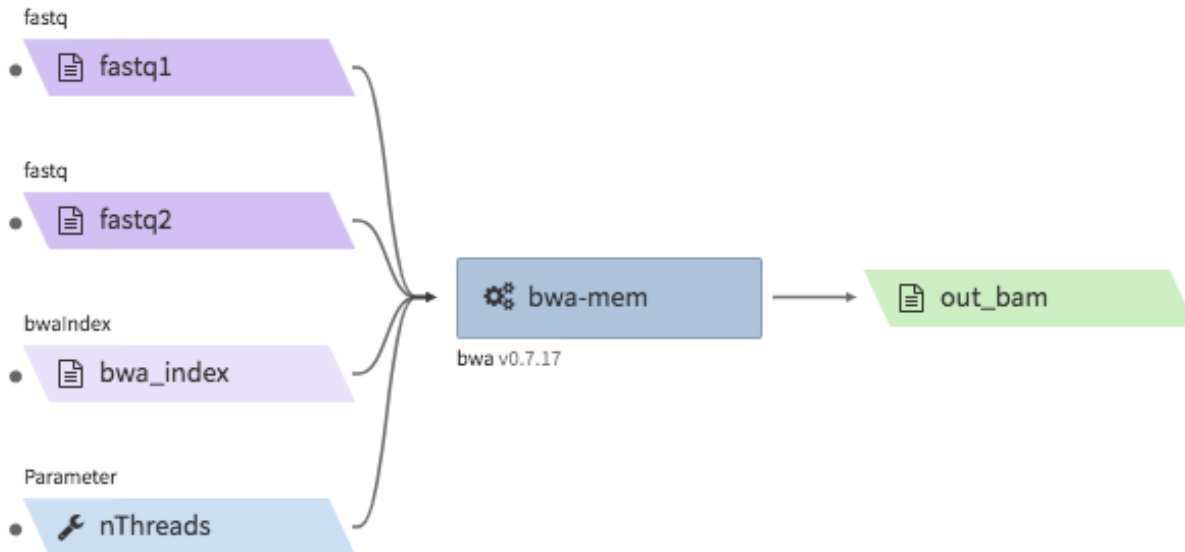
- Example input execution json template :

Use the following as a template and replace <YOUR....> with your input/output/log bucket/file(object) information.

```
{
  "args": {
    "app_name": "bwa-mem",
    "app_version": "0.2.6",
    "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/pipelines-cwl/0.
→2.6/cwl_awsem_v1/",
    "cwl_main_filename": "bwa-mem.cwl",
    "cwl_version": "v1",
    "input_files": {
      "fastq1": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_FASTQ_FILE_R1>"
      },
      "fastq2": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_FASTQ_FILE_R2>"
      },
      "bwa_index": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_TGZ_BWA_INDEX_FILE>"
      }
    },
    "input_parameters": {
      "nThreads": 2
```

(continues on next page)

```
    },
    "output_S3_bucket": "<YOUR_OUTPUT_BUCKET>",
    "output_target": {
      "out_bam": "<YOUR_OUTPUT_BAM_FILE>.bam"
    }
  },
  "config": {
    "log_bucket": "<YOUR_LOG_BUCKET>",
    "key_name": "<YOUR_KEY_NAME>"
  }
}
```

hi-c-processing-bam

- Description : takes in a set of bam files and performs merging, sorting, filtering and produces a `.pairs.gz`
  file (and a `.pairs.gz.px2` index file). The output includes a merged and filter-annotated lossless bam file.

- CWL : https://github.com/4dn-dcic/pipelines-cwl/blob/0.2.6/cwl_awsem_v1/hi-c-processing-bam..cwl

- Docker : `duplexa/4dn-hic:v42.2`

- 4DN workflow metadata : https://data.4dnucleome.org/workflows/023bfb3e-9a8b-42b9-a9d4-216079526f68/

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/b13b2ab8-f176-422f-a1eb-ed213ac991af/

- 4DN reference files:

    - chromsizes files : e.g.) https://data.4dnucleome.org/files-reference/4DNFI823LSII (GRCh38, main chro-
      mosomes only)

    - restriction site files : https://data.4dnucleome.org/search/?file_type=restriction+sites&type=FileReference
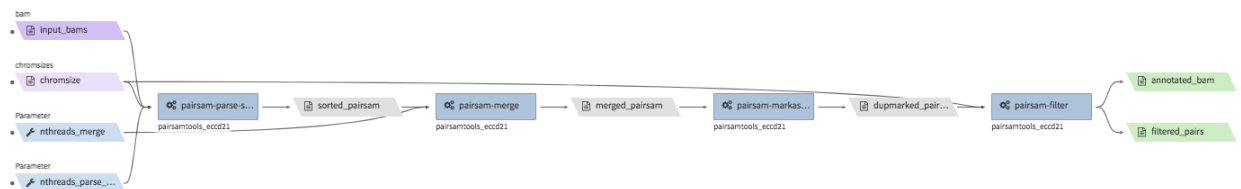
- Example input execution json template :

