fluid-cicb Documentation

Release v0.0.0

Joe Schoonover

Oct 29, 2021

QUICK START

Fluid Run is a tool for running high performance computing (HPC) and research computing (RC) applications on ephemeral resources on Google Cloud. The motivation for developing fluid-run is to support continuous integration and continuous benchmarking (CI/CB) of HPC and RC applications at scale on Google Cloud. By using fluid-run as a build step with Google Cloud Build, developers can automate running tests on GPU accelerated and multi-VM platforms hosted on Google Cloud. Information about the each test, including the system architecture, software version (git sha), build id, and application runtime are recorded and can be saved to Big Query. This allows developers to create an auditable trail of data that comments on the performance of an application over time and accross various hardware.

CHAPTER

ONE

EPHEMERAL RCC TUTORIAL

Fluid Run can be used to create ephemeral compute resources for testing HPC applications and to record information about the test for later analysis. This quickstart guide will introduce you to the necessary ingredients for configuring application tests with fluid-run, using an ephemeral Research Computing Cluster (RCC).

1.1 Demo

You will start by using the rcc-ephemeral example provided in the fluid-run repository. This example creates a Singularity image with the *cowsay* program installed on it and then runs tests for this image on an ephemeral RCC cluster. You will learn how to set up your Google Cloud project and create the necessary resources to support application testing, benchmarking, and logging.

1.1.1 Google Cloud Project Setup

To complete this tutorial, you will need to have an active project on Google Cloud. Sign up and create your first project by visiting https://console.cloud.google.com

Once your project is ready, open Cloud Shell

You will need to activate the following Google Cloud APIs

- Compute Engine
- Cloud Build
- Big Query
- Identity & Access Management (IAM)

```
gcloud config set project PROJECT-ID
gcloud services enable compute.googleapis.com
gcloud services enable bigquery.googleapis.com
gcloud services enable iam.googleapis.com
gcloud services enable cloudbuild.googleapis.com
```

1.1.2 Create a fluid-run Docker image

The fluid-run application is a Cloud Build builder. A Cloud builder is a Docker image that provides an environment and entrypoint application for carrying out a step in a Cloud Build pipeline. You can create the fluid-run docker image and store it in your Google Cloud project's Container Registry.

Open Cloud Shell and clone the fluid-run repository.

```
$ git clone https://github.com/FluidNumerics/fluid-run.git ~/fluid-run
```

Once you've cloned the repository, navigate to the root directory of the repo and trigger a build of the docker image.

```
$ cd ~/fluid-run/
$ gcloud builds submit . --config=ci/cloudbuild.yaml --substitutions=SHORT_SHA=latest
```

This will cause Google Cloud build to create the fluid-run docker image `gcr.io/${PROJECT_ID}/fluid-run:latest` that you can then use in your project's builds.

1.1.3 Create the CI/CB Dataset

The CI/CB dataset is a Big Query dataset that is used to store information about each test run with fluid-run. This includes runtimes for each execution command used to test your application. The fluid-run repository comes with a terraform module that can create this dataset for your project. We've also included an example under the `examples/rcc-ephemeral` directory that you will use for the rest of this tutorial.

Navigate to the `examples/rcc-ephemeral/ci/build_iac` directory

```
$ cd ~/fluid-run/examples/rcc-ephemeral/ci/build_iac
```

The `ci/build_iac` subdirectory contains the Terraform infrastructure as code for provisioning a VPC network, firewall rules, service account, and the Big Query dataset that all support using fluid-run. This example Terraform module is a template for creating these resources, and the `fluid.auto.tfvars` file in this directory is used to concretize certain variables in the template, so that you can deploy the resources in your project.

Open `fluid.auto.tfvars` in a text editor and set <project> to your Google Cloud Project ID. The command below will quickly do the search and replace for you.

```
$ sed -i "s/<project>/$(gcloud config get-value project)/g" fluid.auto.tfvars
```

Now, you will execute a workflow typical of Terraform deployments to initialize, validate, plan, and deploy. All of the commands are shown below, but you should review the output from each command before executing the next.

```
$ terraform init
$ terraform validate
$ terraform plan
$ terraform apply --auto-approve
```

Once this completes, you're ready to run a build using fluid-run.

1.1.4 Manually Trigger a build

Cloud Build pipelines for a repository are specified in a build configuration file written in YAML syntax. In this example, three build steps are provided that create a Docker image, create a Singularity image, and test the Singularity image on an ephemeral RCC cluster. A singularity image is created since, currently, fluid-run only supports testing of GCE VM images and Singularity images. However, as you can see, Singularity can convert a Docker image to a Singularity image that can be passed to fluid-run.

```
steps:

- id: Build Docker Image
  name: 'gcr.io/cloud-builders/docker'
  args: ['build',
         '.',
         '-t',
         'gcr.io/${PROJECT_ID}/cowsay:latest'
  ]

- id: Build Singularity Image
  name: 'quay.io/singularity/singularity:v3.7.1'
  args: ['build',
         'cowsay.sif',
         'docker-daemon://gcr.io/${PROJECT_ID}/cowsay:latest']

- id: CI/CB
  name: 'gcr.io/research-computing-cloud/fluid-run'
  args:
  - '--build-id=${BUILD_ID}'
  - '--git-sha=${COMMIT_SHA}'
  - '--surface-nonzero-exit-code'
  - '--artifact-type=singularity'
  - '--singularity-image=cowsay.sif'
  - '--image=${_IMAGE}'
  - '--project=${PROJECT_ID}'
  - '--zone=${_ZONE}'
  - '--cluster-type=rcc-ephemeral'
  - '--rcc-tfvars=ci/fluid.auto.tfvars'
  - '--save-results'

timeout: 1800s

substitutions:
  _ZONE: 'us-west1-b'
  _IMAGE: 'projects/research-computing-cloud/global/images/family/rcc-centos-7-v3'
```

