Amazon Genomics CLI Preview

Quickstart Guide

Preview Considerations

The following instructions apply to a preview version of Amazon Genomics CLI (AGC). Some capabilities are still under development. We do not recommend using AGC for production workloads at this time. This version requires admin access to an AWS account. Access permissions will be scoped down in future releases. Use a test account that DOES NOT contain sensitive data for this preview.

Known Limitations

All of the following are known limitations of the current release that will be addressed in upcoming releases.

- Project infrastructure is created using the "default" profile created by the AWS CLI.
- Only one project per AWS account per region is supported at this time. Resources created will use a project named "myproj" irregardless of what project name is set in project config files.
- Amazon Genomics CLI has not been tested natively on Windows

Prerequisites

- Environment
  - Operating System:
    - macOS 10.14+
    - Amazon Linux 2
    - Ubuntu 20.04
  - AWS CLI 2.x, configured with a default user
  - internet access
  - AWS Account with admin access
- Permissions
  - AWS AccountId granted access to preview assets and artifacts

Prerequisite installation

Ubuntu 20.04

- Install node.js

```
curl -fsSL https://deb.nodesource.com/setup_15.x | sudo -E bash -
sudo apt-get install -y nodejs
```

- Install and configure AWS CLI

```
sudo apt install awscli
aws configure
# ... set access key ID, secret access key, and region
```

Amazon Linux 2 (on an EC2 instance)

- Install node

```
curl -sL https://rpm.nodesource.com/setup_16.x | sudo -E bash -
sudo yum install -y nodejs
```

- configure a default region

```
aws configure
```

MacOS

- Install Homebrew

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

- Install node

```
brew install node
```

- Install and configure AWS CLI

```
brew install awscli
aws configure
# ... set access key ID, secret access key, and region
```

Download and install Amazon Genomics CLI

Download the Amazon Genomics CLI zip, unzip its contents, and run the `install.sh` script:

```
aws s3 cp s3://healthai-public-assets-us-east-1/amazon-genomics-cli/0.8.1/amazon-
genomics-cli.zip .
unzip amazon-genomics-cli.zip -d agc
./agc/install.sh
```

This will place the `agc` command in `~/bin` and add the command to your `$PATH`

The Amazon Genomics CLI is a statically compiled Go binary. It should run in your environment natively without any additional setup. Test the CLI with:

```
$ agc --help
👩🧬 Launch and manage genomics workloads on AWS.

Commands
```

```
  Getting Started 🌱
    account     Commands for AWS account setup.
                Install or remove AGC from your account.

  Contexts
    context     Commands for contexts.
                Contexts specify workflow engines and computational fleets to use
when running a workflow.

  Logs
    logs        Commands for various logs (currently only CloudWatch).

  Projects
    project     Commands to interact with projects.

  Workflows
    workflow    Commands for workflows.
                Workflows are potentially-dynamic graphs of computational tasks to
execute.

  Settings ⚙️
    version     Print the version number.

Flags
  -h, --help      help for agc
  -v, --verbose   display verbose diagnostic information
      --version   version for agc
Examples
  Displays the help menu for the specified sub-command.
  `$ agc account --help`
```

If this doesn't work immediately, either:

- start a new terminal shell
- source your `~/.bashrc` file

Also verify that you have the latest version of Amazon Genomics CLI with:

```
agc --version
```

You should get the following:

```
agc version: v0.8.0-1-ge65af34
```

If you do not, you may need to uninstall any previous versions of Amazon Genomics CLI and reinstall the latest.

The install steps above will also create a `agc` folder. Inside there is an `examples` folder where you will find example specifications and workflows. The rest of this guide will refer to those examples, and for simplicity, assumes these examples are accessible from `~/agc/examples`.

Account activation

To start using Amazon Genomics CLI with your AWS account, you need to activate it.

```
agc account activate
```

This will create the core infrastructure that Amazon Genomics CLI needs to operate, which includes an S3 bucket, a VPC, and an ECS microservice. This will take 5-10min to complete. You only need to do this once per account per region.

The ECS microservice is Amazon Genomics CLI's control plane. The S3 bucket is used for durable metadata and the VPC is used to isolate compute resources. You can specify your own preexisting S3 Bucket or VPC if needed using `--bucket` and `--vpc` options.

