WDL 101: Running WDLs using Cromwell

December, 2022

Table of Contents

# About this Guide

This guide is intended to be an introduction for users at the Fred Hutch to using our pre-configured Cromwell resources to run WDL workflows using the Fred Hutch computing cluster.

# 1 Introduction

WDL is an open specification for a workflow description language that originated at the Broad but has grown to a much wider audience over time. WDL workflows can be run using an engine, which is software that interprets and runs your WDL on various high performance computing resources, such as SLURM (the Fred Hutch local cluster), AWS (Amazon Web Services), Google and Azure.

At the Fred Hutch we have configured a software from the Broad called Cromwell to allow us to run WDLs on our local cluster that then can be easily ported to other cloud based compute infrastructure when desired. This allows us to simplify our workflow testing and design, leverage WDL for smaller scale work that does not need the cloud, and can let users of all kinds manage their workflow work over time via this tool.

## 1.1 What is Cromwell?

Cromwell is a workflow engine (sometimes called a workflow manager) software developed by the Broad which manages the individual tasks involved in multi-step workflows, tracks job metadata, provides an API interface and allows users to manage multiple workflows simultaneously. Cromwell isn’t the only WDL “engine” that exists, but it is a tool that has been configured for use on the Fred Hutch gizmo cluster in order to make running workflows here very simple.

# 2 Getting Started with Cromwell

To get set up using Cromwell at the Fred Hutch, there is a setup process you need to do only one time in order to customize how Cromwell runs for you. This guide aims to help you get these two steps set up correctly so that in the future you can just run Cromwell directly.

## 2.1 Prerequisites

### 2.1.1 rhino Access

Either you need to have gotten access to use the Fred Hutch SLURM cluster prior OR have taken our “Cluster 101” course. You can find this **free** course [here](https://leanpub.com/courses/fredhutch/fredhutchcluster101). To follow this course, you’ll need to know how to get onto the rhino nodes. If this is not something you are familiar with, you also may want to read a bit more about the use of our cluster over at [SciWiki](https://sciwiki.fredhutch.org/) in the Scientific Computing section about Access Methods, and Technologies.

### 2.1.2 (Optional) AWS Credentials

If you want to run workflows on data stored in AWS S3, you’ll need to have [set up your AWS credentials](https://sciwiki.fredhutch.org/scicomputing/access_credentials/#amazon-web-services-aws) first.  
As of version 1.3 of the diy-cromwell-server configuration, if you have credentials, then the Cromwell server will be configured to allow input files to be directly specified using their AWS S3 url.  
However if you do not have AWS credentials or aren’t using data stored in AWS S3, then you don’t have to do anything.

## 2.2 Database Setup

One of the strengths of using Cromwell is that it can allow you to keep track of what you’ve already done so you don’t have to re-compute jobs that have already completed successfully even if some of your workflow fails. To leverage this feature, called “call caching” and also use our computing resources efficiently, our configuration of Cromwell will help you set up a Cromwell server for 7 days at a time and creates a database that persists forever that the Cromwell servers can communicate with. We have found that by using a MySQL database for your Cromwell server to store information in, it will run faster and be better able to handle simultaneous workflows while also making all the metadata available to you during and after the run.

### 2.2.1 Get Your Database Container

Thus for each user, we suggest setting up a database using the following steps. This only needs to be done one time per user.

1. Go to [DB4Sci](https://mydb.fredhutch.org/login) and see the Wiki entry for DB4Sci [here](https://sciwiki.fredhutch.org/scicomputing/store_databases/#db4sci--previously-mydb).
2. Login using your Fred Hutch credentials
3. Choose Create DB Container, and choose the MariaDB (MySQL) option.
4. Use the following:
   * Your DB Username can be whatever you want
   * Your DB Password should not be your Fred Hutch password
   * Provide a Contact name and your fredhutch.org email
   * Provide a Description (like “Cromwell database”)
   * Set Expected Life of DB: Choose “Long (36+ mo)”
   * Set Frequency of Backup: “Never”
5. Save the DB/Container Name, DB Username and DB Password somewhere handy as you will need them for the configuration step.
6. Click submit, a confirmation screen will appear (hopefully), and you’ll need to note which Port is specified. This is a 5 digit number currently. Write this down!!! If you have trouble with using DB4Sci, you can email scicomp@fredhutch.org and share the information or screen shots of what failed to get help.