Use the following as a template and replace `<YOUR....>` with your input/output/log bucket/file(object) information.

```
{
  "args": {
    "app_name": "hi-c-processing-bam",
    "app_version": "0.2.6",
    "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/pipelines-cwl/0.
↪2.6/cwl_awsem_v1/",
    "cwl_main_filename": "hi-c-processing-bam.cwl",
    "cwl_child_filenames": [
      "pairsam-parse-sort.cwl",
      "pairsam-merge.cwl",
      "pairsam-markasdup.cwl",
      "pairsam-filter.cwl",
      "addfragtopairs.cwl"
    ],
```

```
    "input_files": {
      "chromsize": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_INPUT_CHROMSIZES_FILE>"
      },
      "input_bams": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": [
          "<YOUR_BAM_FILE1>",
          "<YOUT_BAM_FILE2>",
          "<YOUT_BAM_FILE3>"
        ]
      },
      "restriction_file": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_RESTRICTION_SITE_FILE>"
      }
    },
    "input_parameters": {
      "nthreads_parse_sort": 8,
      "nthreads_merge": 8
    },
    "output_S3_bucket": "<YOUR_OUTPUT_BUCKET>",
    "output_target": {
      "out_pairs": "<YOUR_OUTPUT_PAIRS_FILE>.pairs.gz",
      "merged_annotated_bam": "<YOUR_OUTPUT_MERGED_BAM_FILE>.bam"
    },
    "secondary_output_target": {
      "out_pairs": "<YOUR_OUTPUT_PAIRS_FILE>.pairs.gz.px2"
    }
  },
  "config": {
    "log_bucket": "<YOUR_LOG_BUCKET>",
    "key_name": "<YOUR_KEY_NAME>"
  }
}
```
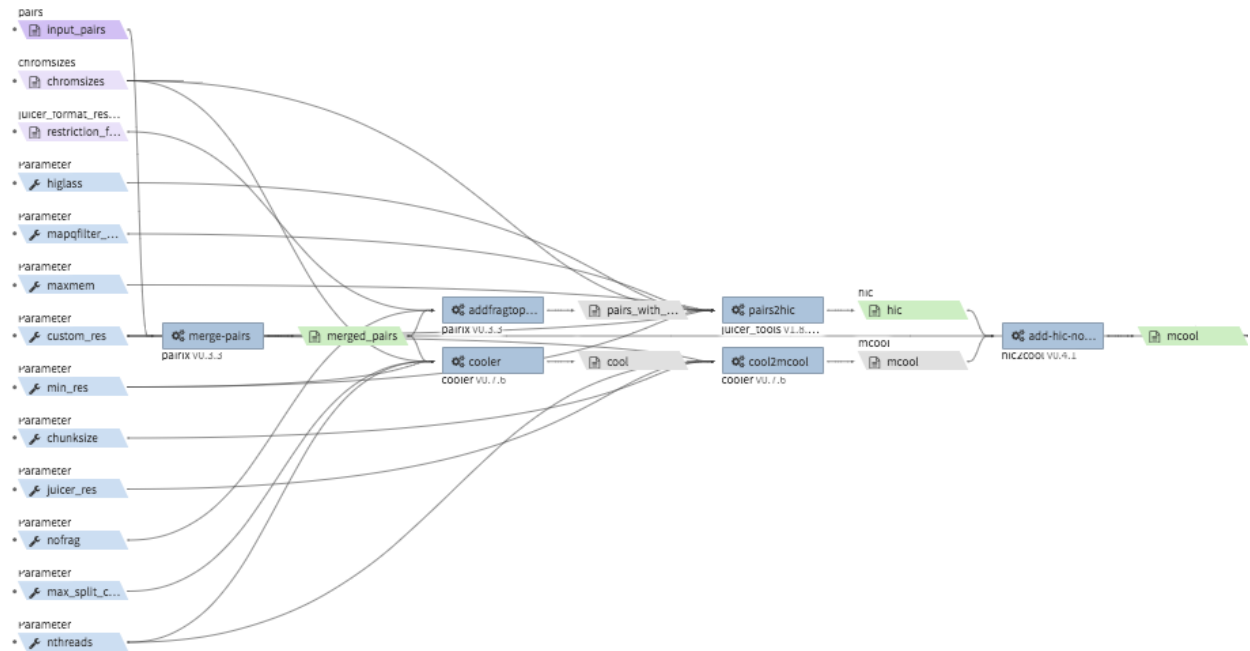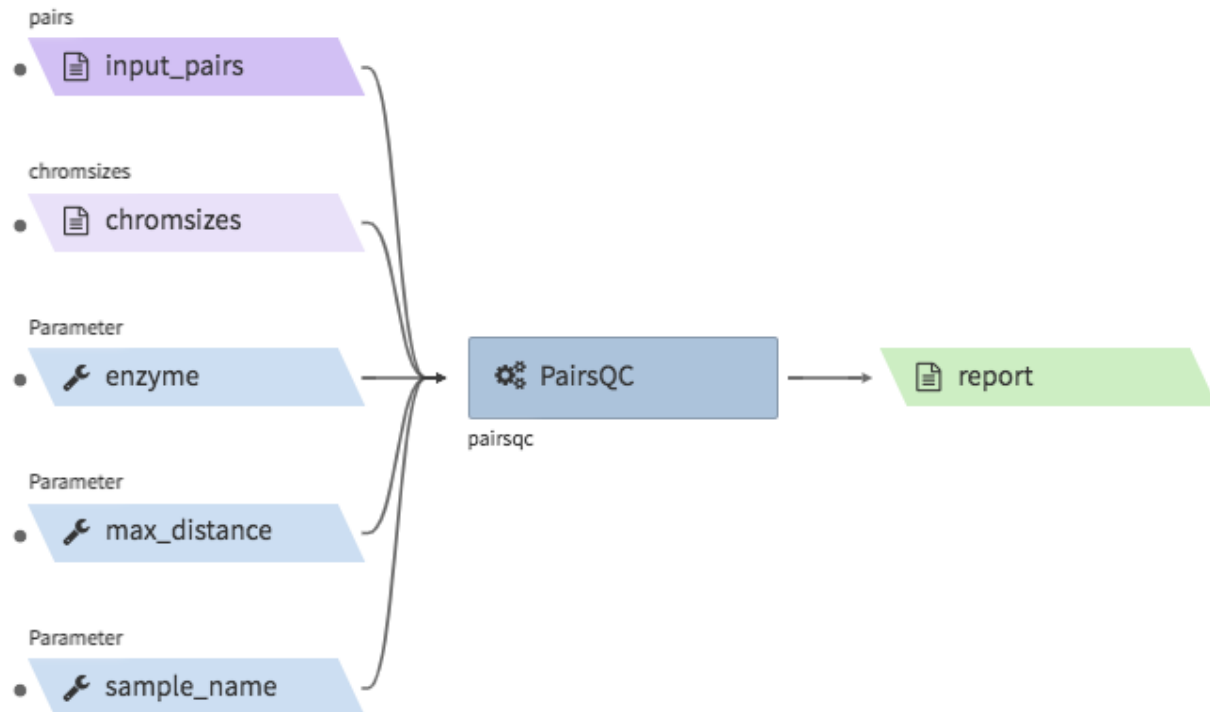
hi-c-processing-pairs

- Description : takes in a set of pairs files, merges them and creates contact matrix files in both `.mcool` and `.hic` formats. The output includes a merged pairs file.

- CWL : https://github.com/4dn-dcic/pipelines-cwl/blob/0.2.6/cwl_awsem_v1/hi-c-processing-pairs.cwl

- Docker : `duplexa/4dn-hic:v42.2`

- 4DN workflow metadata : https://data.4dnucleome.org/workflows/c9e0e6f7-b0ed-4a42-9466-cadc2dd84df0/

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/c0e0da16-a2f9-4e87-a3b2-8f6b4c675a52/

- Example input execution json template :

Use the following as a template and replace `<YOUR....>` with your input/output/log bucket/file(object) information.

```
{
  "args": {
    "app_name": "hi-c-processing-pairs",
    "app_version": "0.2.6",
    "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/pipelines-cwl/0.
    →2.6/cwl_awsem_v1/",
    "cwl_main_filename": "hi-c-processing-pairs.cwl",
    "cwl_child_filenames": [
      "merge-pairs.cwl",
      "addfragtopairs.cwl",
      "pairs2hic.cwl",
      "cooler.cwl",
      "cool2mcool.cwl",
      "extract-mcool-normvector-for-juicebox.cwl",
      "add-hic-normvector-to-mcool.cwl"
    ],
    "cwl_version": "v1",
    "input_files": {
      "chromsizes": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_INPUT_CHROMSIZES_FILE>"
      },
      "input_pairs": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": [
            "<YOUR_INPUT_PAIRS_FILE1>",
            "<YOUR_INPUT_PAIRS_FILE2>",
            "<YOUR_INPUT_PAIRS_FILE3>"
        ]
      },
      "restriction_file": {
```

(continues on next page)

```
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": "<YOUR_RESTRICTION_SITE_FILE>"
      }
    },
    "input_parameters": {
      "ncores": 1,
      "maxmem": "8g"
    },
    "output_S3_bucket": "<YOUR_OUTPUT_BUCKET>",
    "output_target": {
      "mcool": "<YOUR_OUTPUT_MULTIRES_COOL_FILE>.mcool",
      "merged_pairs": "<YOUR_OUTPUT_MERGED_PAIRS_FILE>.pairs.gz",
      "hic": "<YOUR_OUTPUT_HIC_FILE>.hic"
    },
    "secondary_output_target": {
      "output_pairs": "<YOUR_OUTPUT_MERGED_PAIRS_FILE>.pairs.gz.px2"
    }
  },
  "config": {
    "log_bucket": "<YOUR_LOG_BUCKET>",
    "key_name": "<YOUR_KEY_NAME>"
  }
}
```

pairsqc

- Description : calculated QC stats for a pairs file and generates a report zip file containing an `.html` file and other table files.