To manually trigger a build, you can use the `gcloud builds submit` command in your cloud shell. Navigate to the `rcc-ephemeral` example directory, and submit the build

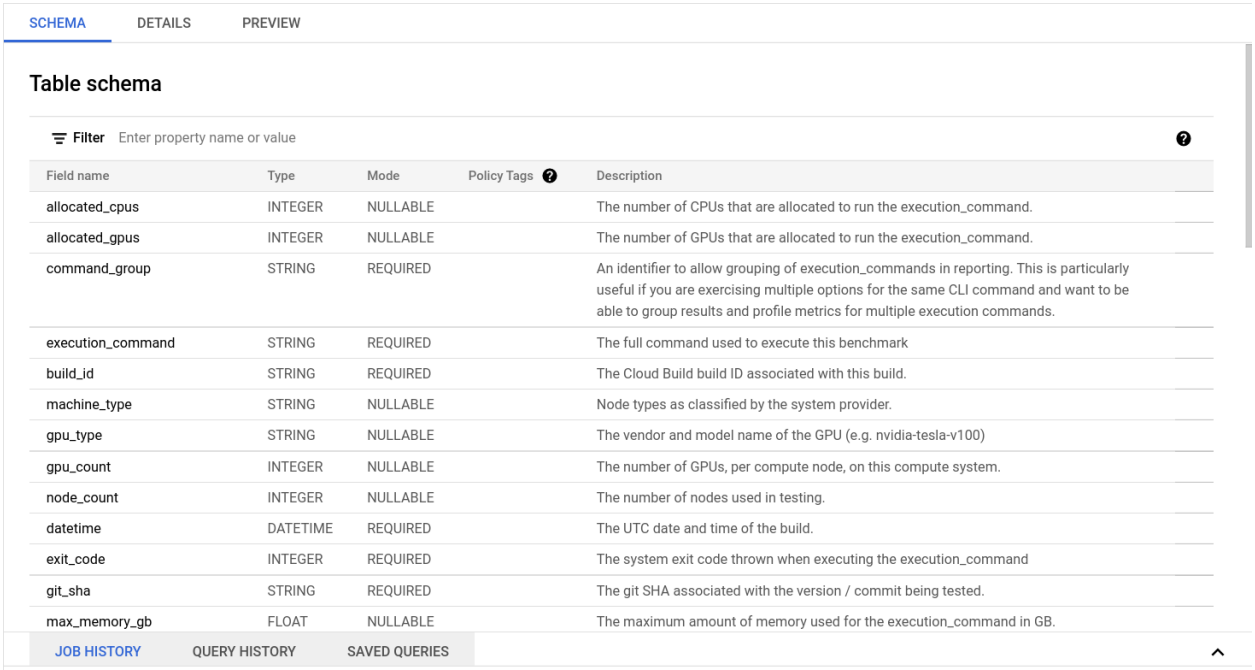
```
$ cd ~/fluid-run/examples/rcc-ephemeral/
$ gcloud builds submit . --config=ci/cloudbuild.yaml
```

Note that the cloud build can be run asynchronously by passing the `--async` flag as well. If you run asynchronously, you can view the status of the build at the Cloud Build Console.

1.1.5 View Data in Big Query

Once the build is complete, the run-time and other data for each execution command is posted to the fluid_cicb dataset in Big Query. In your browser, navigate to Big Query.

In the data explorer panel on the left-hand side, find your Google Cloud project and expand the dropdown menu.



