

# Documentation

- 1: [Overview](#)
- 2: [Getting Started](#)
  - 2.1: [Prerequisites](#)
  - 2.2: [Installation](#)
  - 2.3: [Setup](#)
  - 2.4: [Hello world](#)
- 3: [Examples](#)
- 4: [Concepts](#)
  - 4.1: [Accounts](#)
  - 4.2: [Users](#)
  - 4.3: [Projects](#)
  - 4.4: [Contexts](#)
  - 4.5: [Data](#)
  - 4.6: [Workflows](#)
  - 4.7: [Engines](#)
  - 4.8: [Logs](#)
  - 4.9: [Namespaces](#)
- 5: [Tutorials](#)
  - 5.1: [Walk through](#)
- 6: [Best Practices](#)
  - 6.1: [IAM Permissions](#)
  - 6.2: [Controlling Costs](#)
  - 6.3: [Scaling Workloads](#)
- 7: [Workflow Engines](#)
  - 7.1: [Filesystems](#)
    - 7.1.1: [EFS Workflow Filesystem](#)
    - 7.1.2: [S3 Workflow Filesystem](#)
  - 7.2: [miniwdl](#)
  - 7.3: [Toil](#)
  - 7.4: [Cromwell](#)
  - 7.5: [Nextflow](#)
  - 7.6: [Snakemake](#)
- 8: [Contribution Guidelines](#)
- 9: [Reference](#)
  - 9.1: [agc](#)
  - 9.2: [agc account](#)
  - 9.3: [agc account activate](#)
  - 9.4: [agc account deactivate](#)
  - 9.5: [agc completion](#)
  - 9.6: [agc completion bash](#)
  - 9.7: [agc completion fish](#)

- 9.8: [agc completion powershell](#)
- 9.9: [agc completion zsh](#)
- 9.10: [agc configure](#)
- 9.11: [agc configure describe](#)
- 9.12: [agc configure email](#)
- 9.13: [agc configure format](#)
- 9.14: [agc context](#)
- 9.15: [agc context deploy](#)
- 9.16: [agc context describe](#)
- 9.17: [agc context destroy](#)
- 9.18: [agc context list](#)
- 9.19: [agc context status](#)
- 9.20: [agc logs](#)
- 9.21: [agc logs access](#)
- 9.22: [agc logs adapter](#)
- 9.23: [agc logs engine](#)
- 9.24: [agc logs workflow](#)
- 9.25: [agc project](#)
- 9.26: [agc project describe](#)
- 9.27: [agc project init](#)
- 9.28: [agc project validate](#)
- 9.29: [agc workflow](#)
- 9.30: [agc workflow describe](#)
- 9.31: [agc workflow list](#)
- 9.32: [agc workflow output](#)
- 9.33: [agc workflow run](#)
- 9.34: [agc workflow status](#)
- 9.35: [agc workflow stop](#)

# 1 - Overview

Amazon Genomics CLI Overview.

## Attention

**The Amazon Genomics CLI project has entered its End Of Life (EOL) phase.** The code is no longer actively maintained and the **Github repository will be archived on May 31 2024**. During this time, we encourage customers to migrate to [AWS HealthOmics](#) to run their genomics workflows on AWS, or [reach out to their AWS account team](#) for alternative solutions. While the source code of AGC will still be available after the EOL date, we will not make any updates inclusive of addressing issues or accepting Pull Requests.

# What is the Amazon Genomics CLI?

Amazon Genomics CLI is an open source tool for genomics and life science customers that simplifies and automates the deployment of cloud infrastructure, providing you with an easy-to-use command line interface to quickly setup and run genomics workflows on Amazon Web Services (AWS) specified by languages like CWL, Nextflow, Snakemake, and WDL. By removing the heavy lifting from setting up and running genomics workflows in the cloud, software developers and researchers can automatically provision, configure and scale cloud resources to enable faster and more cost-effective population-level genetics studies, drug discovery cycles, and more.