- CWL : https://github.com/4dn-dcic/pipelines-cwl/blob/0.2.6/cwl_awsem_v1/pairsqc-single.cwl

- Docker : `duplexa/4dn-hic:v42.2`

- 4DN workflow metadata : https://data.4dnucleome.org/workflows/b8c533e0-f8c0-4510-b4a1-ac35158e27c3/

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/902f34fa-dff9-4f26-9af5-64b39b13a069/

- Example input execution json template :

Use the following as a template and replace <YOUR....> with your input/output/log bucket/file(object) information.

```
{
    "args": {
      "app_name": "pairsqc-single",
      "app_version": "0.2.6",
      "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/pipelines-cwl/
→0.2.6/cwl_awsem_v1/",
      "cwl_main_filename": "pairsqc-single.cwl",
      "cwl_version": "v1",
      "input_files": {
        "input_pairs" : {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": "<YOUR_PAIRS_FILE>"
        },
        "chromsizes" : {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": "<YOUR_INPUT_CHROMSIZES_FILE>"
        }
      },
      "secondary_files": {
        "input_pairs": {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": "<YOUR_PAIRS_FILE>.px2"
        }
      },
      "input_parameters" :  {
        "enzyme": "6",
        "sample_name": "4DNFI1ZLO9D7",
        "max_distance": 8.2
```

(continues on next page)

```
    },
    "output_S3_bucket": "<YOUR_OUTPUT_BUCKET>",
    "output_target": {
      "report": "<YOUR_OUTPUT_REPORT_FILE>.zip"
    }
  },
  "config": {
    "log_bucket": "<YOUR_LOG_BUCKET>",
    "key_name": "<YOUR_KEY_NAME>"
  }
}
```
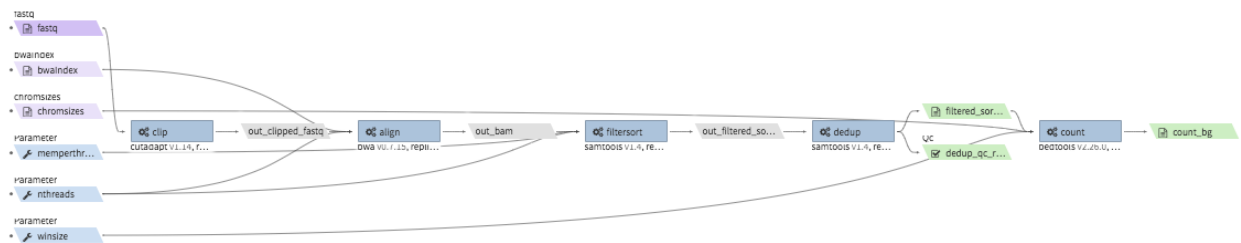
Repli-seq data processing & QC

repliseq-parta

- Description : takes in repli-seq single-end fastq file and performs alignment, sorting, filtering and produces a bedgraph file containing read counts per bin.

- CWL : https://raw.githubusercontent.com/4dn-dcic/docker-4dn-repliseq/v14/cwl/repliseq-parta.cwl

- Docker : `duplexa/4dn-repliseq:v14`

- 4DN workflow metadata : https://data.4dnucleome.org/workflows/4459a4d8-1bd8-4b6a-b2cc-2506f4270a34/

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/66e76f78-0495-4a2a-abfc-2d494d724ded/

- Example input execution json template :

Use the following as a template and replace <YOUR....> with your input/output/log bucket/file(object) information.

```
{
  "args": {
    "app_name": "repliseq-parta",
    "app_version": "v14",
    "cwl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/docker-4dn-
→repliseq/v14/cwl/"
    "cwl_main_filename": "repliseq-parta.cwl",
    "cwl_child_filenames": ["clip.cwl","align.cwl","filtersort.cwl","dedup.cwl",
→"count.cwl"],
    "cwl_version": "v1",
    "input_files": {
      "fastq": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_INPUT_FASTQ>"
      },
      "bwaIndex": {
```

```
                "bucket_name": "<YOUR_INPUT_BUCKET>",
                "object_key": "<YOUR_INPUT_TGZ_BWA_INDEX>"
            },
            "chromsizes": {
                "bucket_name": "<YOUR_INPUT_BUCKET>",
                "object_key": "<YOUR_CHROMSIZES_FILE>"
            }
        },
        "input_parameters": { "nthreads": 8 },
        "output_S3_bucket": "<YOUR_OUTPUT_BUCKET>",
        "output_target": {
            "filtered_sorted_deduped_bam": "<YOUR_OUTPUT_FILTERED_BAM>.bam",
            "dedup_qc_report": "<YOUR_QC_REPORT>.zip",
            "count_bg": "<YOUR_OUTPUT_COUNT_BEDGRAPH_FILE>.bg"
        }
    },
    "config": {
        "log_bucket": "<YOUR_LOG_BUCKET>",
        "key_name": "<YOUR_KEY_NAME>"
    }
}
```
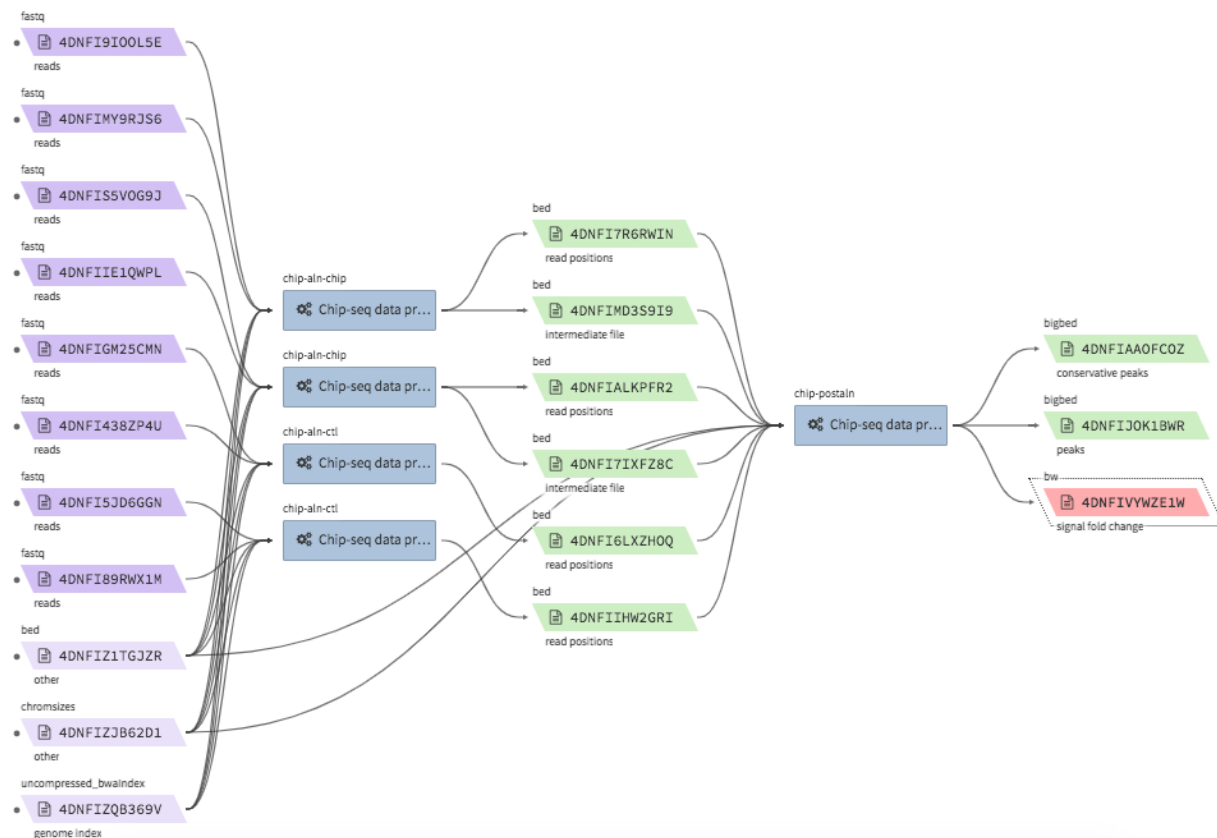
ChIP-seq data processing & QC