Initialize a project

Amazon Genomics CLI uses local folders and config files to define projects. Projects contain configuration settings for contexts and workflows (more on these below). To create a new project do the following:

```
mkdir myproject
cd myproject
agc project init myproject
```

This will create a `project.yaml` config file with the following contents

```
name: myproject
```

This config file will be used to define aspects of the project - e.g. contexts and named workflows the project uses. Command line activities for the remainder of this document should be run from within the project folder.

Contexts

Amazon Genomics CLI uses a concept called "contexts" to run workflows. Contexts encapsulate and automate time consuming tasks like configuring and deploying workflow engines, creating data access policies, and tuning compute clusters for operation at scale. All projects will have a "default" context. You can verify this with:

```
agc context list
```

You should see something like:

```
2021-07-31T03:44:51Z 𝒊  Listing contexts.
CONTEXTNAME    default
```

You need to have a context running to be able to run workflows. To start the default context, run:

```
agc context start default
```

This will take 5-10min to complete.

By default, contexts will only have access to data at a context specific prefix in the bucket Amazon Genomics CLI creates during account activation. You can check this for the `default` context with:

```
agc context describe default
```

You should see something like:

```
2021-07-31T03:51:57Z 𝒊  Describing context definition. Context name: default
CONTEXT    default    s3://<bucket-name>/projects/<project-name>/contexts/default
```

The S3 location shown is where the context can read and write data by default.

You can modify the context - e.g. by adding additional data locations - via the `project.yaml` file. The following config adds three public buckets from the Registry of Open Data on AWS as a set named “public”. It then adds this to the “default” context:

```
name: myproject
data:
  public:
    - location: s3://broad-references
      readOnly: true
    - location: s3://gatk-test-data
      readOnly: true
    - location: s3://1000genomes
      readOnly: true
contexts:
  default:
    dataRefs:
      - public
```

Now when you describe the default context, you should see something like:

```
2021-07-31T04:41:00Z 𝒊  Describing context definition. Context name: default
CONTEXT    default    s3://agc-<account-id>-
<region>/projects/testproject/contexts/default
DATAREF    public
DATA    s3://broad-references    true
DATA    s3://gatk-test-data    true
DATA    s3://1000genomes    true
```

NOTE: For these buckets to be accessible by resources in the context, you need to restart the context if it is already running. Do this by simply running.

```
agc context stop default
agc context start default
```

You can add your own custom contexts in a similar way as described above. For example, a `project.yaml` file that defines an additional context called "test" that also uses the "public" set of data would look like:

```
name: myproject
data:
  public:
    - location: s3://broad-references
      readOnly: true
    - location: s3://gatk-test-data
      readOnly: true
    - location: s3://1000genomes
      readOnly: true
contexts:
  default:
    dataRefs:
      - public
  test:
    dataRefs:
      - public
```

Workflows

Add a workflow

Bioinformatics workflows are written in languages like WDL in either single script files, or in packages of multiple files (e.g. when there are multiple related workflows that leverage reusable elements). For clarity, we'll refer to these workflow script files as "workflow definitions". A "workflow specification" for Amazon Genomics CLI references workflow definitions and combines it with additional metadata, like the workflow language the definition is written in, that Amazon Genomics CLI uses to execute it on appropriate compute resources.

There is a "hello" workflow in the `examples` folder that looks like:

```
version 1.0
workflow w {
    call hello {}
}
task hello {
    command { echo "Hello Amazon Genomics CLI!" }
    runtime {
        docker: "ubuntu:latest"
    }
    output { String out = read_string( stdout() ) }
}
```

To add this workflow specification, edit your `project.yaml` file, adding the following section:

```
workflows:
  hello:
    type: wdl
    sourceURL: /path/to/hello.wdl
```

NOTE: When referring to local workflow definitions, `sourceURL` must either be a full absolute path or a path relative to the `project.yaml` file. Path expansion is currently not supported.

You should now be able to see "hello" workflow as an available workflow to run with:

```
agc workflow list
```

This should return something like:

```
2021-08-04T22:48:59Z i  Listing workflows.
WORKFLOWNAME    hello
```

The workflow specification points to a single-file workflow. Workflows can also be directories. For example, if you have a workflow that looks like:

```
workflows/hello-dir
|-- inputs.json
`-- main.wdl
```

The workflow specification for the workflow above simply points to the parent directory:

```
workflows:
  hello-dir:
    type: wdl
    sourceURL: /path/to/hello-dir
```

In this case, your workflow must be named `main.<workflow-type>` - e.g. `main.wdl`

You can also provide a `MANIFEST.json` file that points to a specific workflow file to run. If you have a folder like:

```
workflows/hello-manifest/
|-- MANIFEST.json
|-- hello.wdl
|-- inputs.json
`-- options.json
```

The `MANIFEST.json` file would be:

```
{
    "mainWorkflowURL": "hello.wdl",
    "inputFileURLs": [
        "inputs.json"
    ]
}
```

At minimum, MANIFEST files must have a `mainWorkflowURL` property which is a relative path to the workflow file in its parent directory.