## Why do I want it?

Amazon Genomics CLI is targeted at bioinformaticians and genomics analysts who are not experts in cloud infrastructure and best practices. If you want to take advantage of cloud computing to run your workflows at scale, but you don't want to become an expert in high performance batch computing and distributed systems then Amazon Genomics CLI is probably for you.

- **What is it good for?:** Abstracting the infrastructure needed to run workflows from the running of the workflows by hiding all the complexity behind a familiar CLI interface. When you need to get your workflows running quickly and are happy to let the tool make the decisions about the best infrastructure according to AWS best practices.
- **What is it not good for?:** Situations where you want, or need, complete and fine-grained control over how your workflows are run in the cloud. Where specifying *exactly* how they run, and what infrastructure is used, is just as important as running them.
- **What is it *not yet* good for?:** [Let us know](#), we'd love to hear your suggestions on how we can make the tool work for you.

## Where should I go next?

Take a look at the following pages to help you start running your workflows quickly:

- [Getting Started](#): Get started with Amazon Genomics CLI
- [Tutorials](#): Tutorials to show you the ropes.

## 2 - Getting Started

What does your user need to know to try your project?

### Attention

**The Amazon Genomics CLI project has entered its End Of Life (EOL) phase.** The code is no longer actively maintained and the **Github repository will be archived on May 31 2024**. During this time, we encourage customers to migrate to [AWS HealthOmics](#) to run their genomics workflows on AWS, or [reach out to their AWS account team](#) for alternative solutions. While the source code of AGC will still be available after the EOL date, we will not make any updates inclusive of addressing issues or accepting Pull Requests.

The following links will help you install Amazon Genomics CLI and quickly run a demo workflow.

## 2.1 - Prerequisites

To run Amazon Genomics CLI the following prerequisites must be met:

- A computer with one of the following operating systems:
  - macOS 10.14+
  - Amazon Linux 2
  - Ubuntu 20.04
  - Windows 10 with a Windows subsystem running Ubuntu which runs the commands
- Internet access
- An AWS Account
- An AWS role with sufficient access. To generate the minimum required policies for admins and users, please follow the instructions [here](#)

Running Amazon Genomics CLI on Windows has not been tested, but it should run in WSL 2 with Ubuntu 20.04

## Prerequisite installation

### Ubuntu 20.04

- Install node.js

```
curl -fsSL https://deb.nodesource.com/setup_15.x | sudo  
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 (e.g. on an EC2 instance)

- Install node

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

- If you have not already done so, configure your AWS credentials and default region

```
aws configure
```

### MacOS

- Install [Homebrew](#)

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com
```

- Install node

```
brew install node
```

- Install and configure AWS CLI

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

## 2.2 - Installation

### Download and install Amazon Genomics CLI

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

To download a specific release, see [releases page](#) of our Github repo.

To download the latest release navigate to <https://github.com/aws/amazon-genomics-cli/releases/>

Once you have downloaded a release, type the following to install:

The latest nightly build can be accessed here: `s3://healthai-public-assets-us-east-1/amazon-genomics-cli/nightly-build/amazon-genomics-cli.zip`

You can download the nightly by running the following:

```
aws s3api get-object --bucket healthai-public-assets-us-east-1
```

```
unzip amazon-genomics-cli-<version>.zip
cd amazon-genomics-cli/
./install.sh
```

This will place the `agc` command in `$HOME/bin`.

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 c

Logs

    logs         Commands for various logs.

Projects

    project      Commands to interact with projects.

Workflows

    workflow     Commands for workflows.  
                  Workflows are potentially-dynamic graph

Settings ⚙️

    configure    Commands for configuration.  
                  Configuration is stored per user.

Flags

    --format string   Format option for output. Valid

    -h, --help        help for agc

    --silent          Suppresses all diagnostic infor

    -v, --verbose     Display verbose diagnostic info

    --version         version for agc

Examples

    Displays the help menu for the specified sub-command.