- Example full pipeline run

encode-chipseq-aln-chip

- Description : takes in fastq files from a single biological replicate (may consist of multiple technical replicates) and generates a TagAlign file for that biological replicate. The output includes another TagAlign file exclusively for xcor analysis in the next step (`encode-chipseq-postaln`).

- WDL : https://github.com/4dn-dcic/chip-seq-pipeline2/blob/4dn-v1.1.1/chip.wdl

- Docker : `4dndcic/encode-chipseq:v1.1.1`

- 4DN workflow metadata : https://data.4dnucleome.org/4dn-dcic-lab:wf-encode-chipseq-aln-chip

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/3e0fc011-5e84-476e-93a7-176d4ce718c6/
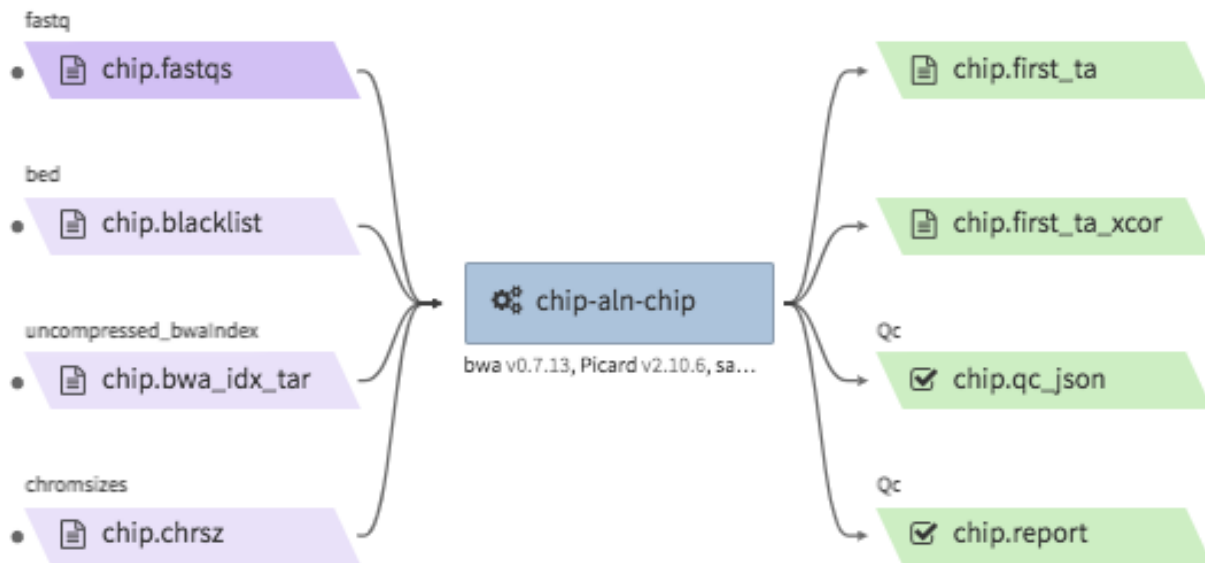
- Example input execution json template :

Use the following as a template and replace <YOUR....> with your input/output/log bucket/file(object) information.

```
{
  "args": {
    "app_name": "encode-chipseq-aln-chip",
    "app_version": "v1.1.1",
    "wdl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/chip-seq-
→pipeline2/4dn-v1.1.1/",
    "wdl_main_filename": "chip.wdl",
    "language": "wdl",
    "input_files": {
      "chip.fastqs": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": [[
            ["<YOUR_INPUT_FASTQ_R1_TECHREP1>.fastq.gz", "<YOUR_INPUT_FASTQ_R2_
→TECHREP1>.fastq.gz"],
            ["<YOUR_INPUT_FASTQ_R1_TECHREP2>.fastq.gz", "<YOUR_INPUT_FASTQ_R2_
→TECHREP2>.fastq.gz"]
        ]]
      },
      "chip.bwa_idx_tar": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
```

