# August 12th Workshop Materials

## Instructions for Hands-on Exercises

### Mini Mutation Calling Exercise

**Workspace:** *broad-firecloud-tutorials/ToolDevWorkshop\_MiniMutationCalling\_*

**Methods:** *broadinstitute/MiniMutationCalling* (ContEst, MuTect, Oncotator)

**Method Configs:** *mhanna/MiniMutationCalling\_Cfg*

**Data:** 2 pairs of cell lines; a whole exome pair, and a “tiny” 100-gene pair of BAMs

**Entities:** participant, sample, pair

**Results:** Nozzle Report

*This Mini Mutation Calling Tutorial includes a subset of tools from our complete Broad Mutation Calling Best Practice Workflow. It contains ContEst, MuTect, and Oncotator.*

For this exercise, we will clone this workspace which is prepopulated with data and a method configuration. Then we will launch the MiniMutationCalling workflow analysis on 2 pairs of cell line BAMs, tiny BAMs containing only 100 genes and whole exome BAMs.

When run on the 100-gene BAMs, the expected runtime is roughly 30 minutes. When run on whole exome BAMs, the expected runtime is roughly 3 hours.

#### Steps

To run the Mini Mutation Calling Tutorial,

1. Navigate to the *broad-firecloud-tutorials/ToolDevWorkshop\_MiniMutationCalling\_* workspace. You can copy this name and paste it into the search bar.
2. Clone the *broad-firecloud-workshops/ToolDevWorkshop\_MiniMutationCalling\_* workspace under the Google Project, *broad-firecloud-workshops* and give it a unique name, e.g., *broad-firecloud-workshops/ToolDevWorkshop\_MiniMutationCalling\_<user name>*.
3. In your cloned workspace, navigate to the Data tab.
4. Upload TSV files in the correct order. First, be sure to download and unzip the *Workshop\_Materials* folder. You will find the TSV files in a subfolder called *MiniMutCallingExercise*. In the Data tab, click **Import Data** → **Import from file** → **Choose file**. You should first upload from *MiniMutCallingExercise 1\_participant.txt*, then *2\_sample.txt*, then *3\_pair.txt*. Review the Data tab to confirm that participant, sample, and pair data appears.
5. Navigate to the Method Configurations tab.
6. Select the *MiniMutationCalling\_Cfg* Method Config and click **Launch Analysis**.
7. In the Launch Analysis window, toggle to pair and select *HCC1143\_WE\_pair*. This is a pair (tumor and normal) of whole exome BAMs on which to run this analysis. Click **Launch**.
8. Return to the Method Configurations tab and select the *MiniMutationCalling\_Cfg* again. Click **Launch Analysis** and this time select *HCC1954\_100\_gene\_pair.* This is a pair of tiny (100 gene) tumor/normal BAMs. Click **Launch**.
9. Check the Monitor tab in about 30 minutes to see if your analysis run on *HCC1954\_100\_gene\_pair* completed. The analysis run for *HCC1143\_WE\_pair* finish in about 3 hours.
10. When the Monitor tab displays Done, click on the Data tab and, using the filtering widget on the left-hand side of the page, filter down to *Entity Name* and *nozzle\_report*. Open the Nozzle Report and review the results of the run.

## Tool Developer Exercises

# 0. Preliminaries

## 0.1 docker: Cannot connect to Docker daemon

If using Docker ToolBox, when issuing a docker run command on your laptop console, you receive the message:

docker: Cannot connect to the Docker daemon. Is the docker daemon running on this host?.

the following shell commands should address the problem:

$ docker-machine restart default

$ eval $(docker-machine env default)

## 0.2 What directory to run the exercises in

You may choose to unzip the Workshop\_Materials.zip archive and run these exercises in any directories of your choosing. You will need write-access to the directory in which you run the exercises.

The recipes below assume the Workshop\_Materials.zip file has been unzipped in the user’s (birger) home directory, and all exercises are run in ~/Workshop\_Exercises. The recipes also assume that you downloaded Cromwell and wdltool from [**Software Installation Instructions**](https://docs.google.com/document/d/1tnj0FjJQ4dM5__HaoDeC-O65FIcwGyYxpG2qczGZfI4/).

# 1. Hello World

In this exercise we work with a previously dockerized Hello World application written in Python.

The Hello World python script has been included with the zip archive (Workshop\_Materials.zip). Note that the following instructions assume you are running the exercise from your home directory and have unzipped Workshop\_Materials.zip in your home directory. You may choose to run the exercise in any directory of your choosing, provided you have write access to that directory. The dockerized version of the application (a docker image) has been written the repository cbirger/hello-world on docker hub, with the tag “5.0”.

**1.1 Test dockerized helloworld application in local docker container**

1. If using Docker Toolbox, launch Launch the Docker QuickStart Terminal; if using Docker for Mac, launch a terminal.
2. Pull the docker image cbirger/hello-world:6.0 from docker hub into your local image repository.

|  |
| --- |
| wm8b6-23c:~ birger$ **docker pull cbirger/hello-world:6.0**  6.0: Pulling from cbirger/hello-world  51f5c6a04d83: Pull complete  a3ed95caeb02: Pull complete  7004cfc6e122: Pull complete  5f37c8a7cfbd: Pull complete  8ad7684cace4: Pull complete  22c0978ecede: Pull complete  b1d402426f39: Pull complete  e7d853e7d62d: Pull complete  6abc0b6e5732: Pull complete  0d96eba6631d: Pull complete  ed7521fa4c17: Pull complete  4deeb48f16f4: Pull complete  Digest: sha256:a23c05411c47f0a21b5f9b607749d0b58dcde691d7a4cf22094afa8b0e77fb77  Status: Downloaded newer image for cbirger/hello-world:6.0  wm8b6-23c:~ birger$ |