Note: If you do not see this screen, you can go to the Manage Containers tab in the MyDB interface and scroll down for your container name, and you can look in the Port column.

At this point you should have a sticky note or something handy where you’ve put the DB Name, DB Username, DB Password and Port ready for the next steps.

### 2.2.2 Make Your Empty Database

Now you have a “container” in which to run a database, but the database itself does not yet exist. Again, this only needs to be done one time per user. You need to create an empty database in that container and then Cromwell will do the rest the first time you start up a server.

To create the database, you’ll need to go to Terminal and connect to rhino then enter the following (where you replace the entire <Port> including the < and > with the text, for example, 34567).

ml MariaDB/10.5.1-foss-2019b  
mysql --host mydb --port <Port> --user <username> --password

It will then prompt you to enter the DB password you specified during setup. Once you are are a “mysql>” prompt, you can do the following.

Note, we suggest you name the database inside the container the same as the container, but you cannot include dashes in your database name.

MariaDB [(none)]> create database <DB Name>;  
## It should do its magic - if it works it says:  
#Query OK, 1 row affected  
MariaDB [(none)]> exit  
#Bye

Now you’re ready to go and never have to set up the database part again and you can use this database to manage all your work over time!

## 2.3 Start up your first Cromwell server

Now that you’ve set up your database so it’s ready for Cromwell to talk to it and save your workflow information to it, you will customize Cromwell to work how you want it to work and you’ll be ready to use it!

### 2.3.1 Customize Your Configuration

To start up your first Cromwell server job, you first need to decide where you want to keep your Cromwell configuration files. This must be a place where rhino can access them, such as in your Home directory, which is typically the default directory when you connect to the rhinos. We suggest you create a cromwell-home folder (or whatever you want to call it) and follow these git instructions to clone it directly.

Then you will set up the customizations that you’re going to want for your server(s) by making user configuration file(s) in your cromwell-home or wherever you find convenient.

You can manage multiple Cromwell profiles this way by just maintaining different files full of credentials and configuration variables that you want.

To get started, do the following on rhino:

mkdir -p cromwell-home  
cd cromwell-home  
git clone --branch main https://github.com/FredHutch/diy-cromwell-server.git

Next you’ll want to move the cromUserConfig.txt template file you just downloaded into your main cromwell-home directory for customization and keeping in the future.

cp ./diy-cromwell-server/cromUserConfig.txt .  
## When you are first setting up Cromwell, you'll need to put all of your User Customizations into this `cromUserConfig.txt` which can serve as a template.   
## After you've done this once, you just need to keep the path to the file(s) handy for the future.

In cromUserConfig.txt there are some variables that allow users to share a similar configuration file but tailor the particular behavior of their Cromwell server to best suit them.  
The following text is also in this repo but these are the customizations you’ll need to decide on for your server.

################## WORKING DIRECTORY AND PATH CUSTOMIZATIONS ###################  
## Where do you want the working directory to be for Cromwell (note: this process will create a subdirectory here called "cromwell-executions")? Note, please include the leading and trailing slashes!!!   
### Suggestion: /fh/scratch/delete90/pilastname\_f/username/  
SCRATCHDIR=/fh/scratch/delete90/...  
  
## Where do you want logs about individual workflows (not jobs) to be written?  
## Note: this is a default for the server and can be overwritten for a given workflow in workflow-options.  
### Suggestion: ~/cromwell-home/workflow-logs  
WORKFLOWLOGDIR=~/cromwell-home/workflow-logs  
  
## Where do you want to save Cromwell server logs for troubleshooting Cromwell itself?  
### Suggestion: ~/cromwell-home/server-logs  
SERVERLOGDIR=~/cromwell-home/server-logs  
  
################ DATABASE CUSTOMIZATIONS #################  
## DB4Sci MariaDB details (remove < and >, and use unquoted text):  
  
CROMWELLDBPORT=...  
CROMWELLDBNAME=...  
CROMWELLDBUSERNAME=...  
CROMWELLDBPASSWORD=...  
  