SCHEMA DETAILS PREVIEW

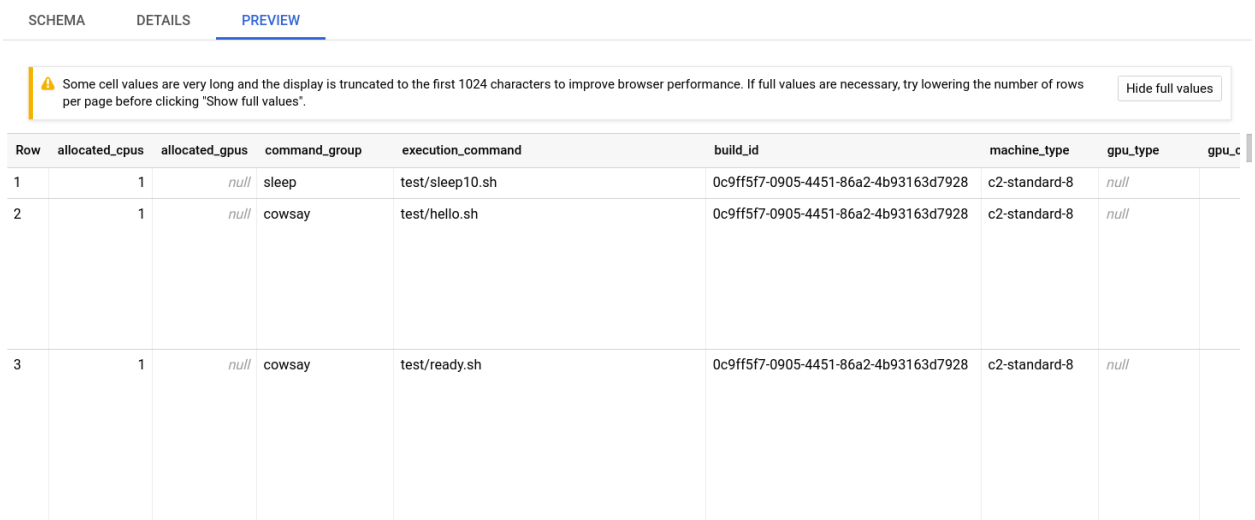
Table schema

Filter Enter property name or value

Field name	Type	Mode	Policy Tags	Description
allocated_cpus	INTEGER	NULLABLE		The number of CPUs that are allocated to run the execution_command.
allocated_gpus	INTEGER	NULLABLE		The number of GPUs that are allocated to run the execution_command.
command_group	STRING	REQUIRED		An identifier to allow grouping of execution_commands in reporting. This is particularly useful if you are exercising multiple options for the same CLI command and want to be able to group results and profile metrics for multiple execution commands.
execution_command	STRING	REQUIRED		The full command used to execute this benchmark
build_id	STRING	REQUIRED		The Cloud Build build ID associated with this build.
machine_type	STRING	NULLABLE		Node types as classified by the system provider.
gpu_type	STRING	NULLABLE		The vendor and model name of the GPU (e.g. nvidia-tesla-v100)
gpu_count	INTEGER	NULLABLE		The number of GPUs, per compute node, on this compute system.
node_count	INTEGER	NULLABLE		The number of nodes used in testing.
datetime	DATETIME	REQUIRED		The UTC date and time of the build.
exit_code	INTEGER	REQUIRED		The system exit code thrown when executing the execution_command
git_sha	STRING	REQUIRED		The git SHA associated with the version / commit being tested.
max_memory_gb	FLOAT	NULLABLE		The maximum amount of memory used for the execution_command in GB.

JOB HISTORY QUERY HISTORY SAVED QUERIES

Find the fluid-cicb dataset and the app_runs table. Once you've selected the app_runs table, select preview.



SCHEMA DETAILS PREVIEW

Some cell values are very long and the display is truncated to the first 1024 characters to improve browser performance. If full values are necessary, try lowering the number of rows per page before clicking "Show full values". Hide full values

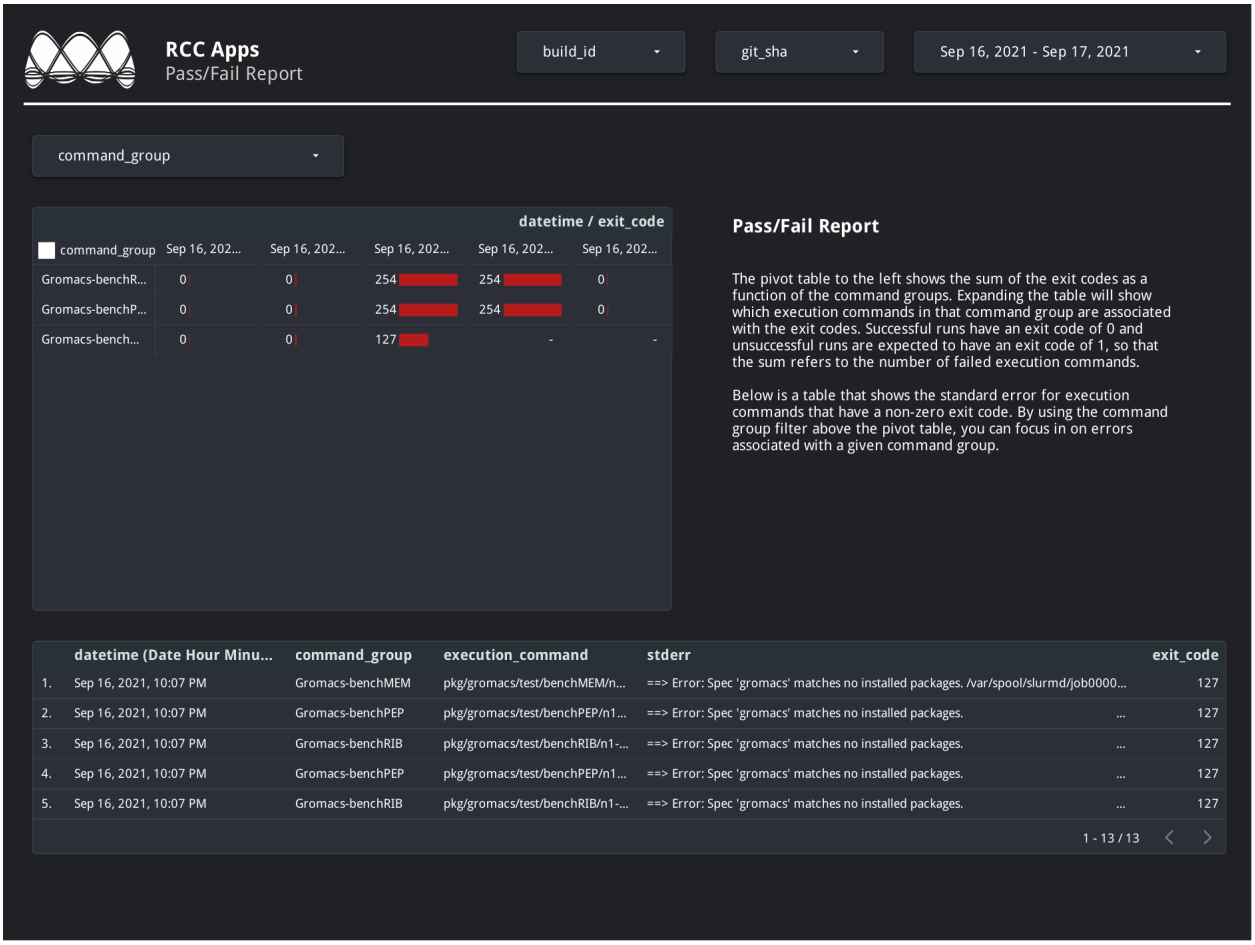
Row	allocated_cpus	allocated_gpus	command_group	execution_command	build_id	machine_type	gpu_type	gpu_c
1	1	null	sleep	test/sleep10.sh	0c9ff5f7-0905-4451-86a2-4b93163d7928	c2-standard-8	null	
2	1	null	cowsay	test/hello.sh	0c9ff5f7-0905-4451-86a2-4b93163d7928	c2-standard-8	null	
3	1	null	cowsay	test/ready.sh	0c9ff5f7-0905-4451-86a2-4b93163d7928	c2-standard-8	null	

At this point, you now have a dataset hosted in Google Cloud. The fluid-run build step with Google Cloud Build will allow you to automate testing and benchmarking of your application and will post results to this dataset.