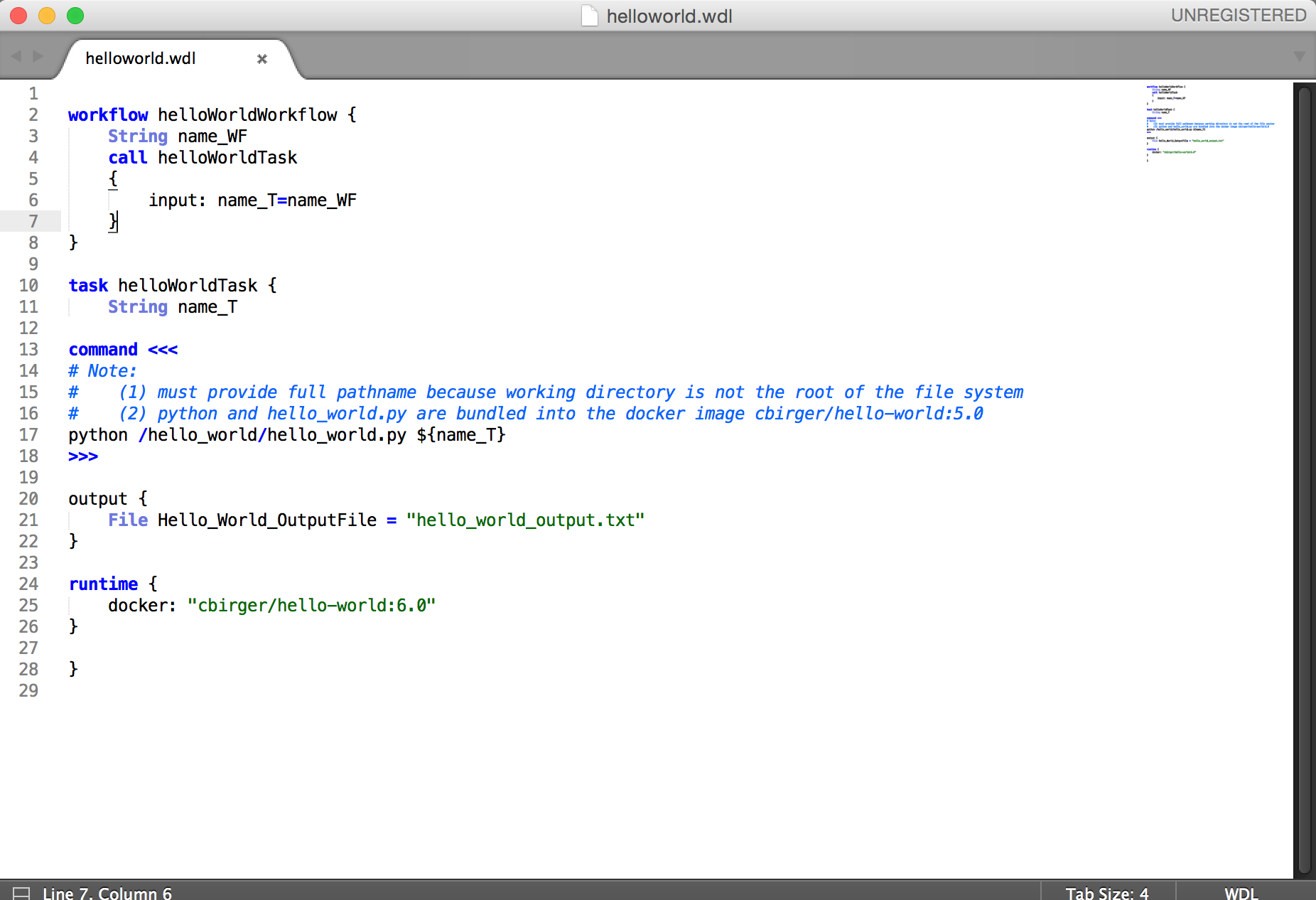
1. Run helloworld application in docker container

|  |
| --- |
| wm8b6-23c:~ birger$ **docker run cbirger/hello-world:6.0 python /hello\_world/hello\_world.py**  Hello World!  wm8b6-23c:~ birger$ **docker run cbirger/hello-world:6.0 python /hello\_world/hello\_world.py "workshop"**  Hello workshop!  wm8b6-23c:~ birger$ |

* Note that the hello\_world.py which we run is in the container loaded with the image cbirger/hello-world:6.0, not the local copy in the Workshop\_Material folder.

**1.2 Create single-task WDL workflow that calls helloworld and test it in locally running instance of Cromwell**

We have already provided you with this wdl file (helloworld.wdl) in the zip archive mailed to workshop attendees.



**1.3 Test helloworld WDL running Cromwell locally**

1. Use wdltool to validate syntax; blank response indicates the WDL is correct

|  |
| --- |
| wm8b6-23c:Workshop\_Materials birger$ **pwd**  /Users/birger/Workshop\_Exercises  wm8b6-23c:Workshop\_Exercises birger$ **java -jar ~/wdl/bin/wdltool-0.4.jar validate ~/Workshop\_Materials/HelloWorld/helloworld.wdl**  wm8b6-23c:Workshop\_Exercises birger$ |

1. Create json template for workflow inputs

|  |
| --- |
| wm8b6-23c:Workshop\_Exercises birger$ **java -jar ~/wdl/bin/wdltool-0.4.jar inputs ~/Workshop\_Materials/HelloWorld/helloworld.wdl**  {  "helloWorldWorkflow.name\_WF": "String"  }  wm8b6-23c:Workshop\_Exercises birger$ **java -jar ~/wdl/bin/wdltool-0.4.jar inputs ~/Workshop\_Materials/HelloWorld/helloworld.wdl > helloworld\_input.json**  wm8b6-23c:Workshop\_Exercises birger$ |

Using your text editor, replace “String” above with name of whom you want to greet; e.g.,

"helloWorldWorkflow.name\_WF": "Workshop"

1. Run WDL with local Cromwell

|  |
| --- |
| wm8b6-23c:Workshop\_Exercises birger$ **java -jar ~/wdl/bin/cromwell-0.19.3.jar run ~/Workshop\_Materials/HelloWorld/helloworld.wdl helloworld\_input.json**  [2016-08-09 14:48:19,104] [info] Slf4jLogger started  [2016-08-09 14:48:19,157] [info] RUN sub-command  [2016-08-09 14:48:19,157] [info] WDL file: /Users/birger/Workshop\_Materials/helloworld.wdl  [2016-08-09 14:48:19,158] [info] Inputs: helloworld\_input.json  [2016-08-09 14:48:19,233] [info] SingleWorkflowRunnerActor: launching workflow  . . .  . . .  . . .  [2016-08-09 14:48:26,203] [info] WorkflowActor [3c8ad7f9]: persisting status of helloWorldWorkflow.$final\_call$copy\_call\_logs to Done.  [2016-08-09 14:48:26,319] [info] WorkflowActor [3c8ad7f9]: persisting status of helloWorldWorkflow.$final\_call$copy\_workflow\_log to Done.  [2016-08-09 14:48:26,330] [info] WorkflowActor [3c8ad7f9]: Beginning transition from Running to Succeeded.  [2016-08-09 14:48:26,338] [info] WorkflowActor [3c8ad7f9]: transitioning from Running to Succeeded.  {  "helloWorldWorkflow.helloWorldTask.Hello\_World\_OutputFile": "/Users/birger/Workshop\_Exercises/cromwell-executions/helloWorldWorkflow/3c8ad7f9-7099-435a-9048-1ad8bddd2a2d/call-helloWorldTask/hello\_world\_output.txt"  }  [2016-08-09 14:48:26,354] [info] SingleWorkflowRunnerActor workflow finished with status 'Succeeded'.  wm8b6-23c:Workshop\_Exercises birger$ |