(continues on next page)

```
      "rename": "GRCh38_no_alt_analysis_set_GCA_000001405.15.fasta.tar",
      "object_key": "<YOUR_INPUT_TAR_BWA_INDEX>"
    },
    "chip.blacklist": {
      "bucket_name": "<YOUR_INPUT_BUCKET>",
      "object_key": "<YOUR_BLACKLIST_FILE>.bed.gz"
    },
    "chip.chrsz": {
      "bucket_name": "<YOUR_INPUT_BUCKET>",
      "object_key": "<YOUR_CHROMSIZES_FILE>.chrom.sizes"
    }
  },
  "input_parameters": {
      "chip.pipeline_type" : "histone",
      "chip.paired_end" : true,
      "chip.choose_ctl.always_use_pooled_ctl" : true,
      "chip.qc_report.name" : "<YOUR_QC_REPORT_NAME>",
      "chip.qc_report.desc" : "<YOUR_QC_REPORT_DESCRIPTION>",
      "chip.gensz" : "hs",
      "chip.bam2ta.regex_grep_v_ta" : "chr[MUE]|random|alt",
      "chip.fraglen": [],
      "chip.bwa.cpu": 16,
      "chip.merge_fastq.cpu": 16,
      "chip.filter.cpu": 16,
      "chip.bam2ta.cpu": 16,
      "chip.xcor.cpu": 16,
      "chip.align_only": true
  },
  "output_S3_bucket": "<YOUR_INPUT_BUCKET>",
  "output_target": {
      "chip.first_ta": "<YOUR_OUTPUT_TAG_ALIGN_FILE>.bed.gz",
      "chip.first_ta_xcor": "<YOUR_OUTPUT_TAG_ALIGN_FILE_FOR_XCOR>.bed.gz"
  }
  },
  "config": {
    "log_bucket": "<YOUR_LOG_BUCKET>",
    "key_name": "<YOUR_KEY_NAME>"
  }
}
```
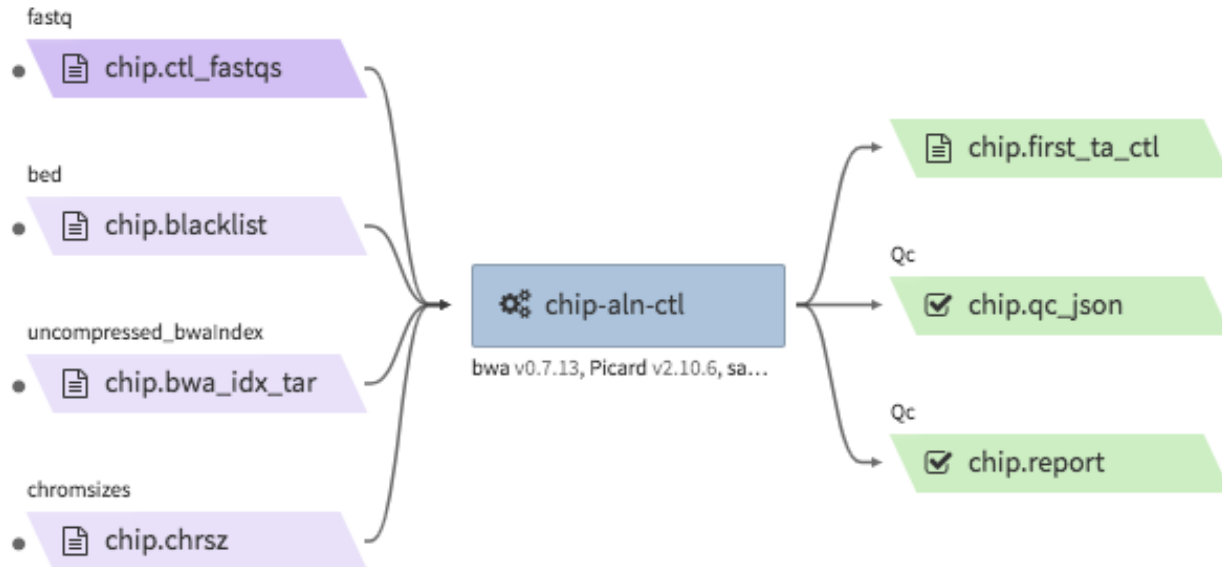
encode-chipseq-aln-ctl

- Description : takes in control fastq files from a single biological replicate (may consist of multiple technical replicates) and generates a TagAlign file for that biological replicate.

- WDL : https://github.com/4dn-dcic/chip-seq-pipeline2/blob/4dn-v1.1.1/chip.wdl

- Docker : `4dndcic/encode-chipseq:v1.1.1`

- 4DN workflow metadata : https://data.4dnucleome.org/4dn-dcic-lab:wf-encode-chipseq-aln-ctl

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/f02336f6-aa6e-491d-8562-db61bcc86303/

- Example input execution json template :

Use the following as a template and replace <YOUR....> with your input/output/log bucket/file(object) information.

```
{
  "args": {
    "app_name": "encode-chipseq-aln-ctl",
    "app_version": "v1.1.1",
    "wdl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/chip-seq-
→pipeline2/4dn-v1.1.1/",
    "wdl_main_filename": "chip.wdl",
    "language": "wdl",
    "input_files": {
      "chip.ctl_fastqs": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": [[
            ["<YOUR_INPUT_FASTQ_R1_TECHREP1>.fastq.gz", "<YOUR_INPUT_FASTQ_R2_
→TECHREP1>.fastq.gz"],
            ["<YOUR_INPUT_FASTQ_R1_TECHREP2>.fastq.gz", "<YOUR_INPUT_FASTQ_R2_
→TECHREP2>.fastq.gz"]
        ]]
      },
      "chip.bwa_idx_tar": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "rename": "GRCh38_no_alt_analysis_set_GCA_000001405.15.fasta.tar",
        "object_key": "<YOUR_INPUT_TAR_BWA_INDEX>"
      },
      "chip.blacklist": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_BLACKLIST_FILE>.bed.gz"
      },
      "chip.chrsz": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_CHROMSIZES_FILE>.chrom.sizes"
      }
    },
    "input_parameters": {
        "chip.pipeline_type" : "histone",
```

(continues on next page)

```
        "chip.paired_end" : true,
        "chip.choose_ctl.always_use_pooled_ctl" : true,
        "chip.qc_report.name" : "<YOUR_QC_REPORT_NAME>",
        "chip.qc_report.desc" : "<YOUR_QC_REPORT_DESCRIPTION>",
        "chip.gensz" : "hs",
        "chip.bam2ta_ctl.regex_grep_v_ta" : "chr[MUE]|random|alt",
        "chip.fraglen": [],
        "chip.bwa_ctl.cpu": 16,
        "chip.merge_fastq_ctl.cpu": 16,
        "chip.filter_ctl.cpu": 16,
        "chip.bam2ta_ctl.cpu": 16,
        "chip.align_only": true
    },
    "output_S3_bucket": "<YOUR_INPUT_BUCKET>",
    "output_target": {
        "chip.first_ta": "<YOUR_OUTPUT_TAG_ALIGN_FILE>.bed.gz",
        "chip.first_ta_xcor": "<YOUR_OUTPUT_TAG_ALIGN_FILE_FOR_XCOR>.bed.gz"
    }
  },
  "config": {
    "log_bucket": "<YOUR_LOG_BUCKET>",
    "key_name": "<YOUR_KEY_NAME>"
  }
}
```
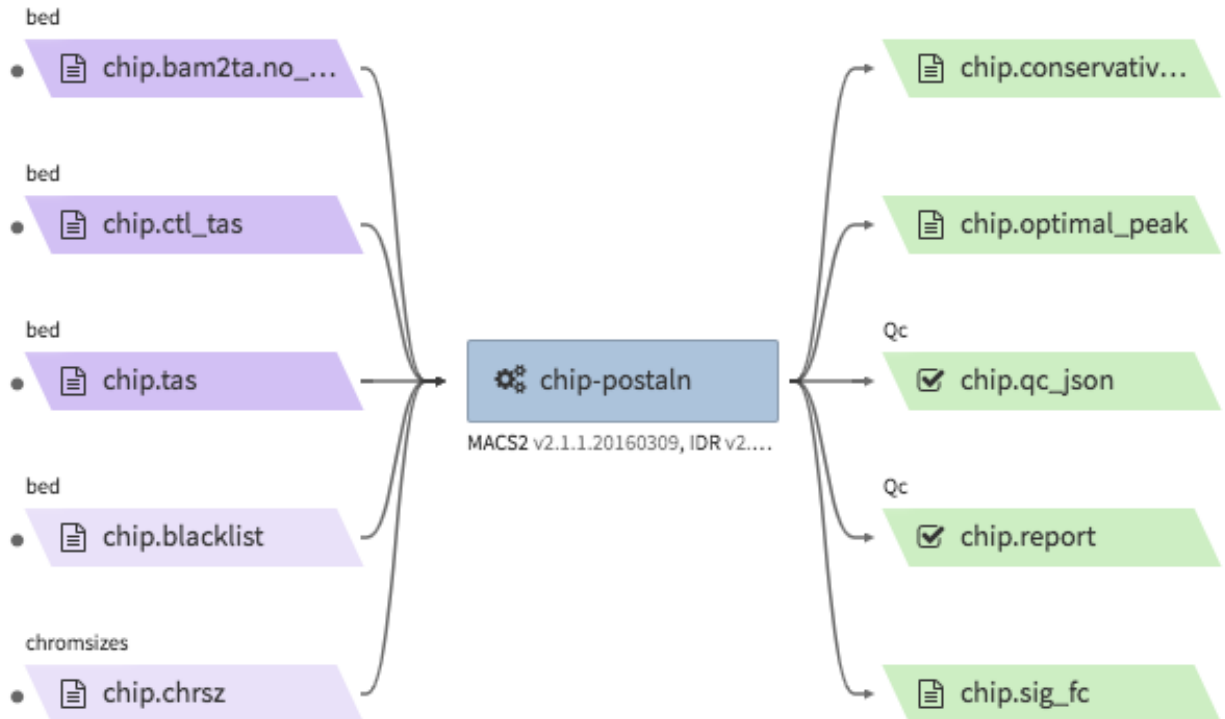
encode-chipseq-postaln