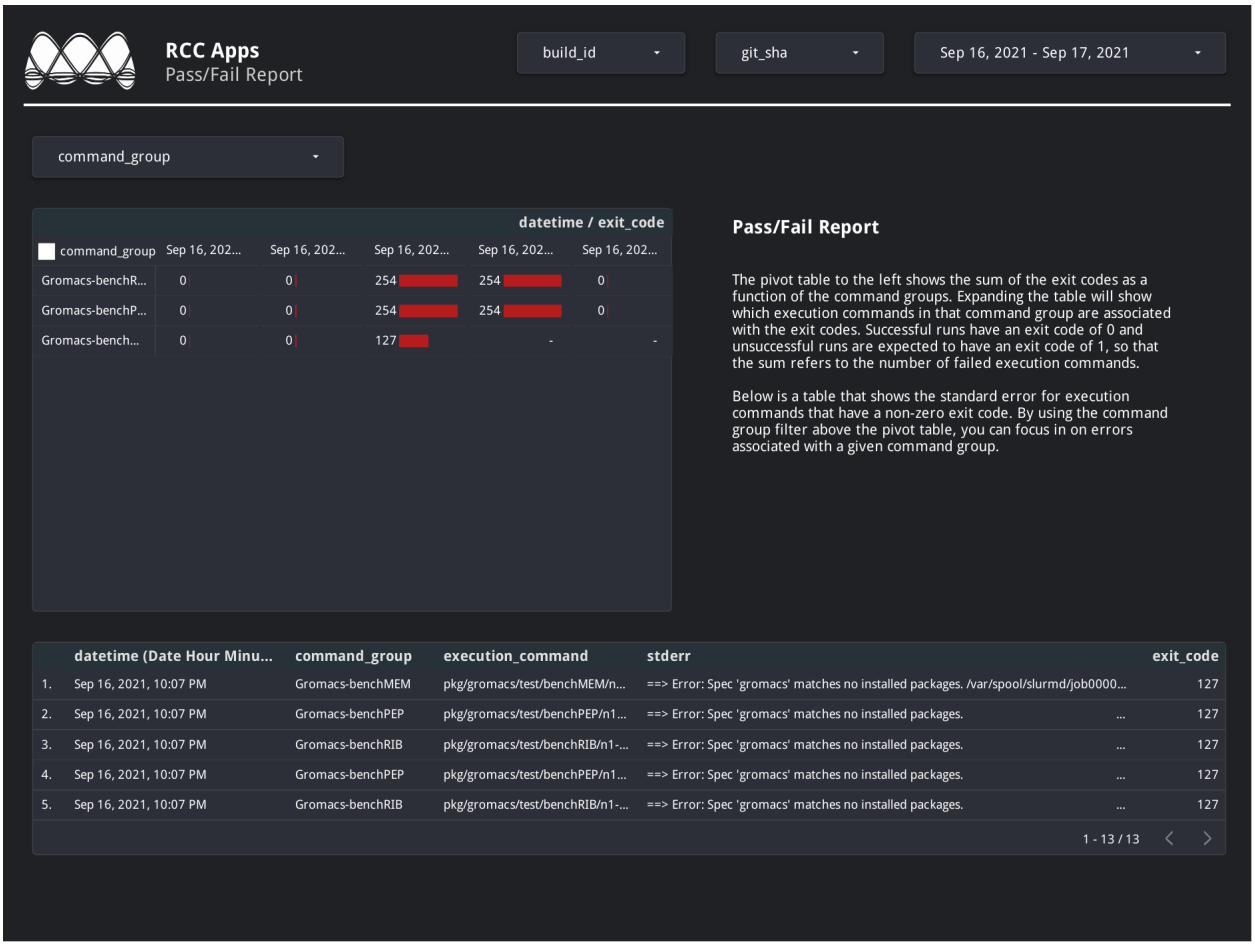
1.1.6 Dashboarding and other post-processing

From here, it is helpful to visualize results. There are a number of solutions available for visualizing data stored in Big Query. Below are a couple dashboard examples using Data Studio with the fluid_cic data set, to give you an idea of where you can take this.

Example Pass-Fail Report



Example Runtime Report



In addition to dashboarding, having a dataset that tracks the performance of your application over time and on a variety of hardware can enable you to automatically check for performance regressions or uncovers performance portability issues. You can write application in C#, Go, Java, Node.js, PHP, Python, and Ruby using the Big Query API to interact with the dataset to add further post-processing and verification to your builds.