## Number of cores for your Cromwell server itself - usually 4 is sufficient.   
###Increase if you want to run many, complex workflows simultaneously or notice your server is slowing down.  
NCORES=4  
  
## Length of time you want the server to run for.   
### Note: when servers go down, all jobs they'd sent will continue. When you start up a server the next time  
### using the same database, the new server will pick up whereever the previous workflows left off. "7-0" is 7 days, zero hours.  
SERVERTIME="7-0"

Note: For this server, you will want multiple cores to allow it to multi-task. Memory is less important when you use an external database. If you notice issues, the particular resource request for the server job itself might be a good place to start adjusting, in conjunction with some guidance from SciComp or the Slack [Question and Answer channel](https://fhbig.slack.com/archives/CD3HGJHJT) folks.

### 2.3.2 Kick off your Cromwell server

Now that you’ve configured your future Cromwell servers, you can kick off your first Cromwell server job. Go to rhino to your “cromwell-home” and do the following:

## You'll want to put `cromwell.sh` somewhere handy for future use, we suggest:  
cp ./diy-cromwell-server/cromwell.sh .  
  
## Then you'll want to make the script "executable":  
chmod +x cromwell.sh  
  
# Then simply start up Cromwell by executing the script and passing it the path to your configuration file.   
./cromwell.sh cromUserConfig.txt

Much like the grabnode command you may have used previously, the script will run and print back to the console instructions once the resources have been provisioned for the server. You should see something like this:

Your configuration details have been found...  
Getting an updated copy of Cromwell configs from GitHub...  
Setting up all required directories...  
Detecting existence of AWS credentials...  
Credentials found, setting appropriate configuration...  
Requesting resources from SLURM for your server...  
Submitted batch job 2733799  
Your Cromwell server is attempting to start up on node/port gizmob5:39071. It can take up to 2 minutes prior to the port being open for use by the shiny app at https://cromwellapp.fredhutch.org or via the R package fh.wdlR. If you encounter errors, you may want to check your server logs at /home/username/cromwell-home/server-logs to see if Cromwell was unable to start up.  
Go have fun now.

IMPORTANT NOTE: Please write down the node and port it specifies here - in this example it’s gizmob5:39071, but yours will be a different combination. This is the only place where you will be able to find the particular node/port for this instance of your Cromwell server, and you’ll need that to be able to send jobs to the Cromwell server. If you forget it, scancel the Cromwell server job and start a new one.

While your server will normally stop after 7 days (the default), at which point if you have jobs still running you can simply restart your server and it will reconnect to existing jobs/workflows. However, if you need to take down your server for whatever reason before that point, you can go to rhino and do:

## Here `username` is your Fred Hutch username  
squeue -u username  
## Or if you want to get fancy:  
squeue -o '%.18i %.9P %j %.8T %.10M %.9l %.6C %R' -u username

You’ll see a jobname “cromwellServer”. Next to that will be a JOBID. In this example the JOBID of the server is 2733799.

If you ever want to shut down your server before the 7 day default run time, you can always go to Rhino in your Terminal and end the server by doing:

scancel 2733799

## 2.4 Starting up your server in the future

Good news! The above instructions are a one time event. In the future, when you want to start up a Cromwell server to do some computing work, all you’ll have to do is: 1. Get onto Rhino in Terminal 2. Change to the cromwell-home directory you made 3. Enter: ./cromwell.sh cromUserConfig.txt and you’re off to the races!

Congrats you’ve started your first Cromwell server!! Now on to how to submit a WDL workflow to it.

# 3 Using Shiny to Manage Workflows

Now that you’ve configured your first Cromwell sever, let’s submit some test workflows to it using the Fred Hutch Shiny app!