- Description : takes in TagAlign files generates from `encode-chipseq-aln-chip` and `encode-chipsq-aln-ctl` and calls peaks. The output files are signal fold change (bigwig) and two peak call sets (bigbed). The pipeline cannot handle more than two biological replicates due to the limitation of the ENCODE pipeline.

- WDL : https://github.com/4dn-dcic/chip-seq-pipeline2/blob/4dn-v1.1.1/chip.wdl

- Docker : `4dndcic/encode-chipseq:v1.1.1`

- 4DN workflow metadata : https://data.4dnucleome.org/4dn-dcic-lab:wf-encode-chipseq-postaln

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/8666c89e-eccb-4dc1-9b12-ceb04802ca09/

- Example input execution json template :

Use the following as a template and replace <YOUR....> with your input/output/log bucket/file(object) information.

```
{
    "args": {
      "app_name": "encode-chipseq-postaln",
      "app_version": "v1.1.1",
      "wdl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/chip-seq-
→pipeline2/4dn-v1.1.1/",
      "wdl_main_filename": "chip.wdl",
      "language": "wdl",
      "input_files" : {
        "chip.tas" : {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": ["<YOUR_INPUT_TAG_ALIGN_BIOREP1>.bed.gz",
                         "<YOUR_INPUT_TAG_ALIGN_BIOREP2>.bed.gz"],
          "rename": ["<YOUR_INPUT_TAG_ALIGN_BIOREP1>.tagAlign.gz",
                     "<YOUR_INPUT_TAG_ALIGN_BIOREP2>.tagAlign.gz"]
        },
        "chip.ctl_tas" : {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": ["<YOUR_INPUT_CTL_TAG_ALIGN_BIOREP1>.bed.gz",
                         "<YOUR_INPUT_CTL_TAG_ALIGN_BIOREP2>.bed.gz"],
          "rename": ["<YOUR_INPUT_CTL_TAG_ALIGN_BIOREP1>.tagAlign.gz",
                     "<YOUR_INPUT_CTL_TAG_ALIGN_BIOREP2>.tagAlign.gz"]
        },
        "chip.bam2ta_no_filt_R1.ta" : {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": ["<YOUR_INPUT_XCOR_TAG_ALIGN_BIOREP1>.bed.gz",
                         "<YOUR_INPUT_XCOR_TAG_ALIGN_BIOREP1>.bed.gz"],
          "rename": ["<YOUR_INPUT_XCOR_TAG_ALIGN_BIOREP1>.tagAlign.gz",
```