1.1.7 Delete Resources

If you've worked through this tutorial on a Google Cloud project where you will continue setting up a CI/CB workflow for your application, you can keep using the resources you've created. However, if you need to tear down the resources created during this tutorial, you can use the commands below

```
$ cd ~/fluid-run/examples/rcc-ephemeral/ci/build_iac
$ terraform destroy --auto-approve
```

1.2 Next Steps

- *Set up your Git Repository*
- *Customize the Benchmarking Cluster*

CHAPTER

TWO

SET UP YOUR REPOSITORY

To use fluid-run, you need to have (at a minimum) a Google Cloud Build configuration file and a fluid-run pipeline file. The simplest location for these files is in the root directory of your repository.

```
Repository Root
o
|
|
o ./cloudbuild.yaml
|
|
o ./fluid-run.yaml
```

The `cloudbuild.yaml` configuration file specifies the steps necessary to build your application and includes a call to fluid-run to test and benchmark your application. Currently, fluid-run is able to test Singularity and Google Compute Engine (GCE) VM Images. If you're able to create a Docker image for your application, you can easily convert to a Singularity image within your build process before calling fluid-run.

The `fluid-run.yaml` pipeline file specifies a set of commands or scripts to execute, where to direct output, and the compute partitions to run on.

2.1 Containerize your application with Singularity

Containers are a lightweight virtual environment where you install your application and all of it's depedencies. They are ideal for improving portability of your application and can help developers reproduce issues reported by their end users. Singularity is a container format made specifically for high performance computing and research computing environments, where users often share common compute resources. Singularity has some advantages over other container options, such as Docker, including built in support for exposing AMD and Nvidia GPUs and running on multi-VM / cluster environments.

A Singularity image can be created by writing a Singularity definition file. The definition file is essentially a set of instructions that dictate the container image to start from and the commands to run to install your application. We recommend that you review the Singularity documentation to learn more about writing a Singularity definition file. If you have not containerized your application yet, this is a good place to start.

Some users have already containerized their application with Docker. If you fall into this category, but would still like to use fluid-run to test and benchmark your application, you can easily convert your Docker image to a Singularity image. In your `cloudbuild.yaml`, you will simply add a step to call `singularity build` using the local docker-daemon as a source. The example below shows a two stage process that creates a Docker image and a Singularity image. After the build completes, the Docker image is posted to Google Container Registry and the Singularity image is posted to Google Cloud Storage.

```
steps:

- id: Build Docker Image
  name: 'gcr.io/cloud-builders/docker'
  args: ['build',
         '.',
         '-t',
         'gcr.io/${PROJECT_ID}/cowsay:latest'
  ]

- id: Build Singularity Image
  name: 'quay.io/singularity/singularity:v3.7.1'
  args: ['build',
         'cowsay.sif',
         'docker-daemon://gcr.io/${PROJECT_ID}/cowsay:latest']

images: ['gcr.io/${PROJECT_ID}/cowsay:latest']

artifacts:
  objects:
    location: 'gs://my-singularity-bucket/latest'
    paths: ['cowsay.sif']
```

2.2 Define Tests

Tests for your application are specified in the `execution_command` field of the `fluid-run.yaml` pipeline file. The fluid-run build step is able to determine if the provided execution command is a script or a single command. This allows you to either specify all of your tests in a set of scripts in your repository or set individual commands in the `fluid-run.yaml` file. Currently, when using the `rcc-ephemeral` or `rcc-static` cluster types, you must specify a script to run; when using the `gce` cluster type, you must specify individual commands.

As an example, the `fluid-run.yaml` file below runs an inline command on the `c2-standard-8` partition (a partition provided in the default RCC cluster).

```
tests:
- command_group: "hello"
  execution_command: "singularity exec ${SINGULARITY_IMAGE} /usr/games/cowsay \"Hello␣
↪World\""
  output_directory: "hello/test"
  partition: "c2-standard-8"
  batch_options: "--ntasks=1 --cpus-per-task=1"
```

Alternatively, you could create a script in your repository (e.g. `test/hello_world.sh`) and reference the path to this script in your `fluid-run.yaml` file. In this case, the contents of the script would have the command(s) you want to run.

```
#!/bin/bash
singularity exec ${SINGULARITY_IMAGE} /usr/games/cowsay "Hello World"
```

The `fluid-run.yaml` then references this file in the `execution_command` field.

```
tests:
- command_group: "hello"
  execution_command: "test/hello_world.sh"
  output_directory: "hello/test"
  partition: "c2-standard-8"
  batch_options: "--ntasks=1 --cpus-per-task=1"
```

When writing your tests, keep in mind that you can use *environment variables* provided by fluid-run. If you are using the `rcc-ephemeral` or `rcc-static` clusters, you can also use Slurm environment variables.

2.3 Add a Cloud Build Configuration file

CHAPTER

THREE

CUSTOMIZE THE TEST CLUSTER

When using the `rcc-ephemeral` cluster type for testing, there are only a limited set of compute instances made avaiable to you by default. Behind the scenes, fluid-run uses Terraform to create ephemeral clusters for application testing and the `rcc-ephemeral` cluster is defined by the rcc-tf module from Fluid Numerics. The default variable concretizations are provided in fluid-run/etc/rcc-ephemeral/default.auto.tfvars. This default configuration provides you with a n1-standard-16 controller with a 1TB pd-ssd disk and a single compute partition, consisting of 5x c2-standard-8 instances.

The fluid-run build step provides a mechanism to customize the cluster so that you can define compute partitions that meet your testing needs. You are able to add instances with GPUs, specify partitions for a heterogeneous cluster (see machine types available on Google Cloud), specify the zone to deploy to, change the controller size, shape, and disk properties, and even add a Lustre file system.

3.1 Getting Started

To customize the cluster, you can add a tfvars definition file that is similar to the fluid-run/etc/rcc-ephemeral/default.auto.tfvars. For reference, the fluid-run/etc/rcc-ephemeral/io.tf file defines all of the variables available for concretizing a `rcc-ephemeral` cluster.