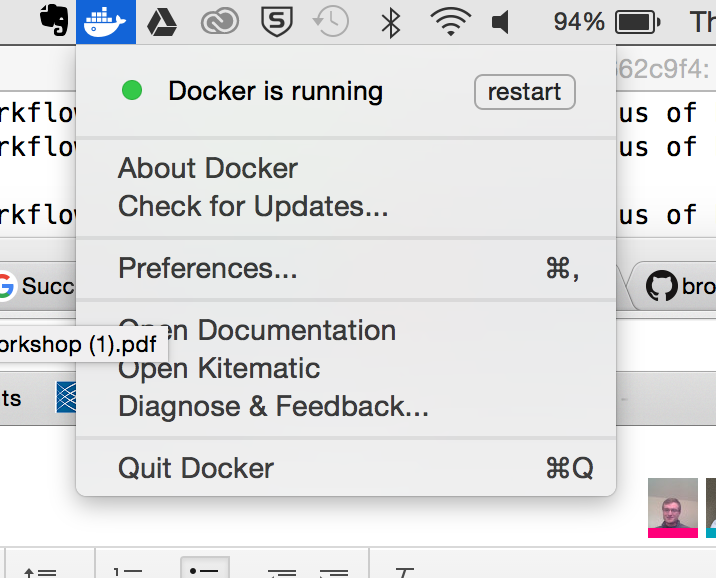
1. Look at output

|  |
| --- |
| wm8b6-23c:Workshop\_Exercises birger$ **ls**  cromwell-executions cromwell-workflow-logs helloworld\_input.json  wm8b6-23c:Workshop\_Exercises birger$ **cd cromwell-executions/helloWorldWorkflow/**  wm8b6-23c:helloWorldWorkflow birger$ **ls**  3c8ad7f9-7099-435a-9048-1ad8bddd2a2d  wm8b6-23c:helloWorldWorkflow birger$ **cd 3c8ad7f9-7099-435a-9048-1ad8bddd2a2d/**  wm8b6-23c:3c8ad7f9-7099-435a-9048-1ad8bddd2a2d birger$ **ls**  call-helloWorldTask  wm8b6-23c:3c8ad7f9-7099-435a-9048-1ad8bddd2a2d birger$ **cd call-helloWorldTask/**  wm8b6-23c:call-helloWorldTask birger$ **ls**  hello\_world\_output.txt rc script stderr stdout  wm8b6-23c:call-helloWorldTask birger$ **cat hello\_world\_output.txt**  Hello Workshop!  wm8b6-23c:call-helloWorldTask birger$ |

**1.4 Use firecloud CLI to upload WDL to FireCloud Method Repository**

We do this from a docker container; it is easier to run the official broadinstitute/firecloud-cli docker image in a docker container than build and install the firecloud CLI to run natively on your machine.

1. First make sure your docker engine is running locally
   * If running Docker Toolbox on a Mac OS X system, that means running the Docker quickstart terminal.
   * If running Docker for Mac, that means checking the status bar to ensure docker is running



1. Then cd to the directory where your wdl file resides.
2. From that directory launch an interactive bash shell in docker container loaded with the broadinstitute/firecloud-cli docker image

|  |
| --- |
| wm8b6-23c:Workshop\_Exercises birger$ **cd ~/Workshop\_Materials/HelloWorld**  wm8b6-23c:Workshop\_Materials birger$ **docker run --rm -it -v "$HOME"/.config:/.config -v "$PWD":/working broadinstitute/firecloud-cli bash**  root@99e9a7c66347:/working# |

* Note the mounting of your current working directory to /working in the container

**-v "$PWD":/working**

* Note the mounting of the .config directory to the container’s .config directory. **-v "$HOME"/.config:/.config** This will allow us to make user credentials available across multiple containers.

1. Within the running docker container run gcloud auth login

|  |
| --- |
| root@0a45e7c10a4e:/working# **gcloud auth login**  Go to the following link in your browser:  https://accounts.google.com/o/oauth2/auth?redirect\_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&prompt=select\_account&response\_type=code&client\_id=32555940559.apps.googleusercontent.com&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fappengine.admin+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcompute&access\_type=offline  Enter verification code: |

Choose the Gmail or Google Apps account under which you registered for FireCloud.





Copy and paste the verification code from your browser login session into the docker container session.

|  |
| --- |
| Enter verification code: **4/Wotli2zQ\_5tgFu7idXzL7rHbdtgxYRuJGuAKcVcYbK0**  Saved Application Default Credentials.  You are now logged in as [birger@broadinstitute.org].  Your current project is [broad-buccp-dev]. You can change this setting by running:  $ gcloud config set project PROJECT\_ID |

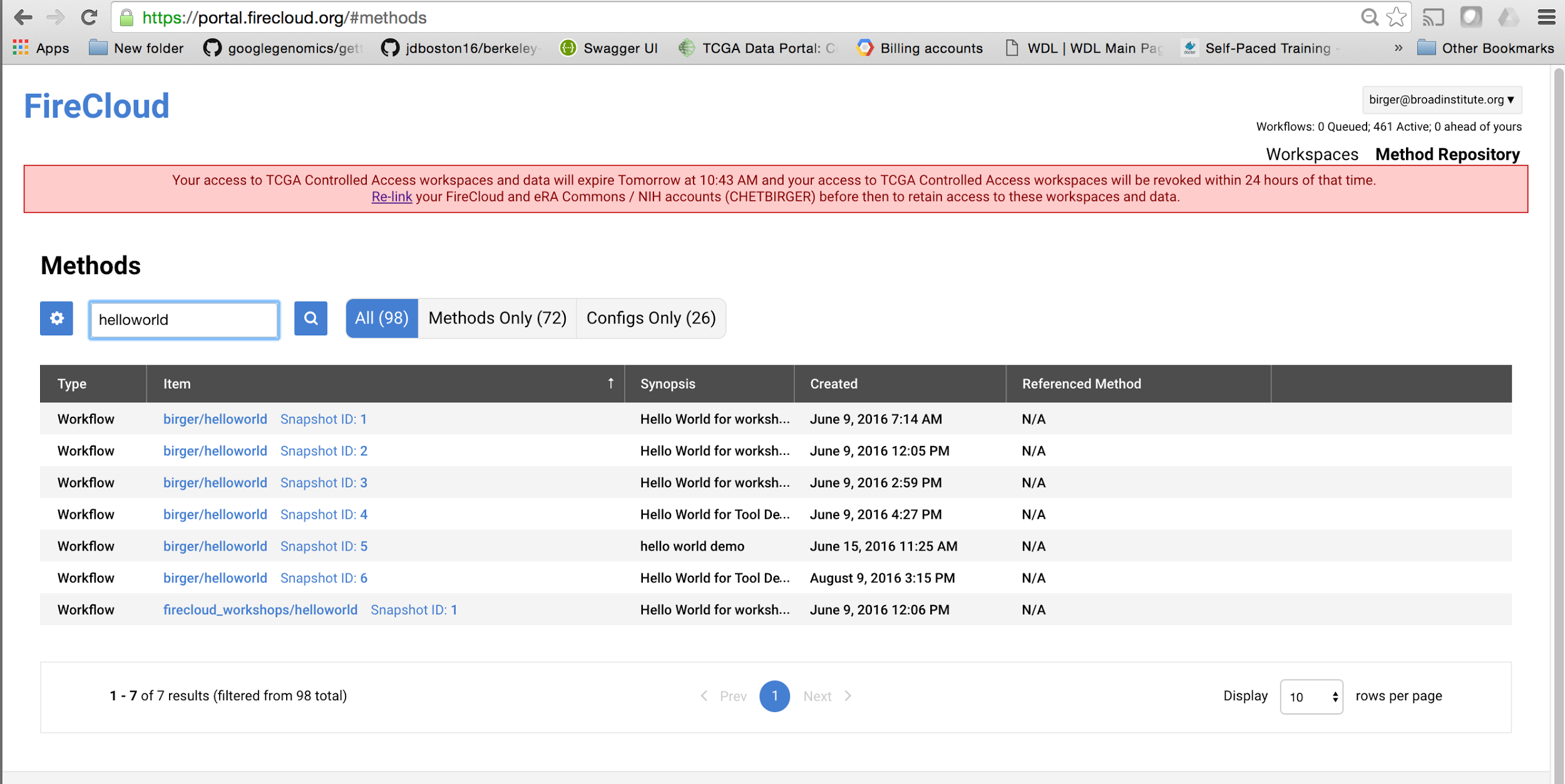
Now any firecloud commands run in the container will have access to credentials needed to authenticate to firecloud. (In addition, because the .config directory in the container binds back to .config on your local machine, new docker containers with the same binding can use those credentials.)