Note: Especially the first time you set up a Cromwell server, it will be busy for a few minutes setting up the database and doing all the work behind the scenes for you. Once it’s “ready” to listen for workflows it will start “listening” for instructions via the Shiny app (or other methods we’ll discuss later in the course). It may take 2-3 minutes before you can follow the rest of these instructions the first time. The time it takes is much shorter in the future (more like ~1 minute).

You can find our Fred Hutch Shiny app here: <https://cromwellapp.fredhutch.org/>

This shiny app will let you use a graphic interface to submit and manage workflows you’ve written in WDL.

## 3.1 Get Test Workflows

Here you’ll see a series of sections that will allow you to do several things. In this guide we’ll use a number of example workflows found in the dl-test-workflows GitHub repository that can be viewed and cloned from [GitHub](https://github.com/FredHutch/wdl-test-workflows).

Each of these example workflows is in a folder containing a WDL file (specifying the workflow itself), and any input files that you’ll need (in JSON format). There is [emerging documentation about the WDL specification itself being generated by the openWDL community here](https://wdl-docs.readthedocs.io/en/1.0.0/). Also, there is some useful, though very detailed, information in the [openWDL GitHub repo for the specification itself where you can learn more](https://github.com/openwdl/wdl/blob/main/versions/development/SPEC.md#introduction).

## 3.2 Login

While this Shiny app runs all the time, in order for it to know where to look for your particular information, you’ll need to to “login” by clicking the “Connect to Server” button on the left.

When you click “Connect to Server”, a box will appear where you will input the node:port combination you were assigned when you [started up your Cromwell server](https://hutchdatascience.org/FH_WDL101_Cromwell/getting-started-with-cromwell.html#kick-off-your-cromwell-server) (it will look something like this: gizmob5:39071).

If your server is not ready to listen for workflows you may see this error:

If so, just wait 1-2 more minutes (if it’s the first time you’ve set up a server, or less if it’s a future instance) and try again. Once the Shiny app can talk to your sever, you’ll see this result screen:

## 3.3 Submit Jobs Tab

Once you’ve connected your server to the Shiny app, you can start by using the “Submit Jobs” tab on the left.

### 3.3.1 Validate a workflow

This checks the format of your workflow files to make sure you have a valid file in a known format that Cromwell can interpret. This is called a “dry run” to ensure that your tasks are wired up correctly, but Cromwell does not try to see if any of your inputs are actually available, only that it can interpret what you told it. One of the reasons this is is that since Cromwell can pull files from local filesystems, AWS S3, Google buckets and Azure blobs, the process to test it’s ability to actually get your inputs will happen while you run the workflow the first time. Luckily, Cromwell will only get file inputs it needs at that moment, and if it can’t it won’t do that specific task (but can continue with the other tasks it can do!).

### 3.3.2 Submit a workflow

This will let you upload the files that contain your workflow description (a WDL), and up to two different sets of input lists (in JSON format). You can run a workflow with no input JSON, one input JSON, or two input JSONs (which will be concatenated or the second will overwrite the first if the same variable is declared in both). You can upload a workflow options JSON, as well as providing text labels of your choosing to workflows if you’d like.

When you click that “Submit Workflow” button, you’ll see confirmation in a new box that appears with the workflow submission ID and status. These IDs are long strings that look something like this: 4e7e244a-d6b1-41db-a324-45229ff34b00 and they’re useful if, for example, you want to abort a workflow, or identify it in the “Track jobs” tab. This workflow id string is unique to an individual workflow run, so if you run the same workflow a second time, you’ll get a different string. This means that this unique identifier string can be used to help understand the data source file(s) used to generate each set of results files, helping make your work reproducible.

### 3.3.3 Abort a workflow

Sometimes you realize you want to kill a workflow. Using the workflow submission id, you can kill specific workflows using this box. Note it will take Cromwell some time to coordinate SLURM job cancellations, but it will clean everything up for you.