It is recommended that you start by creating a file in your repository called `rcc.auto.tfvars` with the following contents

```
cluster_name = "<name>"
project = "<project>"
zone = "<zone>"

controller_image = "<image>"
disable_controller_public_ips = false
controller_machine_type = "n1-standard-16"
controller_disk_size_gb = 1024
controller_disk_type = "pd-ssd"

login_image = "<image>"
disable_login_public_ips = true
login_machine_type = "n1-standard-4"
login_node_count = 0

suspend_time = 2


compute_node_scopes           = [
  "https://www.googleapis.com/auth/cloud-platform"
```

(continues on next page)

(continued from previous page)

```
]
partitions = [
  { name                 = "c2-standard-8"
    machine_type         = "c2-standard-8"
    image                = "<image>"
    image_hyperthreads   = true
    static_node_count    = 0
    max_node_count       = 5
    zone                 = "<zone>"
    compute_disk_type    = "pd-standard"
    compute_disk_size_gb = 50
    compute_labels       = {}
    cpu_platform         = null
    gpu_count            = 0
    gpu_type             = null
    gvnic                = false
    network_storage      = []
    preemptible_bursting = false
    vpc_subnet           = null
    exclusive            = false
    enable_placement     = false
    regional_capacity    = false
    regional_policy      = null
    instance_template    = null
  },
]

create_filestore = false
create_lustre = false
```

You'll notice that their are a few template variables in this example that are demarked by <>. The fluid-run build step is able to substitute values for these variables at build-time based on options provided to the command lined interface. The example above provides a good starting point with some of the necessary template variables in place. It is not recommended to remove the template variables for <name>, <project>, <zone>, or <image>.

For your reference, template variables for `rcc-ephemeral` clusters that are substituted at run-time are given in the table below.

Template Variable	Value/CLI Option	Description
<name>	frun-{build-id}[0:7]	Name of the ephemeral cluster
<project>	–project	Google Cloud Project ID
<zone>	–zone	Google Cloud zone
<image>	–gce-image	Google Compute Engine VM Image self-link
<build_id>	–build-id	Google Cloud Build build ID
<vpc_subnet>	–vpc-subnet	Google Cloud VPC Subnetwork
<service_account>	–service-account	Google Cloud Service Account email address

3.2 Customize Partitions

Partitions are used to define the type of compute nodes available to you for testing. Each partition consists of a homogeneous pool of machines. While each partition has 22 variables to concretely define it, we'll cover a few of the options here to help you make informed decisions when defining partitions for testing.

3.2.1 name

The partition name is used to identify a homogeneous group of compute nodes. When writing your Fluid Run CI File, you will set the `partition` field to one of the partition names set in your `tfvars` file.

3.2.2 machine_type

The machine type refers to a Google Compute Engine machine type. If you define multiple partitions with differing machine types, this gives you the ability to see how your code's performance varies across different hardware

3.2.3 max_node_count

This is the maximum number of nodes that can be created in this partition. When tests are run, the cluster will automatically manage provisioning compute nodes to run benchmarks and tear them down upon completion. Keep in mind that you need to ensure that you have sufficient Quota for the machine type, gpus, and disks in the region that your cluster is deployed to.

3.2.4 image

The `image` expects a self-link to a VM image for the cluster. It is recommended that you leave this field set to the template variable `"<image>"` so that fluid-run can set this field for you.

3.2.5 gpu_type / gpu_count

The `gpu_type` field is used to set the type of GPU to attach to each compute node in the partition. Possible values are

- nvidia-tesla-k80
- nvidia-tesla-p100
- nvidia-tesla-v100
- nvidia-tesla-p4
- nvidia-tesla-t4
- nvidia-tesla-a100 (A2 instances only)

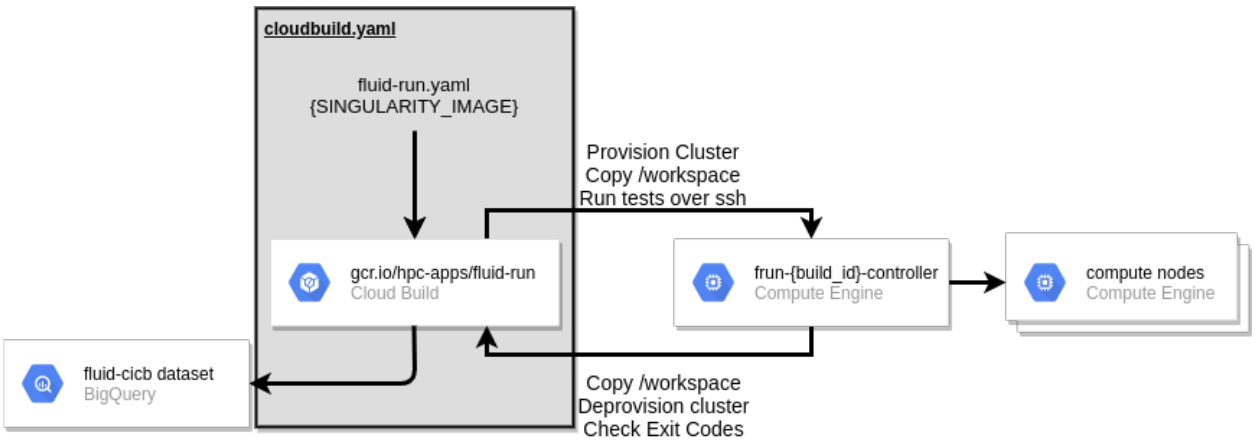
The `gpu_count` field is used to set the number of GPUs per machine in the partition. For most GPUs, you can set this to 0, 1, 2, 4, or 8. Currently, GPUs must be used with an *n1* machine type on Google Cloud (except for the A100 GPUs). Keep in mind that each GPU type is available in certain zones and that there are restrictions on the ratio of vCPU to GPU.