1. Within the same container run the firecloud cli command to upload a copy of your WDL to FireCloud’s method repository:

Below is the docker command to upload a copy of your WDL to FireCloud’s method repository. Note that the -s option specifies the Method Repository namespace in which to publish the WDL. **You will want to create your own namespace...replace “birger” with your own name.**

|  |
| --- |
| root@2393d001c81f:/working# **firecloud -u** [**https://api.firecloud.org/api**](https://api.firecloud.org/api) **-m push -s birger -n helloworld -t Workflow -y "Hello World for Tool Developers Workshop" helloworld.wdl**  Succesfully pushed. Reponse:  {  "name": "helloworld",  "createDate": "2016-08-09T19:15:28Z",  "payload": "workflow helloWorldWorkflow {\n\tString name\_WF\n\tcall helloWorldTask\n\t{\n\t\tinput: name\_T=name\_WF\n\t}\n}\n\ntask helloWorldTask {\n\tString name\_T\n\ncommand <<<\n# Note: \n# (1) must provide full pathname because working directory is not the root of the file system\n# (2) python and hello\_world.py are bundled into the docker image cbirger/hello-world:5.0\npython /hello\_world/hello\_world.py ${name\_T}\n>>>\n\noutput {\n\tFile Hello\_World\_OutputFile = \"hello\_world\_output.txt\"\n}\n\nruntime {\n\tdocker: \"cbirger/hello-world:6.0\"\n}\n\n}",  "url": "http://agora.dsde-prod.broadinstitute.org/api/v1/methods/birger/helloworld/6",  "documentation": "",  "synopsis": "Hello World for Tool Developers Workshop",  "entityType": "Workflow",  "snapshotId": 6,  "namespace": "birger"  }  root@2393d001c81f:/working# |

1. Go to Method Repository to see your uploaded method



**1.5 Run the tool within FireCloud**

You will now run the helloworld method on data you upload to a FireCloud workspace. You will create a new workspace, add participant entities to that workspace, create a method configuration in your workspace that maps the inputs and outputs of your helloworld method to participant entity attributes, and run the helloworld method on those workspace entities.

1. Create a new workspace by clicking on Create New Workspace... on the FireCloud portal’s main page. The workspace will be charged to the Google Project broad-firecloud-workshops. Workspace names within that project must be unique. Name your workspace helloworld\_<username>.



1. Upload participant and participant\_set entities to the new workspace
   * Go to the data tab, select Import Data... → Import from file… → Choose file...
   * Choose the participants.tsv TSV file provided in the zip archive emailed to all workshop attendees.
   * Select Choose file… again
   * Choose the participant\_set\_membership.tsv TSV file that is also provided in the zip archive





1. Create a Method Configuration

* Go to the Method Configurations tab and click Import Configuration...
* Filter on <your namespace>/helloworld and select the helloworld workflow you previously uploaded to the method repository.



* Select the desired Workflow, and then click Import



* Specify expressions for the workflow’s input and output parameters.
  + Set the expression for the helloWorldWorkflow.name\_WF input paramter to this.name
  + Set the expression for the the helloWorldWorkflow.helloWorldTask.Hello\_World\_OutputFile to this.greeting
  + Click Save

1. Launch the helloworld method (workflow) on a single participant

* Click Launch Analysis…
* Select the first participant in the list and click Launch



* Monitor the running workflow
  + Click on the single workflow in the monitor tab’s list of the workspace’s submitted workflows to drill down into the workflow’s status. Note that if the workflow’s status is queued, you will not be able to drill down; if this is the case, refresh the page until the Status becomes “Running”



* + The HelloWorld workflow should complete within minutes.



1. View the results

* Go to the Data tab and click on the first participant’s greeting attribute value
* Click Open

1. Launch the helloworld method (workflow) on all participants in a participant set

* Go to the Method Configuration Tab and select the helloworld method configuration
* Click on Launch Analysis…
* Click on participant\_set(1)
* Select the participant set ACC (it will become highlighted in yellow)
* In the Define Expression box, enter this.participants



* Click Launch

In most cases, you will see a listing of the five workflows (one for each participant) that have been queued for submission.



* Click on the Monitor tab, and then click on your most recent analysis submission. You will be able to see the current status of the five helloworld workflows.
* When all five workflows have the Done status, click on the data tab, and look at the participants table. You will see a listing of the five output files. Click on any of the output file hyperlinks to see its content.

# 2. Linear Chaining WDL

The purpose of this exercise is to demonstrate how to create the WDL “plumbing” that feeds the output of task #1 to the output of task #2, whose output, in turn, is fed into task #3. This results in a three-step workflow, where tasks #1 through #3 run sequentially. For expediency we will demonstrate this by running the workflow on the locally running version of cromwell rather than uploading and running it on FireCloud. The purpose of the exercise is to highlight how tasks outputs get fed into the inputs of downstream tasks.

You will find the wdl file (linear\_chain.wdl) in the linear\_chain\_wdl folder of the unzipped archive.

|  |
| --- |
| task addTwoTask {  String inputNum  command <<<  OUT\_NUM=$((${inputNum} + 2)) ;  echo $OUT\_NUM  >>>  output {  String outNum=read\_string(stdout())  }  runtime {  docker: "ubuntu:14.04.4"  }  }  task multiplyByFiveTask {  String inputNum  command <<<  OUT\_NUM=$((${inputNum} \* 5)) ;  echo $OUT\_NUM  >>>  output {  String outNum=read\_string(stdout())  }    runtime {  docker: "ubuntu:14.04.4"  }  }  task addThreeTask {  String inputNum  command <<<  OUT\_NUM=$((${inputNum} + 3)) ;  echo $OUT\_NUM  >>>  output {  String final=read\_string(stdout())  }    runtime {  docker: "ubuntu:14.04.4"  }  }  workflow calculatorWorkflow {  String inputNum  call addTwoTask {  input:  inputNum=inputNum  }  call multiplyByFiveTask {  input:  inputNum=addTwoTask.outNum  }  call addThreeTask {  input:  inputNum=multiplyByFiveTask.outNum  }  } |

There are several things to note with this wdl file:

* The workflow is trivial:
  + y = x + 2
  + z = y \* 5
  + q = z + 3
* A command is a *task section* that starts with the keyword 'command', and is enclosed in curly braces or <<< >>>. Sometimes a command is sufficiently long enough or might use { or } characters such that using a different set of delimiters would make it more clear. In this case, enclose the command in <<<...>>>
* Our docker image is an official base ubuntu image. Our command block is making a series of bash shell command calls; we are not running a java, R, python or matlab application that could be bundled into a custom docker image. Don’t focus on the bash commands...the important aspect of this file is how the output of one task is specified as the input of a downstream task.