### 3.3.4 Troubleshoot a workflow

Especially in the beginning if you have catastrophic workflow failures and you can’t even figure out what’s going on, you can come back to this Troubleshoot box to retrieve the entire, unformatted JSON output of all metadata Cromwell has about your workflow. You probably are better served by the “Track Jobs” tab for checking how your workflow is going, but if there’s nothing there that’s helpful, then this box is where you’ll want to go.  
> Note: this output is not for the faint of heart, but it will give you hints once you get used to understanding what Cromwell is telling you.

## 3.4 Track Jobs Tab

Once you’ve submitted a workflow, you’ll want to track how it’s going in the Track Jobs tab.

### 3.4.1 History of workflows

At the top, you’ll see that you can display as many days of workflow history as you’d like, filter that result for workflows with a specific name or with specific status(es) like ‘failed’, ‘succeeded’, etc. This can help if you have submitted a LOT of workflows and you don’t want to see them all, or if the Cromwell server is still busy working through all of your submissions and recording their status.

Once you click “Update View”, the revelant workflows will be returned and you’ll see various information on those workflows.

First, there’s a “Workflows Run” plot, showing how long each workflow ran for, and status for each.

Underneath, you’ll see a “Cromwell Overview” table showing metadata for each workflow. Click on the workflow you’re interested in to populate the rest of the tables (below).

### 3.4.2 Diving into a Workflow

Once you’ve selected a workflow row, you’ll see some summary information about that workflow.

You can see a plot of the timing and outcomes of all the calls in that workflow.

Then there is a table of each call containing useful information such as the directory where the job is working (callRoot), its SLURM job ID, what computing resources or software environment were used, and the job’s status.

Then you can use the Job Failures and Call Caching tables to retrieve information relevant to those processes by clicking the “Get/Refresh … Metadata” buttons (sometimes these can be quite large, and thus they do not load until you want them).

Finally, once a workflow succeeds, Cromwell can tell you (and this Shiny app can help you download) a table showing where to find the workflow outputs (note this is not every file created, only the ones you specify as “results” using the WDL file’s ‘workflow output’ block). This lets you find output files and interact with them, archive them, or otherwise copy them to longer term storage for use.

## 3.5 Run Test Workflows

Now that you know how to use the app, it’s time to run a test workflow.

We have curated some basic workflows that you can use to test whether your Cromwell server is set up correctly and to let you to play with Cromwell. Once your server is up, run through the examples in our [Test Workflow folder](https://github.com/FredHutch/diy-cromwell-server/tree/main/testWorkflows).

Note: For test workflows that use Docker containers, the first time you run them you may notice that jobs aren’t being sent very quickly. That is because for our cluster, we need to convert those Docker containers to something that can be run by Singularity. The first time a Docker container is used, it must be converted, but in the future Cromwell will used the cached version of the Docker container and jobs will be submitted more quickly.

# 4 Fred Hutch Customizations

Cromwell can help run WDL workflows on a variety of computing resources such as SLURM clusters (like the Fred Hutch cluster), as well as AWS, Google and Azure cloud computing systems. Using WDL workflows allows users to focus on their workflow contents rather than the intricacies of particular computing platform.  
However, there are optimizations of how those workflows run that may be specific to each computing tool (“task”).

We’ll discuss some of the available customizations to help you run WDLs on our cluster that still allow those workflows to be portable to other computing platforms.

## 4.1 Standard Runtime Variables

These runtime variables can be used on any computing platform, and the values given here are the defaults for our Fred Hutch configuration if there is a default set.

* cpu: 1
  + An integer number of cpus you want for the task.
* memory: 2000
  + An integer number of MB of memory you want to use for the task
* docker:
  + A specific Docker container to use for the task. An example of the value for this variable is: "ubuntu:latest". No default container is specified in our configuration and this runtime variable should only be used if/when a task should be run inside a docker container, in which case you’ll want to specify both the container name and specific version. If left unset or left out of the runtime block of a task completely, the Fred Hutch configuration will run the task as a regular job and not use docker containers at all. For the custom Hutch configuration, docker containers can be specified and the necessary conversions (to Singularity) will be performed by Cromwell (not the user). Note: when docker is used, soft links cannot be used in our filesystem, so workflows using very large datasets may run slightly slower due to the need for Cromwell to copy files rather than link to them.