CHAPTER

FOUR

ARCHITECTURE REFERENCE

4.1 Overview



Fluid Run is a docker image that is meant to be used as a build step in your repository's Cloud Build pipeline. Currently, Fluid Run accepts only Google Compute Engine VM Images or Singularity Images as build artifacts that can be tested. However, if you currently build Docker images, you can easily create a Singularity image from your Docker image.

4.2 Workflow

When fluid-run is called, it will provision an ephemeral Slurm controller for processing tests specified in your *Fluid Run CI YAML*. Once the controller is provisioned, the local workspace from Cloud Build is copied to the controller and jobs are submitted and tracked by the cluster-workflow script.

When each job finishes, this script will align run-time, max memory used, exit code, stdout, stderr, and information about the systems with each execution command is run on. When all jobs are finished, the workspace on the controller is copied back to the Cloud Build workspace. In the last step, the recorded details about your tests are loaded up to Big Query. By default, fluid-run will throw a non-zero exit code if any of your tests show a non-zero exit code; this behavior can be overridden with the *–ignore-exit-code* flag.

CHAPTER

FIVE

THE FLUID RUN CI FILE

5.1 Overview

The Fluid Run CI File is a YAML or json file in your repository that specifies the tests/benchmarks you want to run after building your code. Currently, this file consists of the a single list object `tests` that has the following attributes :

- `execution_command` This is the path to a script in your repository to run a specific test
- `command_group` The command group is used to group execution commands that are dependent. Execution commands in the same command group are run sequentially in the order they are placed in the `tests` block, unless the *–ignore-job-dependencies* flag is sent to fluid-run
- `output_directory` The directory on the cluster, relative a unique workspace, where the execution command should be run.
- `partition` The compute partition to run the execution command under. *See How to Customize the Cluster*
- `batch_options` Options to send to Slurm sbatch to submit the job (excluding the `--partition` option)

5.2 Example

```yaml
tests:
  - command_group: "sleep"
    execution_command: "test/sleep10.sh"
    output_directory: "sleep"
    partition: "c2-standard-8"
    batch_options: "--ntasks=1 --cpus-per-task=1 --time=05:00"

  - command_group: "cowsay"
    execution_command: "test/hello.sh"
    output_directory: "cowsay-hello"
    partition: "c2-standard-8"
    batch_options: "--ntasks=1 --cpus-per-task=1 --time=05:00"

  - command_group: "cowsay"
    execution_command: "test/ready.sh"
    output_directory: "cowsay-ready"
    partition: "c2-standard-8"
    batch_options: "--ntasks=1 --cpus-per-task=1 --time=05:00"
```

CHAPTER

SIX

COMMAND LINE INTERFACE

6.1 Usage

The fluid-run container is intended to be used as a build step in Google Cloud Build. Once you create the fluid-run container image, you can call it in your cloud build configuration file using something like the following:

```
- id: CI/CB
  name: 'gcr.io/${PROJECT_ID}/fluid-run'
  args:
  - '--build-id=${BUILD_ID}'
  - '--git-sha=${COMMIT_SHA}'
  - '--singularity-image=cowsay.sif'
  - '--project=${PROJECT_ID}'
```

In this example, a minimal set of arguments are provided to fluid-run to run tests on a singularity image called *cowsay.sif*. By default, fluid-run would look for a CI file at *./fluid-run.yaml* in your repository. Additionally, it would use a *rcc-ephemeral* cluster type for testing and only expose a limited set of machine types for running tests and benchmarks.

6.2 CLI Arguments

There are a number of options to customize the behavior of fluid-run. The table below provides a complete summary of the arguments along with their default values.

Argument	Required	Cluster Type	Artifact Type	Default Value
–build-id	Yes	All	All	“”
–cluster-type	No	All	All	rcc-ephemeral
–git-sha	Yes	All	All	“”
–ci-file	No	All	All	./fluid-run.yaml
–node-count	No	GCE	All	1
–machine-type	No	GCE	All	c2-standard-8
–compiler	No	All	All	“”
–target-arch	No	All	All	“”
–gpu-count	Yes	GCE	All	0
–gpu-type	Yes	GCE	All	“”
–nproc	No	GCE	All	1
–task-affinity	Yes	GCE	All	“”
–mpi	Yes	GCE	All	false
–vpc-subnet	No	All	All	“”
–service-account	No	GCE	All	fluid-run@${PROJECT}.iam.gserviceaccount.com
–artifact-type	Yes	All	All	singularity
–singularity-image	Yes	All	singularity	“”
–gce-image	No	All	All	project/research-computing-cloud/global/images/rcc-foss
–project	Yes	All	All	“”
–zone	No	All	All	us-west1-b
–rcc-controller	Yes	RCC-Static	All	“”
–rcc-tfvars	No	RCC-Ephemeral	All	./fluid.auto.tfvars
–save-results	No	All	All	false
–ignore-exit-code	No	All	All	false
–ignore-job-dependencies	No	All	All	false

This next table gives a description for each of the command line arguments.

CHAPTER

SEVEN

BENCHMARK DATASET SCHEMA

With each *execution_command* in your CI file, fluid-run will align variables about your build and testing environment along with runtimes to create a fully auditable record of the execution. This allows you to naturally generate a database over time that can track how your application performs over time and on all available hardware on Google Cloud. Knowing this information is critical for optimizing costs for your applications on public cloud systems. The table below provides an overview of the current schema.