**2.1 Test linear\_chain WDL running Cromwell locally**

1. Create json template for workflow inputs

|  |
| --- |
| wm8b6-23c:Workshop\_Exercises birger$ **java -jar ~/wdl/bin/wdltool-0.4.jar inputs ~/Workshop\_Materials/linear\_chain\_wdl/linear\_chain.wdl > linear\_chain\_inputs.json**  wm8b6-23c:Workshop\_Exercises birger$ **cat linear\_chain\_inputs.json**  {  "calculatorWorkflow.inputNum": "String"  }  wm8b6-23c:Workshop\_Exercises birger$ |

Using your text editor, edit linear\_chain\_inputs.json and replace “String” above with a number; e.g.,

"calculatorWorkflow.inputNum": "1"

1. Run WDL with local Cromwell

|  |
| --- |
| wm8b6-23c:Workshop\_Exercises birger$ **java -jar ~/wdl/bin/cromwell-0.19.3.jar run ~/Workshop\_Materials/linear\_chain\_wdl/linear\_chain.wdl linear\_chain\_inputs.json**  [2016-08-09 18:18:32,31] [info] Slf4jLogger started  [2016-08-09 18:18:32,72] [info] RUN sub-command  [2016-08-09 18:18:32,72] [info] WDL file: /Users/birger/Workshop\_Materials/linear\_chain\_wdl/linear\_chain.wdl  [2016-08-09 18:18:32,73] [info] Inputs: linear\_chain\_inputs.json  [2016-08-09 18:18:32,121] [info] SingleWorkflowRunnerActor: launching workflow  [2016-08-09 18:18:32,914] [info] Running with database db.url = jdbc:hsqldb:mem:e1369abc-db8a-4bc3-b304-954dc5a71529;shutdown=false;hsqldb.tx=mvcc  . . .  . . .  calculatorWorkflow.$final\_call$copy\_call\_logs to Done.  [2016-08-09 18:19:05,225] [info] WorkflowActor [29ee9c06]: Beginning transition from Running to Succeeded.  [2016-08-09 18:19:05,233] [info] WorkflowActor [29ee9c06]: transitioning from Running to Succeeded.  {  "calculatorWorkflow.addThreeTask.final": "18",  "calculatorWorkflow.addTwoTask.outNum": "3",  "calculatorWorkflow.multiplyByFiveTask.outNum": "15"  }  [2016-08-09 18:19:05,250] [info] SingleWorkflowRunnerActor workflow finished with status 'Succeeded'.  wm8b6-23c:Workshop\_Exercises birger$ |

Note that the outputs from each task in the workflow are displayed in a json object written to stdout.

# 3. Task Aliasing

You often will want to call the same task multiple times within a workflow. You will need to be able to distinguish between the different calls when referencing outputs. This is done through *task aliasing*. The following exercise provides a simple example of aliasing.

|  |
| --- |
| task addTwoTask {  String inputNum  command <<<  OUT\_NUM=$((${inputNum} + 2)) ;  echo $OUT\_NUM  >>>  output {  String outNum=read\_string(stdout())  }  runtime {  docker: "ubuntu:14.04.4"  }  }  task addNumsTask {  String inputNumOne  String inputNumTwo  command <<<  OUT\_NUM=$((${inputNumOne} + ${inputNumTwo})) ;  echo $OUT\_NUM > outFile.txt  echo "The numbers $IN\_NUM\_ONE and $IN\_NUM\_TWO were added!" > log.txt  >>>  output {  File outFile="outFile.txt"  String outNum=read\_string("outFile.txt")  File log="log.txt"  }    runtime {  docker: "ubuntu:14.04.4"  }  }  workflow TaskAliasingWorkflow {  String dataOne  String dataTwo  call addTwoTask as FirstAdder {  input:  inputNum=dataOne  }  call addTwoTask as SecondAdder {  input:  inputNum=dataTwo  }  call addNumsTask {  input:  inputNumOne=FirstAdder.outNum,  inputNumTwo=SecondAdder.outNum  }  } |

Note the following:

* The workflow is a sequence of trivial arithmetic operations:
  + y1 = x1 + 2
  + y2 = x2 + 2
  + z = y1 + y2
* As with the linear chaining example, the command statements are enclosed in <<< >>> rather than curly braces, the command block is a series of bash shell commands and the loaded docker image is an official base ubuntu image.
* The two calls to addTwoTask are independent of one another (they are not wired together); cromwell can schedule the two task instances to run in parallel.
* The addNumsTask has two input parameters and produces three outputs: a String representation of the sum and two files, a log file and the file containing the sum.

**3.1 Test task\_aliasing WDL running Cromwell locally**

1. Create json template for workflow inputs

|  |
| --- |
| wm8b6-23c:Workshop\_Exercises birger$ **java -jar ~/wdl/bin/wdltool-0.4.jar inputs ~/Workshop\_Materials/task\_aliasing\_wdl/task\_aliasing.wdl > task\_aliasing\_inputs.json**  wm8b6-23c:Workshop\_Exercises birger$ cat task\_aliasing\_inputs.json  {  "TaskAliasingWorkflow.dataOne": "String",  "TaskAliasingWorkflow.dataTwo": "String"  }  wm8b6-23c:Workshop\_Exercises birger$ |

Using your text editor, edit task\_aliasing\_input.json and replace Strings above with numbers; e.g.,

"TaskAliasingWorkflow.dataOne": "1"

"TaskAliasingWorkflow.dataTwo": "5"

1. Run WDL with local Cromwell

|  |
| --- |
| wm8b6-23c:Workshop\_Exercises birger$ **java -jar ~/wdl/bin/cromwell-0.19.3.jar run ~/Workshop\_Materials/task\_aliasing\_wdl/task\_aliasing.wdl task\_aliasing\_inputs.json**  [2016-08-10 08:55:07,966] [info] Slf4jLogger started  [2016-08-10 08:55:07,990] [info] RUN sub-command  [2016-08-10 08:55:07,991] [info] WDL file: /Users/birger/Workshop\_Materials/task\_aliasing\_wdl/task\_aliasing.wdl  [2016-08-10 08:55:07,991] [info] Inputs: task\_aliasing\_inputs.json  [2016-08-10 08:55:08,22] [info] SingleWorkflowRunnerActor: launching workflow  [2016-08-10 08:55:08,518] [info] Running with database db.url = jdbc:hsqldb:mem:f7dc941d-288b-428f-9f08-4d61ce126f1a;shutdown=false;hsqldb.tx=mvcc  [2016-08-10 08:55:11,689] [info] WorkflowManagerActor submitWorkflow input id = None, effective id = c8368e89-78c9-4f93-98ae-a37ee3a0b075  . . .  . . .  [2016-08-10 08:55:15,249] [info] WorkflowActor [c8368e89]: persisting status of TaskAliasingWorkflow.$final\_call$copy\_workflow\_outputs to Done.  [2016-08-10 08:55:15,257] [info] WorkflowActor [c8368e89]: Beginning transition from Running to Succeeded.  [2016-08-10 08:55:15,265] [info] WorkflowActor [c8368e89]: transitioning from Running to Succeeded.  {  "TaskAliasingWorkflow.SecondAdder.outNum": "7",  "TaskAliasingWorkflow.addNumsTask.outFile": "/Users/birger/Workshop\_Exercises/cromwell-executions/TaskAliasingWorkflow/c8368e89-78c9-4f93-98ae-a37ee3a0b075/call-addNumsTask/outFile.txt",  "TaskAliasingWorkflow.addNumsTask.outNum": "10",  "TaskAliasingWorkflow.addNumsTask.log": "/Users/birger/Workshop\_Exercises/cromwell-executions/TaskAliasingWorkflow/c8368e89-78c9-4f93-98ae-a37ee3a0b075/call-addNumsTask/log.txt",  "TaskAliasingWorkflow.FirstAdder.outNum": "3"  }  [2016-08-10 08:55:15,282] [info] SingleWorkflowRunnerActor workflow finished with status 'Succeeded'.  wm8b6-23c:Workshop\_Exercises birger$ |

Note the three outputs of the addNumsTask: outFile, outNum and log. The output files are written to the host file system. If running this workflow in FireCloud, the output files are written to the workspace’s Google Cloud Storage bucket.