## 4.2 Fred Hutch Custom Runtime Variables

For the gizmo cluster, the following custom runtime variables are available (below we show each variable with its current default value). You can change these variables in the runtime block for individual tasks in a WDL file: task-level specifications will override the defaults.

* walltime: "18:00:00"
  + A string (“HH:MM:SS”) that specifies how much time you want to request for the task. Can also specify >1 day, e.g. “1-12:00:00” is 1 day+12 hours.
* partition: "campus-new"
  + Which cluster partition to use. The default is campus-new: other options currently include restart or short but check [SciWiki](https://sciwiki.fredhutch.org/scicomputing/) for updated information.
* modules: ""
  + A space-separated list of the environment modules you’d like to load (in that order) prior to running the task. See below for more about software modules.
* dockerSL:
  + This is a custom configuration for the Hutch that allows users to use docker and softlinks only to specific locations in Scratch. It is helpful when working with very large files. An example of the value for this variable is: "ubuntu:latest". Just like the docker: runtime variable, only specify this if you want the task to run in a container (otherwise the default will be a non-containerized job).
* account:
  + This allows users who run jobs for multiple PI accounts to specify which account to use for each task, to manage cluster allocations. An example of the value for this variable is "paguirigan\_a", following the pilastname\_f pattern.

## 4.3 Software environments

### 4.3.1 Modules

At Fred Hutch we have huge array of pre-curated software modules installed on our SLURM cluster which you can [read about in SciWiki](https://sciwiki.fredhutch.org/scicomputing/compute_scientificSoftware/).  
The custom configuration of our Cromwell server allows users to specify one or more modules to use for individual tasks in a workflow. The desired module(s) can be requested for a task in the runtime block of your calls like this:

runtime {  
 modules: "GATK/4.2.6.1-GCCcore-11.2.0 SAMtools/1.16.1-GCC-11.2.0"  
}

In this example, we specify two modules, separated by a space (with quotes surrounding both). The GATK module will be loaded first, followed by the SAMtools module. In this example you’ll note the “toolchain” used to build each modules is the same (“GCC-11.2.0”). When you load >1 module for a single task it is important to ensure that they are compatible with each other: choose versions built with the same toolchain if you can.

### 4.3.2 Docker

However, if you want to move your WDL workflow to the cloud in the future, you’ll want to leverage Cromwell’s ability to run your tasks in Docker containers. Users can specify docker containers in runtime blocks. Cromwell will maintain a local cache of previously used containers, facilitating the pull of Docker containers and conversion for use. This behavior allows us to evade rate-limiting by DockerHub and improves speed of your workflows.

## 4.4 Guidance and Support

There are a variety of resources on campus from the [SciWiki](https://sciwiki.fredhutch.org) to SciComp office hours (found in the Research Data Support Teams team) to asking computing related questions in the [Fred Hutch Data Slack #question-and-answer channel](https://fhdata.slack.com/archives/CD3HGJHJT) to coming to the [Effective Computing Drop in hours held by Fred Hutch DaSL](https://fhdata.slack.com/archives/C03T4QPNDA9).

### 4.4.1 Effective Computing Drop In Hour

This is a drop-in style biweekly meeting that provides resources for folks to talk about their computational work and improve over time. Find current events by checking out the Fred Hutch Data Slack workspace [here](https://fhdata.slack.com).

### 4.4.2 Slack Workflow Manager Channel

In the [Fred Hutch Data Slack workspace](https://fhdata.slack.com/) the [#workflow-managers](https://fhdata.slack.com/archives/CJFP1NYSZ) channel is a good place to find help from Hutch peers and staff who support Cromwell and WDL

# 5 Using the fh.wdlR Package

Chapter 3 showed you how to use the Hutch Shiny app to submit workflows to running Cromwell servers, and to monitor their progress. The Shiny app is built using an R package (fh.wdlR) available via GitHub.

<https://github.com/FredHutch/fh.wdlR>

You can also use this R package through R/RStudio on your local machine (on VPN or on campus) to directly submit workflows to your Cromwell server from the R command line, and to track calls and workflow execution status directly.