Workflows can also be from remote sources like GitHub:

```
workflows:
  remote:
    type: wdl
    sourceURL: https://raw.githubusercontent.com/openwdl/learn-
wdl/master/1_script_examples/1_hello_worlds/1_hello/hello.wdl
```

Running a workflow

To run a workflow you need a running context. See the section on contexts above if you need to start one. To run the “hello” workflow in the “default” context, run:

```
agc workflow run hello
```

If you have another context in your project, for example one named “test”, you can run the “hello” workflow there with:

```
agc workflow run hello --context test
```

If your workflow was successfully submitted you should get something like:

```
2021-08-04T23:01:37Z 𝒊  Running workflow. Workflow name: 'hello', Arguments: '',
Context: 'default'
"06604478-0897-462a-9ad1-47dd5c5717ca"
```

The last line is the workflow execution id. You use this id to reference a specific workflow execution.

Running workflows is an asynchronous process. After submitting a workflow from the CLI, it is handled entirely in the cloud. You can now close your terminal session if needed. The workflow will still continue to run. You can also run multiple workflows at a time. The underlying compute resources will automatically scale. Try running multiple instances of the “hello” workflow at once.

You can check the status of all running workflows with:

```
agc workflow status
```

You should see something like this:

```
WORKFLOWINSTANCE            a920ffb6-6352-4bec-bffe-
9475ee4ee279        hello        terminated
WORKFLOWINSTANCE            cd7b3d2d-6e57-4957-897b-
4b092331da96        hello        running
WORKFLOWINSTANCE            f7ec14a3-ef3f-4ab8-8a94-
dd3a8f8c0ae5        hello        running
```

For more information, you can use:

```
agc workflow status -l
```

This will provide extra details like if a workflow completed with an error and workflow execution duration:

```
WORKFLOWINSTANCE    2m58.525571288s        a920ffb6-6352-4bec-bffe-
9475ee4ee279    COMPLETE    hello    Wed, 07 Jul 2021 04:47:11 +0000    terminated
WORKFLOWINSTANCE    23.025560738s        cd7b3d2d-6e57-4957-897b-
4b092331da96    COMPLETE    hello    Wed, 07 Jul 2021 04:55:22 +0000    terminated
WORKFLOWINSTANCE    16.778381004s        f7ec14a3-ef3f-4ab8-8a94-
dd3a8f8c0ae5    COMPLETE    hello    Wed, 07 Jul 2021 04:55:29 +0000    terminated
```

The columns are `duration, execution-id, info, workflow-name, start-time, status`

When a workflow finishes successfully, its “Status” will change to `terminated` and its “Info” will be `COMPLETE`. If there is an error with the workflow, this will be indicated in the “Info” column.

If you have multiple workflows running simultaneously the above command will show the state of all of them, as well as workflows that have previously completed.

If you want to check the status of a specific workflow you can do so by referencing the workflow execution by it’s unique id:

```
agc workflow status <uuid>
```

Using workflow inputs

You can provide runtime inputs to workflows at the command line. For example, say you have a workflow named “read” that requires reading a data file that looks like:

```
version 1.0
workflow ReadFile {
    input {
        File input_file
    }
    call read_file { input: input_file = input_file }
}

task read_file {
    input {
        File input_file
    }
    String content = read_string(input_file)

    command {
        echo '~{content}'
    }
    runtime {
        docker: "ubuntu:latest"
        memory: "4G"
    }

    output { String out = read_string( stdout() ) }
}
```

You would add this workflow to your `project.yaml` file with:

```
workflows:
  # .. other workflows ..
  read:
    type: wdl
    sourceURL: workflows/read.wdl
```

You can create an input file locally for this workflow:

```
mkdir inputs
echo "this is some data" > inputs/data.txt
cat << EOF > inputs/read.inputs.json
{"ReadFile.input_file": "inputs/data.txt"}
EOF
```

Finally, you would submit the workflow with it’s corresponding inputs file with:

```
agc workflow run read --args inputs/read.inputs.json
```

Amazon Genomics CLI will scan the file provided to `--args` for local paths, sync those files to S3, and rewrite the inputs file in transit to point to the appropriate S3 locations.