Field name	Type	Mode	Cluster Type(s)	Description
allocated_cpus	INTEGER	NULLABLE	RCC	The number of CPUs that are allocated to run th
allocated_gpus	INTEGER	NULLABLE	RCC	The number of GPUs that are allocated to run th
batch_options	STRING	NULLABLE	RCC	Additional options sent to the batch scheduler
command_group	STRING	REQUIRED	RCC, GCE	An identifier to allow grouping of execution_cor
execution_command	STRING	REQUIRED	RCC, GCE	The full command used to execute this benchma
build_id	STRING	REQUIRED	RCC, GCE	The Cloud Build build ID associated with this b
machine_type	STRING	NULLABLE	RCC, GCE	Node types as classified by the system provider.
gpu_type	STRING	NULLABLE	RCC, GCE	The vendor and model name of the GPU (e.g. nv
gpu_count	INTEGER	NULLABLE	RCC, GCE	The number of GPUs, per compute node, on this
node_count	INTEGER	NULLABLE	RCC, GCE	The number of nodes used in testing.
datetime	DATETIME	REQUIRED	RCC, GCE	The UTC date and time of the build.
exit_code	INTEGER	REQUIRED	RCC, GCE	The system exit code thrown when executing the
git_sha	STRING	REQUIRED	RCC, GCE	The git SHA associated with the version / comm
max_memory_gb	FLOAT	NULLABLE	RCC	The maximum amount of memory used for the e
stderr	STRING	NULLABLE	RCC, GCE	Standard error produced from running execution
stdout	STRING	NULLABLE	RCC, GCE	Standard output produced from running executio
partition	STRING	NULLABLE	RCC	The name of the scheduler partition to run the jo
runtime	FLOAT	NULLABLE	RCC, GCE	The runtime for the execution_command in seco
compiler	STRING	NULLABLE	RCC, GCE	Compiler name and version, e.g. *gcc@10.2.0*, us
target_arch	STRING	NULLABLE	RCC, GCE	Architecture targeted by compiler during applica
controller_machine_type	STRING	NULLABLE	RCC	Machine type used for the controller, for Slurm b
controller_disk_size_gb	INTEGER	NULLABLE	RCC	The size of the controller disk in GB.
controller_disk_type	STRING	NULLABLE	RCC	The type of disk used for the controller.
filestore	BOOLEAN	NULLABLE	RCC	A flag to indicated if filestore is used for worksp
filestore_tier	STRING	NULLABLE	RCC	The filestore tier used for file IO.
filestore_capacity_gb	INTEGER	NULLABLE	RCC	The size of the filestore disk capacity in GB.
lustre	BOOLEAN	NULLABLE	RCC	A flag to indicated if lustre is used for workspac
lustre_mds_node_count	INTEGER	NULLABLE	RCC	Number of Lustre metadata servers
lustre_mds_machine_type	STRING	NULLABLE	RCC	The machine type for the Lustre MDS servers.
lustre_mds_boot_disk_type	STRING	NULLABLE	RCC	The boot disk type for the Lustre MDS servers.
lustre_mds_boot_disk_size_gb	INTEGER	NULLABLE	RCC	The size of the Lustre boot disk in GB.
lustre_mdt_disk_type	STRING	NULLABLE	RCC	The mdt disk type for the Lustre MDS servers.

Table 1 – continued from previous page

Field name	Type	Mode	Cluster Type(s)	Description
lustre_mdt_disk_size_gb	INTEGER	NULLABLE	RCC	The size of the Lustre boot disk in GB.
lustre_mdt_per_mds	INTEGER	NULLABLE	RCC	The number of metadata targets per MDS.
lustre_oss_node_count	INTEGER	NULLABLE	RCC	Number of Lustre metadata servers
lustre_oss_machine_type	STRING	NULLABLE	RCC	The machine type for the Lustre OSS servers.
lustre_oss_boot_disk_type	STRING	NULLABLE	RCC	The boot disk type for the Lustre OSS servers.
lustre_oss_boot_disk_size_gb	INTEGER	NULLABLE	RCC	The size of the Lustre boot disk in GB.
lustre_ost_disk_type	STRING	NULLABLE	RCC	The ost disk type for the Lustre OSS servers.
lustre_ost_disk_size_gb	INTEGER	NULLABLE	RCC	The size of the Lustre boot disk in GB.
lustre_ost_per_oss	INTEGER	NULLABLE	RCC	The number of object storage targets per OSS.
compact_placement	BOOLEAN	NULLABLE	RCC	A flag to indicate if compact placement is used.
gvnic	BOOLEAN	NULLABLE	RCC	A flag to indicate if Google Virtual NIC is used.
lustre_image	STRING	NULLABLE	RCC	The VM image used for the Lustre deployment.
vm_image	STRING	NULLABLE	RCC	VM image used for the GCE instance running th

CHAPTER

EIGHT

ENVIRONMENT VARIABLES

When running batch scripts on RCC style platforms and when running in-line commands on GCE clusters, some environment variables are provided for you to use during runtime.

Since RCC clusters use a Slurm job scheduler, you also have access to common Slurm environment variables when *–cluster-type=rcc-static* or *–cluster-type=rcc-ephemeral*.

Variable	Description
WORKSPACE	The path to the working directory where your job is executed.
PROJECT	The Google Cloud project hosting your test cluster.
GIT_SHA	The git sha associated with the run test.
SINGULARITY_IMAGE	The full path to the Singularity image on the test cluster.

8.1 Example Job Script (Singularity)

When writing a job script to test your application, you can use the provided environment variables to reference the working directory and the full path to the Singularity image produced during the build phase in Cloud Build. The example below provides a basic demonstration for using environment variables in your test scripts.

```
#!/bin/bash

cd ${WORKSPACE}
spack load singularity
singularity exec ${SINGULARITY_IMAGE} /usr/games/cowsay "Great.. I'm self aware."
```

CHAPTER

NINE

SUPPORT

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search