## 5.1 Install fh.wdlR from GitHub

You will need the following packages installed first:

install.packages(pkgs = c("httr", "jsonlite", "magrittr",  
 "dplyr", "ssh", "purrr", "paws", "tidyr"))  
  
# Not required for the package but certainly handy and used in our demo here:  
install.packages("tidyverse")

Then you can install the most recent version of fh.wdlR by:

require(remotes)  
remotes::install\_github('FredHutch/fh.wdlR')

Install a specific release version (in this case v2.0.2) by:

require(remotes)  
remotes::install\_github('FredHutch/fh.wdlR@v2.0.2')

## 5.2 Example workflow process

## Load packages  
library(fh.wdlR); library(tidyverse);

Tell your R session how to find your Cromwell server (note you’ll need to be on campus or on VPN).

## Set your Cromwell URL  
setCromwellURL(nodeAndPort = "gizmoXXX:20202")

### 5.2.1 Validate your workflow formatting

list.files(pattern = "\*.wdl")  
valid <- cromwellValidate(WDL = "myworkflow.wdl"); valid[["errors"]]

Go fix your issues (if there are any), now send your workflow to Cromwell.

### 5.2.2 Submit Workflows

thisJob <- cromwellSubmitBatch(WDL = "myworkflow.wdl",  
 Params = "myworkflow-parameters.json",  
 Batch = "myworkflow-batch.json",  
 Options = "workflow-options.json")  
  
# thisJob$id is now the unique Cromwell ID for your entire workflow - you can use that to request all sorts of metadata!!!  
thisOne<- thisJob$id; thisOne

Now get all your metadata and track the workflow!!

### 5.2.3 Track Workflows

# Returns a data frame of all jobs run in the past number of days (uses your database)  
jobs <- cromwellJobs(days = 2)  
  
# Returns a data frame (one line if you only submit one workflow id) containing workflow level metadata  
w <- cromwellWorkflow(thisOne)  
  
# This is handy to print the current status of the workflow(s) is(are)  
w$status  
  
# Returns a data frame containing all call level metadata  
c <- cromwellCall(thisOne)  
  
# Handy set of dplyr commands to tell you about how the various calls are doing  
c %>% group\_by(callName, executionStatus) %>% summarize(status = n()) %>% arrange(executionStatus)  
  
# Returns a data frame containing call level call caching metadata  
ca <- cromwellCache(thisOne)  
  
# Handy set of dplyr commands to tell you about what sort of call caching is happening  
ca %>% group\_by(callCaching.hit, callName) %>% summarize(hits = n())  
  
# Opens up a popup in your browser with a timing diagram in it.  
cromwellTiming(thisOne)  
  
# Returns a data frame containing call level failure metadata  
f <- cromwellFailures(thisOne)  
  
# Will tell Cromwell to abort the current workflow - note this cannot be undone and it will take a while to stop all the jobs.   
abort <- cromwellAbort(thisOne)  
  
# When a workflow is done, request information about the workflow outputs.  
out <- cromwellOutputs(thisOne)

## 5.3 Look Under the Hood

When all else fails, pick through the ugly metadata yourself to see what’s happening.

# Ugly list of raw metadata should you need it for workflow troubleshooting  
WTF <- cromwellGlob(thisOne); WTF[["failures"]]

# About the Authors

These credits are based on our [course contributors table guidelines](https://www.ottrproject.org/more_features.html#giving-credits-to-contributors).