# 

# 

# 4. Scatter/Gather Example

One may implement scatter/gather workflows in FireCloud using WDL’s scatter function. Using the scatter function, one can instruct FireCloud to produce parallelizable jobs running the same task on each input of an input array. The outputs produced are passed along to the next task as an array, without ever having to make an explicit declaration that the output is an array.

workflow scatter\_gather\_example {

call prepare

scatter (x in prepare.array) {

call analysis {

input: str=x

}

}

call gather {

input: array=analysis.out

}

output {

gather.str

}

}

4.1 Create a new workspace titled “test\_scatter\_gather\_<your name>”

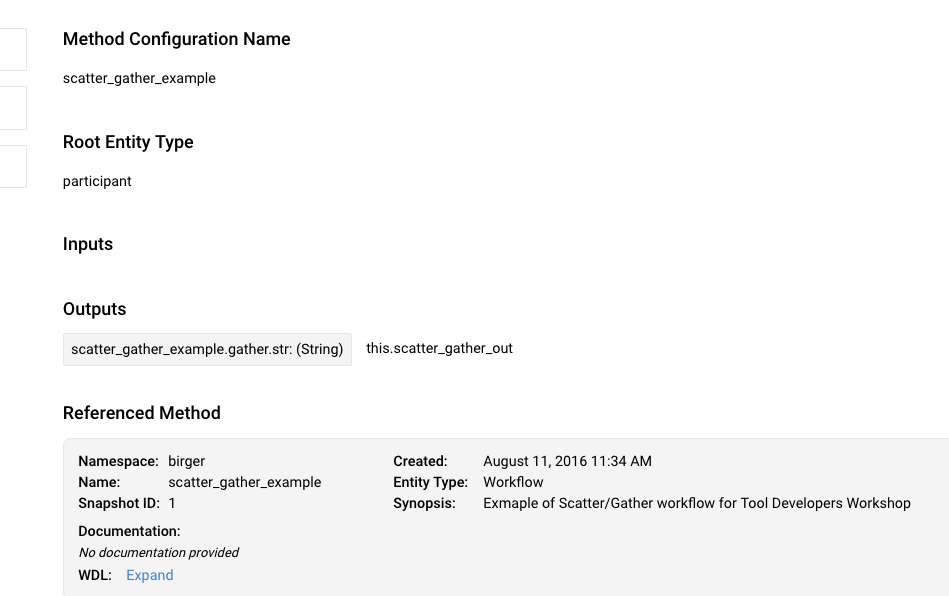
4.2 Upload the participant.tsv file, located in the Workshop\_Materials/scatter\_gatehr\_wdl/ directory of your expanded zip file.

4.3 Click on the Method Configurations tab, and then click on Import Configuration….

4.4 Filter on “scatter” and select the workflow birger/scatter\_gather\_example Snapshot ID: 1

4.5 Click on Import

4.6 Click on Edit this page, and make the following edits to the method config



4.6 Launch the Method Configuration against the participant any of the participants in the workspace’s data model. Note that in this example workflow, no inputs are drawn from the data model, but to run a workflow, it still needs to target a data model entity.

4.7 Monitor the workflow from the monitor tab. You should see four instances of the analysis task. This is the scattering operation. When all four analysis instances are completed, the gather task will launch and its input will be an array of the four outputs from the four analysis tasks.



# 5. Setting Runtime Parameters in the Method Configuration

The runtime block in a WDL task definition contains specifications of runtime attributes that customize the environment when the task is called. This table lists the currently available runtime attributes for cromwell:

|  |  |  |  |
| --- | --- | --- | --- |
| **Runtime Attribute** | **LOCAL** | **JES (FireCloud)** | **Description** |
| continueOnReturnCode | x | x | OPTIONAL  Normally a non-zero return code indicates a failure. You can override this behavior by specifying the continueOnReturnCode attribute. When set to false (the default), any non-zero return code will be considered a failure, When set to true, all return codes will be considered successful.  You may also set this to an integer (e.g., 1) or an array of integers (e.g., [0, 1]), and only those integers will be considered successful return codes  Defaults to false |
| cpu |  | x | OPTIONAL  Minimum number of processor cores to use. Defaults to 1. |
| disks |  | x | OPTIONAL  Disks to attach to docker container. Disks are specified as a comma separate list; each disk is represented as a space-delimited triplet:  local-disk <disk size in GB> [SSD | HDD]  For example:  runtime {  disks: “local-disk 100 HDD”  }  Defaults to local-disk 10 HDD |
| docker | x | x | MANDATORY  Specifies the docker image. |
| failOnStderr | x | x | OPTIONAL  Some programs write to stderr when there is an error, but still return a zero exit code. Set this attribute to true for these programs if you want a task to be viewed as failing if anything is written to stderr.  Defaults to false |
| memory |  | x | OPTIONAL  Minimum amount of RAM required by the task. JES will allocate a VM with at least this amount of RAM.  Example  runtime {  Memory: “4 GB”  } |
| preemptible |  | x | OPTIONAL  Set this to a nonzero integer if you want to use preemptible VMs, which cost a third of whan non-preemptible machines cost. The integer value indicates the maximum number of times Cromwell shoudl request a preemptible machine for a task before defaulting back to a non-preemptible one.  Defaults 0 |
| bootDiskSizeGb |  | x | OPTIONAL  Specifies the boot disk size. This is the disk where the the docker image itself is booted, not the working directory of your task on the VM. Its primary purpose is to ensure larger docker images can fit on the boot disk.  runtime {  bootDiskSizeBg: 50  }  Defaults to 10 GB |

Many of these you may want to hardcode in your WDL. For example, if your docker image is greater than 10 GB size, you will want to set the bootDiskSizeGb. Similarly, continueOnReturnCode and failOnStderr are tied to the program. Other attributes, however, may depend on the input data or your budget, and in these cases you may want to be able to set them through the method config. Attributes that fall into this category include memory, disk and preemptible. In the following example, we run a WDL workflow whose runtime attributes are set within the method configuration.

5.1 Create a new workspace titled “test\_runtime\_env\_<your name>”

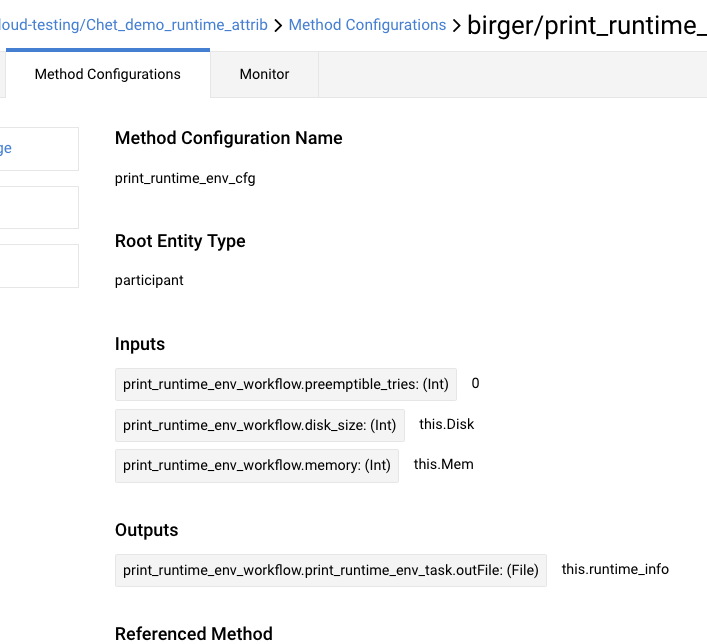
5.2 Upload the participant.tsv and the participant\_set\_membership.tsv files. Both are located in the Workshop\_Materials/runtime\_attributes\_wdl/ directory of your expanded zip file.