Accessing workflow results

Workflow results are written to an S3 bucket specified or created by Amazon Genomics CLI during account activation. See the section on account activation above for more details. You can list or retrieve the S3 URI for the bucket with:

```
AGC_BUCKET=$(aws ssm get-parameter \
    --name /agc/_common/bucket \
    --query 'Parameter.Value' \
    --output text)
```

and then use `aws s3` commands to explore and retrieve data from the bucket. For example, to list the bucket contents:

```
aws s3 ls $AGC_BUCKET
```

You should see something like:

```
PRE project/
PRE scripts/
```

Data for multiple projects are kept in `project/<project-name>` prefixes. Looking into one you should see:

```
PRE cromwell-execution/
PRE workflow/
```

The `cromwell-execution` prefix is specific to the engine Amazon Genomics CLI uses to run WDL workflows. Workflow results will be in `cromwell-execution` partitioned by workflow name, workflow execution id, and task name. The `workflow` prefix is where named workflows are cached when you run workflows definitions stored in your local environment.

Accessing workflow logs

You can get a summary of the log information available for a workflow as follows:

```
agc workflow logs <uuid>
```

You should see something like this:

```
LogStreamName 1e0fdb8f-2522-46a2-8064-6109a936eebd:
  TaskLogs:
    Name: w.hello
    LogStreamName: 82004336-17eb-4ee1-9d3d-ba4cf3b06b83
    StartTime: 2021-06-16T18:25:08.170Z,    EndTime: 2021-06-16T18:25:53.594Z
    Stdout: s3://agc-<account-id>-<region>/project/<project-name>/cromwell-
execution/w/1e0fdb8f-2522-46a2-8064-6109a936eebd/call-hello/hello-stdout.log
    Stderr: s3://agc-<account-id>-<region>/project/<project-name>/cromwell-
execution/w/1e0fdb8f-2522-46a2-8064-6109a936eebd/call-hello/hello-stderr.log
    CW log:
      log-group: /aws/batch/job
      log-stream:
cromwell_ubuntu_latest72d98123b11b4086634680ff136112993c23763c/default/ebf7361dbf3544
37bfb9b15053a145f6
    ExitCode: 0
```

For each task – which corresponds to an AWS Batch job that was run – you will be told what stdout/stderr files exist in the
S3 output bucket and what Cloudwatch log streams exist.

To retrieve the content of a specific Cloudwatch log stream, use the following command:

```
agc logs log [<log-stream-group>] <log-stream-name>
```

The values for `log-stream-group` and `log-stream-name` to use are in the `CW log` section of each task in the workflow log. In the example above, these are:

```
log-group: /aws/batch/job
log-stream:
cromwell_ubuntu_latest72d98123b11b4086634680ff136112993c23763c/default/ebf7361dbf3544
37bfb9b15053a145f6
```

Logs for individual tasks look like:

```
Event messages for Cloudwatch log stream {/aws/batch/job,
cromwell_ubuntu_latest72d98123b11b4086634680ff136112993c23763c/default/ebf7361dbf3544
37bfb9b15053a145f6}:
  *** LOCALIZING INPUTS ***
  download: s3://agc-<account-id>-<region>/project/<project-name>/cromwell-
execution/w/1e0fdb8f-2522-46a2-8064-6109a936eebd/call-hello/script to phosphate-
output-bucket-733263974272-us-west-2-demo/cromwell-execution/w/1e0fdb8f-2522-46a2-
8064-6109a936eebd/call-hello/script
  *** COMPLETED LOCALIZATION ***
  Hello Amazon Genomics CLI!
  *** DELOCALIZING OUTPUTS ***
  upload: ./hello-rc.txt to s3://agc-<account-id>-<region>/project/<project-
name>/cromwell-execution/w/1e0fdb8f-2522-46a2-8064-6109a936eebd/call-hello/hello-
rc.txt
  upload: ./hello-stderr.log to s3://agc-<account-id>-<region>/project/<project-
name>/cromwell-execution/w/1e0fdb8f-2522-46a2-8064-6109a936eebd/call-hello/hello-
stderr.log
  upload: ./hello-stdout.log to s3://agc-<account-id>-<region>/project/<project-
name>/cromwell-execution/w/1e0fdb8f-2522-46a2-8064-6109a936eebd/call-hello/hello-
```

```
stdout.log
  *** COMPLETED DELOCALIZATION ***
```

If you omit the log-stream-group then the AWS Batch log stream group (/aws/batch/job/) will be assumed.