| Credits | Names |
| --- | --- |
| **Pedagogy** |  |
| Lead Content Instructor(s) | Amy Paguirigan |
| Content Editor(s)/Reviewer(s) | Checked your content |
| **Technical** |  |
| Template Publishing Engineers | [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/) |
| Publishing Maintenance Engineer | [Candace Savonen](https://www.cansavvy.com/) |
| Technical Publishing Stylists | [Carrie Wright](https://carriewright11.github.io/), [Candace Savonen](https://www.cansavvy.com/) |
| Package Developers ([ottrpal](https://github.com/jhudsl/ottrpal)) [Candace Savonen](https://www.cansavvy.com/), [John Muschelli](https://johnmuschelli.com/), [Carrie Wright](https://carriewright11.github.io/) |  |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value   
## version R version 4.0.2 (2020-06-22)  
## os Ubuntu 20.04.3 LTS   
## system x86\_64, linux-gnu   
## ui X11   
## language (EN)   
## collate en\_US.UTF-8   
## ctype en\_US.UTF-8   
## tz Etc/UTC   
## date 2022-12-28   
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date lib source   
## assertthat 0.2.1 2019-03-21 [1] RSPM (R 4.0.3)   
## bookdown 0.24 2022-02-15 [1] Github (rstudio/bookdown@88bc4ea)   
## callr 3.4.4 2020-09-07 [1] RSPM (R 4.0.2)   
## cli 2.0.2 2020-02-28 [1] RSPM (R 4.0.0)   
## crayon 1.3.4 2017-09-16 [1] RSPM (R 4.0.0)   
## desc 1.2.0 2018-05-01 [1] RSPM (R 4.0.3)   
## devtools 2.3.2 2020-09-18 [1] RSPM (R 4.0.3)   
## digest 0.6.25 2020-02-23 [1] RSPM (R 4.0.0)   
## ellipsis 0.3.1 2020-05-15 [1] RSPM (R 4.0.3)   
## evaluate 0.14 2019-05-28 [1] RSPM (R 4.0.3)   
## fansi 0.4.1 2020-01-08 [1] RSPM (R 4.0.0)   
## fs 1.5.0 2020-07-31 [1] RSPM (R 4.0.3)   
## glue 1.6.1 2022-01-22 [1] CRAN (R 4.0.2)   
## htmltools 0.5.0 2020-06-16 [1] RSPM (R 4.0.1)   
## knitr 1.33 2022-02-15 [1] Github (yihui/knitr@a1052d1)   
## lifecycle 1.0.0 2021-02-15 [1] CRAN (R 4.0.2)   
## magrittr 2.0.2 2022-01-26 [1] CRAN (R 4.0.2)   
## memoise 1.1.0 2017-04-21 [1] RSPM (R 4.0.0)   
## pkgbuild 1.1.0 2020-07-13 [1] RSPM (R 4.0.2)   
## pkgload 1.1.0 2020-05-29 [1] RSPM (R 4.0.3)   
## prettyunits 1.1.1 2020-01-24 [1] RSPM (R 4.0.3)   
## processx 3.4.4 2020-09-03 [1] RSPM (R 4.0.2)   
## ps 1.3.4 2020-08-11 [1] RSPM (R 4.0.2)   
## purrr 0.3.4 2020-04-17 [1] RSPM (R 4.0.3)   
## R6 2.4.1 2019-11-12 [1] RSPM (R 4.0.0)   
## remotes 2.2.0 2020-07-21 [1] RSPM (R 4.0.3)   
## rlang 0.4.10 2022-02-15 [1] Github (r-lib/rlang@f0c9be5)   
## rmarkdown 2.10 2022-02-15 [1] Github (rstudio/rmarkdown@02d3c25)  
## rprojroot 2.0.2 2020-11-15 [1] CRAN (R 4.0.2)   
## sessioninfo 1.1.1 2018-11-05 [1] RSPM (R 4.0.3)   
## stringi 1.5.3 2020-09-09 [1] RSPM (R 4.0.3)   
## stringr 1.4.0 2019-02-10 [1] RSPM (R 4.0.3)   
## testthat 3.0.1 2022-02-15 [1] Github (R-lib/testthat@e99155a)   
## usethis 2.1.5.9000 2022-02-15 [1] Github (r-lib/usethis@57b109a)   
## withr 2.3.0 2020-09-22 [1] RSPM (R 4.0.2)   
## xfun 0.26 2022-02-15 [1] Github (yihui/xfun@74c2a66)   
## yaml 2.2.1 2020-02-01 [1] RSPM (R 4.0.3)   
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library