5.3 Click on the Method Configurations tab, and then click on Import Configuration….

5.4 Filter on “runtime” and select the workflow birger/print\_runtime\_env Snapshot ID: 6

5.5 Click on Import

5.6 Click on Edit this page, and make the following edits to the method config



5.6 Launch the Method Configuration against the participant set *test\_set*. Be sure to set the “Define Expression” field to this.participants. Five workflows will be launched, one for each participant in the set. Each workflow draws its Disk and Mem runtime parameters from the data model.

5.7 When all five workflows are Done, view the runtime\_info output file (bucket URL for these output files are written back to the data model) to see the Disk and Memory allocated to the VM in each workflow.

## SUPPLEMENTARY Material:

#### How to set up a New FireCloud Billing Project

#### Run ‘Workshop\_MiniMutationCalling’ via the command line

#### Glossary

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## 1. How to set up a New FireCloud Billing Project

In order for a FireCloud Administrator to create a **new** FireCloud Billing Project, you must first:

**1.** Set up a Google Billing Account, using one of the options below:

* Create your own Google Billing Account using a credit card or bank account.
* Talk to your institutional procurement office and see if they have a preferred account set up method with Google (such as a third party reseller or an existing account).
* Set up a Google Billing Account through a third party reseller. There are many options and two examples are: [**Onix Networking**](https://docs.google.com/document/d/1wfXx2QB981bbNJbTzxerHrZ-hScjdQL0SQxqwnmcK30/edit?ts=573f80d7) or [**Sada Systems**](https://drive.google.com/a/broadinstitute.org/file/d/0B3ze6VAAKcukenM1YlRRQWRTeG9kQXQ2a2FPdlZNOW5mcFVN/view?ts=5743a0d4). Third party resellers provide additional billing options at no extra cost.

**2.** Add **Google@broadinstitute.com** as a Billing Administrator to your Google Billing Account.

**3.** Request a new FireCloud Billing Project by submitting an [**Internal Broad Request Form**](https://broadinstitute.atlassian.net/secure/CreateIssue.jspa?pid=13201&issuetype=11400) or [**FireCloud Billing Project Request Form**](https://docs.google.com/forms/d/1GVw8-W9NHqWI_Ce2I7iuNlriIm8vD1ty48d86xBUhhk/viewform). This request should include your Google Billing Account ID. You can can remove **Google@broadinstitute.com** as a Billing Administrator after you receive confirmation that your FireCloud Billing Project has been created.

Go [**here**](https://software.broadinstitute.org/firecloud/guide/topic?name=firecloud-google) for more information. You can also email [Help@FireCloud.org](mailto:Help@FireCloud.org) if you have any questions or problems.

## 2. Run ‘Workshop\_MiniMutationCalling’ via the command line

**Note:** this does not include the steps to upload TSVs.

wm8b1-75c:May24Workshop esalinas$ cat mini\_mutation\_calling\_script.sh

#!/bin/bash

#set verbosity

set -x

#define paths for software

FISSFC\_PATH=/usr/local/bin/fissfc

PYTHON\_PATH=/usr/local/bin/python

##################################

# REQUIREMENTS

# 1) Google Cloud SDK is required <https://cloud.google.com/sdk/>

# 2) Having logged in with 'gcloud auth login'

# (using the google cloud SDK) is also required

# 3) fissfc is required (<https://pypi.python.org/pypi/fissfc/0.6.0>)

# 4) Python required to have a compatible and sufficiently recent openssl

# OpenSSL 1.0.2e 3 Dec 2015 or newer should suffice

echo -ne "import ssl\nprint(ssl.OPENSSL\_VERSION)"|$PYTHON\_PATH

#This script was developed using OpenSSL 1.0.2e 3 Dec 2015

#With some older versions of OpenSSL an error

# 'httplib2.SSLHandshakeError:

# [SSL: TLSV1\_ALERT\_PROTOCOL\_VERSION]

# tlsv1 alert protocol version (\_ssl.c:590)' may arise

#test connectivity

$FISSFC\_PATH ping

#2016-05-18T15:11:08.220+0000

CONN=`/usr/local/bin/fissfc ping 2>/dev/null |perl -ne 'print "success" if /^\d{4}\-\d{2}\-[A-Za-z0-9]{5}:\d+:\d+\.\d+\+\d+$/'|tail -1` ;

if [ "$CONN" = "success" ] ; then

echo "Ping seems to have been successful!" ;

#list workspaces

$FISSFC\_PATH space\_list

#define workspace names and namespaces

TUTORIAL\_WSNS="broad-firecloud-tutorials"

TUTORIAL\_WS="Workshop\_MiniMutationCalling"

TARGET\_WSNS="broad-firecloud-testing"

TARGET\_WS="eddie\_mmc2"

#clone the mini mutation workspace

echo -ne "To clone $TUTORIAL\_WSNS / $TUTORIAL\_WS to $TARGET\_WSNS / $TARGET\_WS ..." ;

$FISSFC\_PATH space\_clone $TUTORIAL\_WSNS $TUTORIAL\_WS $TARGET\_WSNS $TARGET\_WS ;

#list data in the workspace

echo -ne "import firecloud.api as fapi\nr,c=fapi.get\_entities\_with\_type(\"$TARGET\_WSNS\",\"$TARGET\_WS\")\nprint c"| $PYTHON\_PATH

#submit a job!

#against the HCC\_pairs

echo -ne "import firecloud.api as fapi\nfapi.create\_submission(\"$TARGET\_WSNS\",\"$TARGET\_WS\",\"workshop\",\"MiniMutationCalling\_Cfg\",\"HCC1954\_100\_gene\_pair\",\"pair\",None)"|$PYTHON\_PATH ;

echo -ne "import firecloud.api as fapi\nfapi.create\_submission(\"$TARGET\_WSNS\",\"$TARGET\_WS\",\"workshop\",\"MiniMutationCalling\_Cfg\",\"HCC1143\_WE\_pair\",\"pair\",None)"|$PYTHON\_PATH ;

#polling, polling, polling....for submission status!