(continues on next page)

```
                "<YOUR_INPUT_XCOR_TAG_ALIGN_BIOREP2>.tagAlign.gz"]
        },
        "chip.blacklist" : {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": "<YOUR_BLACKLIST_FILE>.bed.gz"
        },
        "chip.chrsz" : {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": "<YOUR_CHROMSIZES_FILE>"
        }
    },
    "input_parameters": {
      "chip.pipeline_type" : "histone",
      "chip.paired_end" : true,
      "chip.choose_ctl.always_use_pooled_ctl" : true,
      "chip.qc_report.name" : "<YOUR_QC_REPORT_NAME>",
      "chip.qc_report.desc" : "<YOUR_QC_REPORT_DESCRIPTION>",
      "chip.gensz" : "hs",
      "chip.xcor.cpu": 4,
      "chip.spp_cpu": 4
    },
    "output_S3_bucket": "<YOUR_OUTPUT_BUCKET>",
    "output_target": {
      "chip.sig_fc": "<YOUR_OUTPUT_SIGNAL_FC_FILE>.bw",
      "chip.optimal_peak": "<YOUR_OUTPUT_OPTIMAL_PEAK_FILE>.bb",
      "chip.conservative_peak": "<YOUR_OUTPUT_CONSERVATIVE_PEAK_FILE>.bb",
      "chip.report": "<YOUR_OUTPUT_QC_REPORT>.html",
      "chip.qc_json": "<YOUR_OUTPUT_QC_JSON>.json"
    }
  },
  "config": {
    "log_bucket": "<YOUR_LOG_BUCKET>",
    "key_name": "<YOUR_KEY_NAME>"
  }
}
```
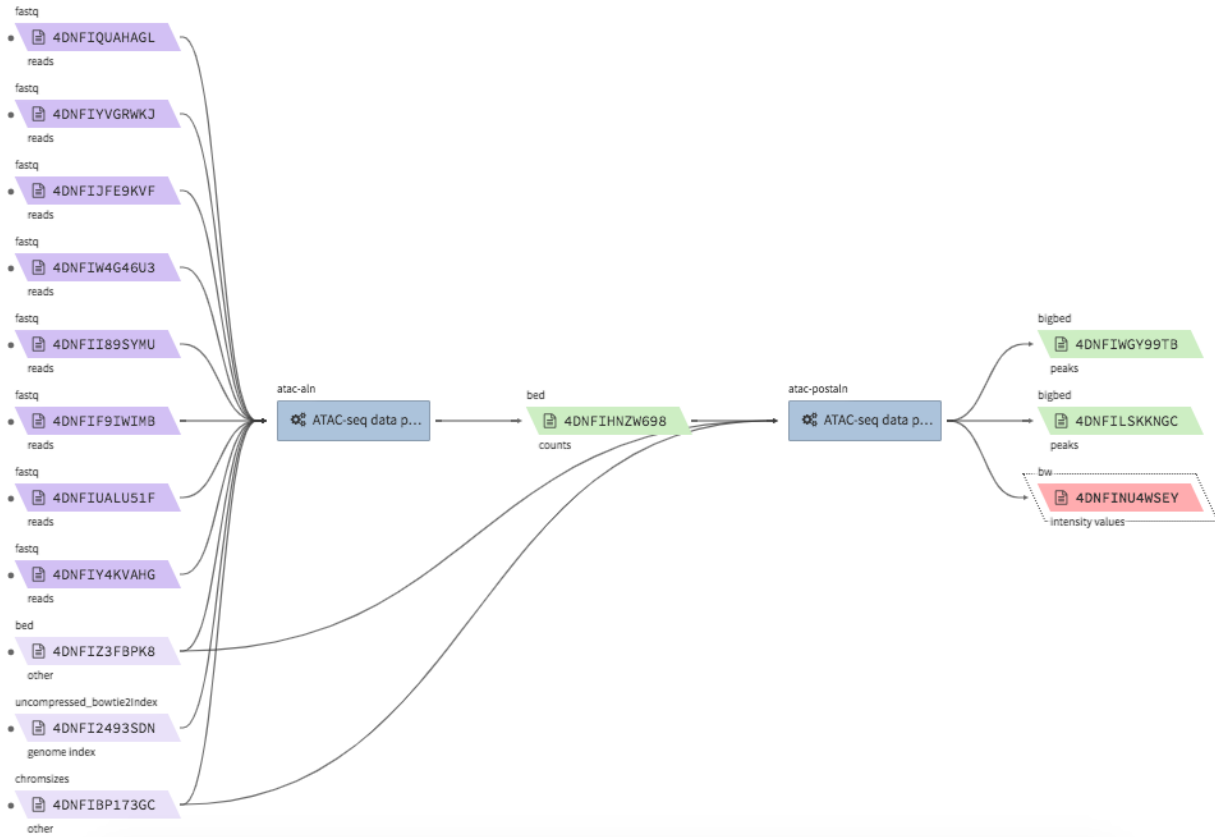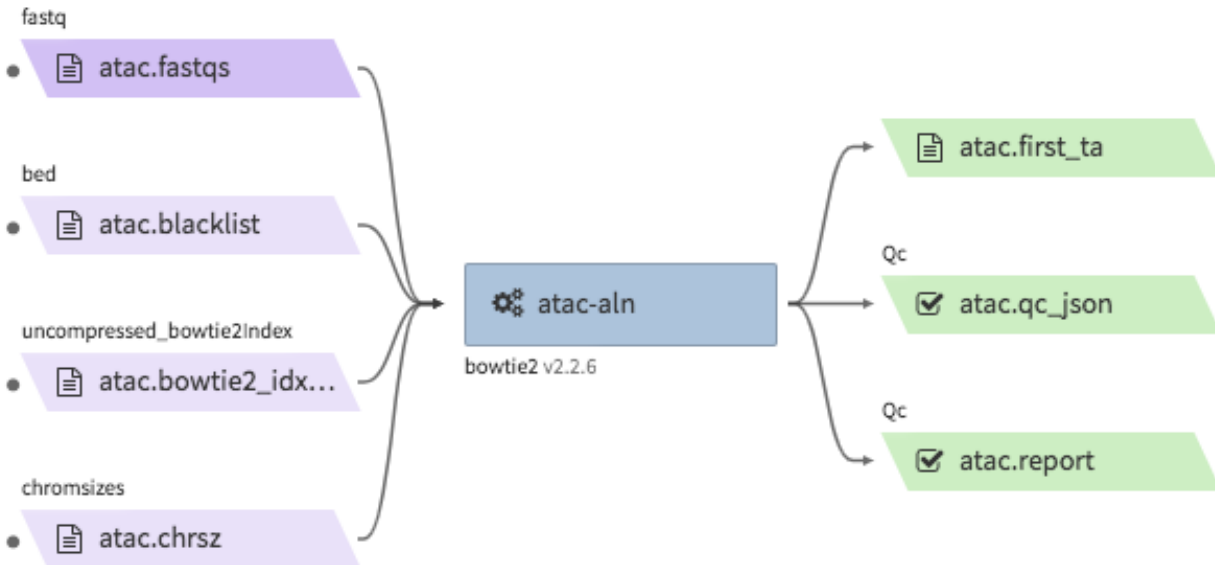
ATAC-seq data processing & QC

- Example full pipeline run

encode-atacseq-aln

- Description : takes in fastq files from a single biological replicate (may consist of multiple technical replicates) and generates a TagAlign file for that biological replicate.

- WDL : https://github.com/4dn-dcic/atac-seq-pipeline/blob/4dn-v1.1.1/atac.wdl

- Docker : `4dndcic/encode-atacseq:v1.1.1`

- 4DN workflow metadata : https://data.4dnucleome.org/4dn-dcic-lab:wf-encode-atacseq-aln

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/c57697c4-c589-4025-ad81-e212a5220f74/

- Example input execution json template :

Use the following as a template and replace <YOUR....> with your input/output/log bucket/file(object) information.

```
{
  "args": {
    "app_name": "encode-atacseq-aln",
    "app_version": "1.1.1",
    "wdl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/atac-seq-
→pipeline/4dn-v1.1.1/",
    "wdl_main_filename": "atac.wdl",
    "language": "wdl",
    "input_files": {
      "atac.bowtie2_idx_tar": {
        "rename": "mm10_no_alt_analysis_set_ENCODE.fasta.tar",
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_TAR_BOWTIE2_INDEX>"
      },
      "atac.fastqs": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": [[
            ["<YOUR_INPUT_FASTQ_R1_TECHREP1>.fastq.gz", "<YOUR_INPUT_FASTQ_R2_
→TECHREP1>.fastq.gz"],
            ["<YOUR_INPUT_FASTQ_R1_TECHREP2>.fastq.gz", "<YOUR_INPUT_FASTQ_R2_
→TECHREP2>.fastq.gz"]
          ]]
      },
      "atac.blacklist": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_BLACKLIST_FILE>.bed.gz"
      },
      "atac.chrsz": {
        "bucket_name": "<YOUR_INPUT_BUCKET>",
        "object_key": "<YOUR_CHROMSIZES_FILE>"
      }
    },
    "input_parameters": {
```
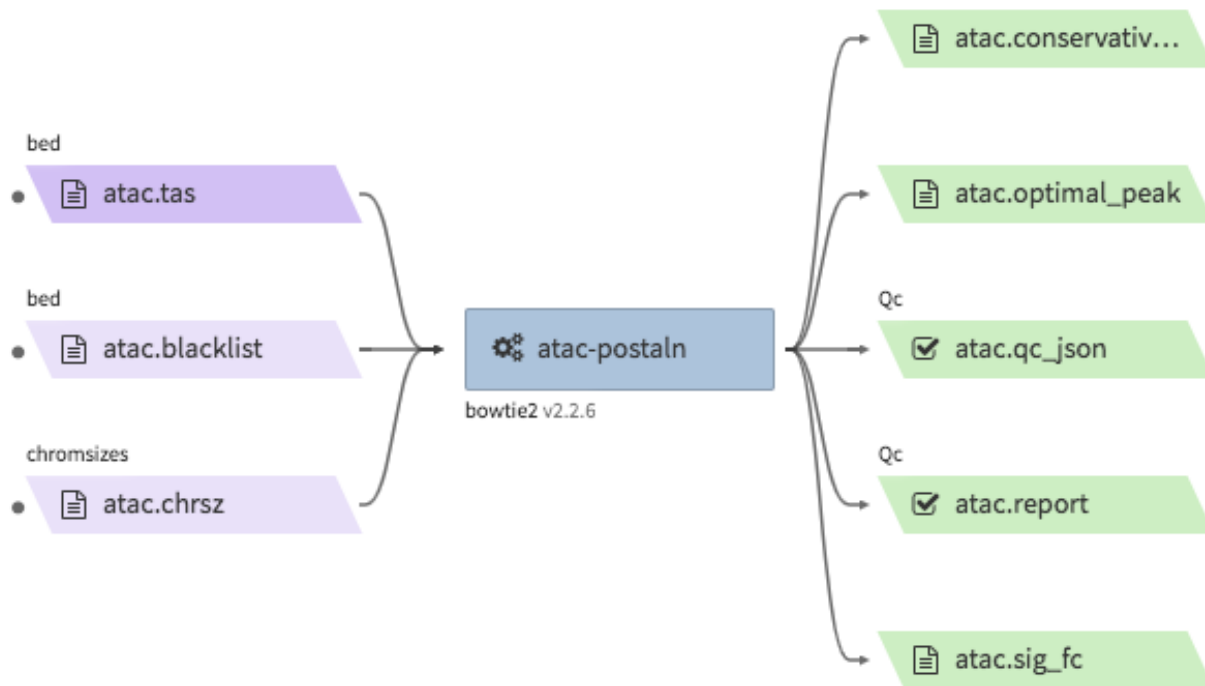
(continues on next page)

```
      "atac.trim_adapter.cpu": 4,
      "atac.paired_end": true,
      "atac.bam2ta.regex_grep_v_ta": "chr[MUE]|random|alt",
      "atac.enable_xcor": false,
      "atac.disable_ataqc": true,
      "atac.filter.cpu": 4,
      "atac.trim_adapter.auto_detect_adapter": true,
      "atac.bam2ta.cpu": 4,
      "atac.bowtie2.cpu": 4,
      "atac.gensz": "mm",
      "atac.pipeline_type": "atac",
      "atac.align_only": true
    },
    "output_S3_bucket": "<YOUR_OUTPUT_BUCKET>",
    "output_target": {
      "atac.first_ta": "<YOUR_OUTPUT_TAGALIGN>.bed.gz",
      "atac.report": "<YOUR_OUTPUT_QC_REPORT>.html",
      "atac.qc_json": "<YOUR_OUTPUT_QC_JSON.json>",
    }
  },
  "config": {
    "log_bucket": "<YOUR_LOG_BUCKET>",
    "key_name": "<YOUR_KEY_NAME>"
  }
}
```