If your workflow fails, useful debug information is typically reported by the workflow engine logs. To get those for the `cromwell` engine in the `prod` context, you run:

```
agc logs engine cromwell --project myproj --context default
```

You should get something like:

```
Event messages for Cloudwatch log stream {/agc/myproj/prod/cromwell---11e32d1d-6812-
4de0-9581-2c795a7f0831, cromwell/web/d8901182b3154adca986953da11feb52}:
  2021-07-07 04:39:44,705  INFO  - Running with database db.url =
jdbc:hsqldb:mem:a8c7c9a2-df8e-442f-ab62-db3ad3267dd5;shutdown=false;hsqldb.tx=mvcc
  2021-07-07 04:39:53,424  INFO  - Running migration RenameWorkflowOptionsInMetadata
with a read batch size of 100000 and a write batch size of 100000
  2021-07-07 04:39:53,435  INFO  - [RenameWorkflowOptionsInMetadata] 100%
  2021-07-07 04:39:53,588  INFO  - Running with database db.url =
jdbc:hsqldb:mem:3e482536-11e0-439d-8b34-cfc98d8f7810;shutdown=false;hsqldb.tx=mvcc
  2021-07-07 04:39:54,050  WARN  - Unrecognized configuration key(s) for AwsBatch:
auth, numCreateDefinitionAttempts, filesystems.s3.duplication-strategy,
numSubmitAttempts, default-runtime-attributes.scriptBucketName
  2021-07-07 04:39:54,297  INFO  - Slf4jLogger started
  2021-07-07 04:39:54,528 cromwell-system-akka.dispatchers.engine-dispatcher-4
INFO  - Workflow heartbeat configuration:
  {
    "cromwellId" : "cromid-f52e2fc",
    "heartbeatInterval" : "2 minutes",
    "ttl" : "10 minutes",
    "failureShutdownDuration" : "5 minutes",
    "writeBatchSize" : 10000,
    "writeThreshold" : 10000
  }
  2021-07-07 04:39:54,727 cromwell-system-akka.dispatchers.service-dispatcher-8
INFO  - Metadata summary refreshing every 1 second.

[... truncated ...]

  2021-07-07 04:55:41,717 cromwell-system-akka.dispatchers.engine-dispatcher-20
INFO  - WorkflowExecutionActor-f7ec14a3-ef3f-4ab8-8a94-dd3a8f8c0ae5 [UUID(f7ec14a3)]:
Workflow w complete. Final Outputs:
  {
    "w.hello.out": "Hello Amazon Genomics CLI!"
  }
  2021-07-07 04:55:41,720 cromwell-system-akka.dispatchers.engine-dispatcher-20
INFO  - WorkflowManagerActor WorkflowActor-f7ec14a3-ef3f-4ab8-8a94-dd3a8f8c0ae5 is in
a terminal state: WorkflowSucceededState
  2021-07-07 04:55:41,720 cromwell-system-akka.dispatchers.engine-dispatcher-20
INFO  - WorkflowManagerActor WorkflowActor-cd7b3d2d-6e57-4957-897b-4b092331da96 is in
a terminal state: WorkflowSucceededState
```

Cleanup

When you are done running workflows, it is recommended you stop all cloud resources to save costs.

Stop a context with:

```
agc context stop <context-name>
```

This will destroy all compute resources, but retain any data in S3.

If you want stop using Amazon Genomics CLI in your AWS account entirely, you need to deactivate it:

```
agc account deactivate
```

This will remove Amazon Genomics CLI's core infrastructure. If Amazon Genomics CLI created a VPC as part of the `activate` process, it will be **removed**. If Amazon Genomics CLI created an S3 bucket for you it will be **retained**.

To uninstall Amazon Genomics CLI from your local machine, run the following command:

```
./agc/uninstall.sh
```

> *Note uninstalling the CLI will not remove any resources or persistent data from your AWS account.*

Release Notes

0.8.1 (Glutamine hotfix)

- Patch issue with ts-node affecting CDK deployments

0.8.0 (Glutamine)

- Use our official name "Amazon Genomics CLI" and abbreviation "AGC"
- Local workflow inputs are automatically uploaded to S3 during a run
- When running a workflow with local inputs, only changed local files are uploaded to S3
- DNS Namespace and Cloudmap Naming include project and context names
- Added retry logic for task data localization from S3
- Check for container image availability before deploying Fargate tasks