DONE\_COUNT=0

until [ "$DONE\_COUNT" == "2" ] ; do

TOKEN=`gcloud auth print-access-token`

echo "Sleep a bit, then poll for submission status...." ;

sleep 30

DONE\_COUNT=`curl -X GET --header "Accept: application/json" --header "Authorization: Bearer $TOKEN" "[https://api.firecloud.org/api/workspaces/${TARGET\_WSNS}/${TARGET\_WS}/submissions](https://api.firecloud.org/api/workspaces/$%7BTARGET_WSNS%7D/$%7BTARGET_WS%7D/submissions)" 2>/dev/null |grep -i 'status'|grep -ic 'done'`

echo "DONE COUNT IS $DONE\_COUNT" ;

date

done ;

#list data in the workspace

echo -ne "import firecloud.api as fapi\nr,c=fapi.get\_entities\_with\_type(\"$TARGET\_WSNS\",\"$TARGET\_WS\")\nprint c"| $PYTHON\_PATH

else

echo "Ping seems to have been unsuccessful... abort!" ;

fi ;

## 3. Glossary

### Basic Concepts

**Analysis Submission**

A user submits an Analysis Submission to the workspace service when launching a method configuration against an entity set. An analysis submission is a combination of a method config and entity set; this combination identifies the method that will run, the number of times it will be run (the number of entities in the set), and the inputs and outputs for each run.

**BAM file**

An input unit consisting of tab-delimited text that contains sequence alignment data. It is the binary version of a SAM file.

**Controlled Access**

De-identified data that may be unique to individuals. FireCloud users with dbGaP-authorization and a linked eRA Commons account can access TCGA controlled access data.

**Data Model**

Organizes data and meta-data for workspaces and analysis runs. The data model includes predefined entity types (e.g., participants and sample sets), relationships, and entity attributes. For your convenience, results from analysis runs are populated directly to the data model. Currently, the data model is tailored to TCGA data, but will be extensible to non-TCGA projects with a germline or cell-line focus.

**Entity**

Refers to physical items (e.g., participants) or collections of physical items (e.g., participant sets). Entities provide organization and hierarchical structure for data. For example, a participant entity refers to a participant. A sample entity refers to a sample that may belong to that participant.

**Entity Attributes**

FireCloud uses entity attributes to describe data entities (e.g., a participant identifier) and reference entity file locations (e.g., the URL to a Google Cloud Storage bucket). Entity attributes can be fed into and populated from a workflow analysis.

**FireCloud RESTful API**

All functionality presented through the user interface is also available to users through a public-facing secure RESTful API. Comprehensive on-line documentation for this API is available at <https://api.firecloud.org>. This online documentation employs the Swagger representation of RESTful APIs. The FireCloud RESTful API’s endpoints are organized into the following categories:

* [Entities](https://api.firecloud.org/#!/Entities)
* [Method Configurations](https://api.firecloud.org/#!/Method_Configurations)
* [Method Repository](https://api.firecloud.org/#!/Method_Repository)
* [NIH](https://api.firecloud.org/#!/NIH)
* [OAuth](https://api.firecloud.org/#!/OAuth)
* [Profile](https://api.firecloud.org/#!/Profile)
* [Storage](https://api.firecloud.org/#!/Storage)
* [Submissions](https://api.firecloud.org/#!/Submissions)
* [Workspaces](https://api.firecloud.org/#!/Workspaces)

**Load Files (TSV Files)**

FireCloud uses tab-separated-value (TSV) files to import entities and entity attributes into the Data tab. Each line in the TSV file corresponds to an entity and must reference entities of the same type. The FireCloud Data Model supports the following entity types:

* Participant
* Sample
* Pair
* Participant Set
* Sample Set
* Pair Set

**Methods**

A WDL description of a task or workflow in FireCloud.

**Method Configurations (Method Configs)**

Bind data to Methods and specify which attributes to use as inputs and outputs to an analysis runs. You can specify attributes in Method Config output fields that will get updated with results from an analysis run.

**Method Repository**

Contains methods for analyzing data (workflows and their constituent tasks), and method configs. Tool developers can upload their own methods using the FireCloud Command Line Interface (CLI).

**Open Access**

Public de-identified data that *is* *not* unique to individuals. All FireCloud users can access open access TCGA data.

**Task**

In FireCloud methods and WDL, tasks refer to executable programs that are bundled into a Docker image.

**Workflow**

Workflows are comprised of one or more tasks and contain the method and the method input parameters. FireCloud submits both tasks and workflows to Google Job Execution System (JES) when you run an analysis.

**Workspace**

Computational sandbox in which a FireCloud user organizes genomic data and metadata into a data model.

**Workspace Access Controls (ACLs)**

Define permissions and enable the secure sharing of workspaces among FireCloud users. ACLs contain three access levels: READER, WRITER, and OWNER where each access level represents an expanded set of permissions from the previous.

**Workspace Attributes**

Globally accessible input values within a workspace. If you enter workspace attributes in the workspac**e Summary** tab, they can serve as inputs for any Method Config within your workspace.

### Google Cloud Platform Concepts

**Google Billing Account**

In order for a FireCloud Administrator to create a new FireCloud Google Project, you must first create a Google Billing Account. Google Billing Accounts are billed for cloud storage and compute costs that are tracked through FireCloud Google Projects. You will need to provide a bank account or credit card to set up a Google Billing Account, or use a Google Reseller for alternative payment options (e.g., cost objects).

**Google Cloud Storage Bucket**

Each workspace is associated with a single dedicated Google Cloud Storage bucket, created under the Google Project with which the workspace is created.

**Google Developers Console**

The Google Developers Console is the user interface for Google Cloud Platform. You can view buckets and bucket data and Google Project information through the Google Developers Console.

**Google Project**   
Every workspace is linked to a single Google Project that tracks all cloud storage and cloud compute costs incurred within that workspace. Only FireCloud administrators can create and grant you access to Google Projects for use in FireCloud.

**GSUTIL**

Google Cloud Storage’s command line utility. Use this to upload data and files to Google buckets.

### Tool Developer Concepts

**Cromwell**

Cromwell is the workflow execution service used to run and test WDL workflows. When creating WDL workflows, you can test on a local installation of the Cromwell execution engine prior to uploading and testing on FireCloud. Cromwell reads WDL, which describes executable tasks packaged into docker containers. Cromwell then calls Google’s Job Execution System to run the executable tasks packaged into docker containers.

**Docker**

FireCloud uses [Docker](http://www.docker.com/) to distribute tools and applications for use in its methods. Docker allows applications and their dependencies to be packaged into discrete runtime environments, called Docker containers.

**Docker Container**

Docker containers wrap software in a file system that can contain the dependencies to run your tools on FireCloud. These dependencies can include code, system tools, system libraries and anything you can install on a server, thus enabling portability of tools across operating systems.

**Docker Host**

Virtual machine on which containers are launched, managed with ‘docker-machine.’

**DockerHub**

DockerHub is a cloud-based registry service for Docker images. You can store and share your Docker images through repositories (repos), both public and private for use on FireCloud.

**Docker Image**

Docker images store software and operating systems.

**FireCloud Command Line Interface (CLI)**

This FireCloud CLI enables tool developers to push methods and method configurations to FireCloud.

**FireCloud Cookbook**

A newly developed tool that allows users to run most processes available in the FireCloud user interface through the command line. You can run Hands-on exercises from the workshop using this tool.

**FISSfc**

Contains bindings to the FireCloud RESTful API and allows users to script FireCloud tasks through the command line, bypassing the FireCloud user interface.

**WDL (Workflow Description Language)**

Workflow Description Language (WDL) is a language specifically designed for expressing genomics workflows. WDL workflows are represented in a way that can be read by humans and understood by Cromwell, the Workflow Execution Service that will run the specified tools to analyze data.