encode-atacseq-postaln

- Description : takes in TagAlign files generates from `encode-atacseq-aln` and calls peaks. The output files are signal fold change (bigwig) and two peak call sets (bigbed). The pipeline cannot handle more than two biological replicates due to the limitation of the ENCODE pipeline.

- WDL : https://github.com/4dn-dcic/atac-seq-pipeline/blob/4dn-v1.1.1/atac.wdl

- Docker : `4dndcic/encode-atacseq:v1.1.1`

- 4DN workflow metadata : https://data.4dnucleome.org/4dn-dcic-lab:wf-encode-atacseq-postaln

- 4DN example run: https://data.4dnucleome.org/workflow-runs-awsem/afe50cb7-7417-4870-a5be-060600738fb0/

- Example input execution json template :

Use the following as a template and replace <YOUR....> with your input/output/log bucket/file(object) information.

```
{
    "args": {
      "app_name": "encode-atacseq-postaln",
      "app_version": "v1.1.1",
      "wdl_directory_url": "https://raw.githubusercontent.com/4dn-dcic/atac-seq-
→pipeline/4dn-v1.1.1/",
      "wdl_main_filename": "atac.wdl",
      "language": "wdl",
      "input_files" : {
        "atac.tas" : {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": ["<YOUR_INPUT_TAG_ALIGN_BIOREP1>.bed.gz",
                         "<YOUR_INPUT_TAG_ALIGN_BIOREP2>.bed.gz"],
          "rename": ["<YOUR_INPUT_TAG_ALIGN_BIOREP1>.tagAlign.gz",
                     "<YOUR_INPUT_TAG_ALIGN_BIOREP2>.tagAlign.gz"]
        },
        "atac.blacklist" : {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": "<YOUR_BLACKLIST_FILE>.bed.gz"
        },
        "atac.chrsz" : {
          "bucket_name": "<YOUR_INPUT_BUCKET>",
          "object_key": "<YOUR_CHROMSIZES_FILE>"
        }
      },
      "input_parameters": {
        "atac.pipeline_type" : "atac",
        "atac.paired_end" : true,
        "atac.gensz" : "hs",
```

(continues on next page)

```
      "atac.disable_ataqc": true,
      "atac.enable_xcor": false
    },
    "output_S3_bucket": "<YOUR_OUTPUT_BUCKET>",
    "output_target": {
      "atac.sig_fc": "<YOUR_OUTPUT_SIGNAL_FC_FILE>.bw",
      "atac.optimal_peak": "<YOUR_OUTPUT_OPTIMAL_PEAK_FILE>.bb",
      "atac.conservative_peak": "<YOUR_OUTPUT_CONSERVATIVE_PEAK_FILE>.bb",
      "atac.report": "<YOUR_OUTPUT_QC_REPORT>.html",
      "atac.qc_json": "<YOUR_OUTPUT_QC_JSON>.json"
    }
  },
  "config": {
    "log_bucket": "<YOUR_LOG_BUCKET>",
    "key_name": "<YOUR_KEY_NAME>"
  }
}
```

1.4.14 How it works

Tibanna launches and monitors pipeline runs using two-layer scheduling. The upstream regulator is based on a finite state machine called AWS Step Function and the downstream workflow engine is based on `cwltool` which runs Docker/CWL-based pipelines on an EC2 instance (or `cromwell` for Docker/WDL-based pipelines). Tibanna's AWS Step Function launches several AWS Serverless Lambda functions that submits and monitors a pipeline execution on a pre-custom-configured autonomous virtual machine (EC2 instance) (AWSEM; Autonomous Workflow Step Executor Machine). The `cwltool`/`'cromwell'` is auto-executed on an instance.

Tibanna allows multi-layer, real-time monitoring. The logs of what's happening on an instance including the `cwltool` log is regularly sent to a designated S3 bucket (Tibanna log bucket). Logs generated by the AWS Lambda functions are sent to AWS CloudWatch, a service provided by AWS; AWS Step function sends logs either as an output json or as an exception. Users can ssh into the EC2 instance where a workflow is currently being executed, for more detailed investigation. A metrics plot is generated and stored for every job for monitoring how CPU/Memory/disk usage changes over time during the run and for each process. The user can also check the top command outputs generated from the instance at 1 minute interval, without ssh-ing into the machine, since these reports are sent to S3 regularly. Tibanna provides API to access these reports easily.

Tibanna uses AWS IAM roles to ensure secure access but also allows use of profile information for accessing public data that requires AWS access keys and secret keys by setting environmental variables for AWS Lambda functions. There is no need to store any security information inside docker image or anywhere in the code.

We have also implemented an accompanying resource optimizer for 4DN pipelines (https://github.com/SooLee/Benchmark), which calculates total CPU, memory and space required for a specific workflow run to determine EC2 instance type and EBS (Elastic Block Store) volume size. The calculation is based on input size, workflow parameters and the benchmarking results characteristic of individual workflows. The resource optimizer is essential for automated parameterization of data-dependent workflow runs, while maximizing the benefit of the elasticity of the cloud platform. Tibanna currently uses this optimizer to auto-determine instance types and EBS sizes for 4DN workflow runs.