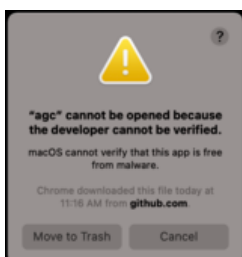
    `\$ agc account --help`

If this doesn't work immediately, try:

- start a new terminal shell
- modifying your `$HOME/.bashrc` (or equivalent file)  
    appending the following line and restarting your shell:

```
export PATH=$HOME/bin:$PATH
```

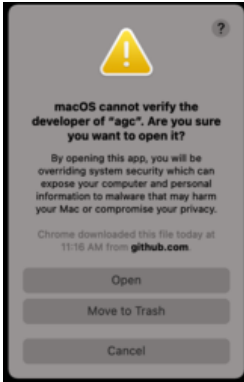
If you are running this on MacOS, you may see this below popup window when you initially run any agc commands due to Apple's security restrictions.





Click Cancel and navigate to Apple's System Preferences, click Security & Privacy, then click General. Near the bottom, you will see a line indicating "agc" was blocked from use because it is not from an identified developer. To the right, click Allow Anyway.

Now go back to the terminal and run `agc --help` again. You will see this new popup window below asking you to override the system security.



Click Open and now your `agc` is correctly installed.

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

```
agc --version
```

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

## Command Completion

Amazon Genomics CLI can generate shell completion scripts that enable 'Tab' completion of commands. Command completion is optional and not required to use Amazon Genomics CLI. To generate a completion script you can use:

```
agc completion <shell>
```

where "shell" is one of:

### Bash

```
source <(agc completion bash)
```

To load completions for each session, execute once:

## Linux:

```
agc completion bash > /etc/bash_completion.d/agc
```

## macOS:

If you haven't already installed `bash-completion`, execute the following once

```
brew install bash-completion
```

and then, add the following line to your `~/.bash_profile`:

```
[[ -r "/usr/local/etc/profile.d/bash_completion.sh" ]]
```

Once bash completion is installed

```
agc completion bash > /usr/local/etc/bash_completion.d,
```

## Zsh:

If shell completion is not already enabled in your environment, you will need to enable it. You can execute the following once:

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
```

To load completions for each session, execute once:

```
agc completion zsh > "${fpath[1]}/_agc"
```

You will need to start a new shell for this setup to take effect.

## fish:

```
agc completion fish | source
```

To load completions for each session, execute once:

```
agc completion fish > ~/.config/fish/completions/agc.fish
```

PowerShell:

```
agc completion powershell | Out-String | Invoke-Expression
```

To load completions for every new session, run:

```
agc completion powershell > agc.ps1
```

and source this file from your PowerShell profile.

## 2.3 - Setup

### 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 a DynamoDB table, an S3 bucket and a VPC. This will take ~5min to complete. You only need to do this once per account region.

The DynamoDB table is used by the CLI for persistent state. The S3 bucket is used for durable workflow data and Amazon Genomics CLI 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.

### CDK Bootstrap

#### Attention

This step is NOT required when using Amazon Genomics CLI version 1.2 or above

Amazon Genomics CLI uses AWS CDK to deploy infrastructure. Activating an account will bootstrap the AWS Environment for CDK app deployments. CDK Bootstrap deploys the infrastructure needed to allow CDK to deploy CDK defined infrastructure. Full details are available [here](#).

### Define a username

Amazon Genomics CLI requires that you define a username and email. You can do this using the following command:

```
agc configure email you@youreemail.com
```

The username only needs to be configured once per computer that you use Amazon Genomics CLI from.

## 2.4 - Hello world

When you install Amazon Genomics CLI it will create a folder named `agc`. Inside there is an `examples/demo-project` folder containing an `agc-project.yaml` with some demo projects including a simple “hello world” workflow.

### Running Hello World

1. Ensure you have met the [prerequisites](#) and [installed](#) Amazon Genomics CLI
2. Ensure you have followed the [activation](#) steps
3. `cd ~/amazon-genomics-cli/examples/demo-wdl-project`
4. `agc context deploy --context myContext`, this step takes approximately 5 minutes to deploy the infrastructure
5. `agc workflow run hello --context myContext`, take note of the returned workflow instance ID.
6. Check on the status of the workflow `agc workflow status -r <workflow-instance-id>`. Initially you will see status like `SUBMITTED` but after the elastic compute resources have been spun up and the workflow runs you should see something like the following: `WORKFLOWINSTANCE myContext 9ff7600a-6d6e-4bda-9ab6-c615f5d90734 COMPLETE 2021-09-01T20:17:49Z`

Congratulations! You have just run your first workflow in the cloud using Amazon Genomics CLI! At this point you can run additional workflows, including submitting several instances of the “hello world” workflow. The elastic compute resources will expand and contract as necessary to accommodate the backlog of submitted workflows.

### Reviewing the Results

Workflow results are written to an S3 bucket specified or created by Amazon Genomics CLI during account activation. 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. Workflow output will be in the `s3://agc-<account-num>-<region>/project/<project-name>/userid/<user-id>/context/<context-name>/workflow/<workflow-name>/` path. The rest of the path

depends on the engine used to run the workflow. For Cromwell it will continue with: `.../cromwell-execution/<wdl-wf-name>/<workflow-run-id>/<task-name>`

If a workflow declares outputs then you may obtain these using the command:

```
agc workflow output <workflow_run_id>
```

You should see a response similar to:

```
OUTPUT id 6cc6f742-dc87-4649-b319-1af45c4c09c6
OUTPUT outputs.hello_agc.hello.out Hello Amazon Genom:
```

You can also obtain task logs for a workflow using the following form `agc logs workflow <workflow-name> -r <instance-id> .`

Note, if the workflow did not actually run any tasks due to call caching then there will be no output from this command.

## Cleaning Up

Once you are done with `myContext` you can destroy it with:

```
agc context destroy myContext
```

This will remove the cloud resources associated with the named context, but will keep any S3 outputs and CloudWatch logs.

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.

## Next Steps

- Familiarize yourself with [Amazon Genomics CLI Concepts](#)
- Try some [tutorials](#)

## 3 - Examples

Amazon Genomics CLI in action!

### Attention

**The Amazon Genomics CLI project has entered its End Of Life (EOL) phase.** The code is no longer actively maintained and the **Github repository will be archived on May 31 2024**. During this time, we encourage customers to migrate to [AWS HealthOmics](#) to run their genomics workflows on AWS, or [reach out to their AWS account team](#) for alternative solutions. While the source code of AGC will still be available after the EOL date, we will not make any updates inclusive of addressing issues or accepting Pull Requests.

As part of our GitHub distribution we provide some example projects along with their workflows in the `examples/` folder. These projects are also included in the `$HOME/agc/examples` folder that is created when you install Amazon Genomics CLI.

The `demo-project` shows some basic concepts and tests, while the `gatk-best-practices-project` provides some real world genetics workflows. We also include a `demo-nextflow` project to show some basic examples of running Nextflow workflows in Amazon Genomics CLI.

See also the [Getting Started](#) and [Tutorials](#) sections.



## 4 - Concepts

What do you need to know about Amazon Genomics CLI in order to use it - or potentially contribute to it?

### Attention

**The Amazon Genomics CLI project has entered its End Of Life (EOL) phase.** The code is no longer actively maintained and the **Github repository will be archived on May 31 2024**. During this time, we encourage customers to migrate to [AWS HealthOmics](#) to run their genomics workflows on AWS, or [reach out to their AWS account team](#) for alternative solutions. While the source code of AGC will still be available after the EOL date, we will not make any updates inclusive of addressing issues or accepting Pull Requests.

For a general introduction to the AWS Genomics CLI, refer to the [Overview](#).

Amazon Genomics CLI uses a handful of core concepts to abstract the deployment of infrastructure needed to run workflows and to organize workflows and dependencies. Gaining an understanding of the concepts below will help you understand how Amazon Genomics CLI works and how it is organized.

## 4.1 - Accounts

How AWS Genomics CLI interacts with AWS Accounts

Amazon Genomics CLI requires an AWS account in which to deploy the cloud infrastructure required to run and manage workflows. To begin working with Amazon Genomics CLI and account must be “Activated” by the Amazon Genomics CLI application using the [account activate](#) command.

### Which AWS Account is Used by Amazon Genomics CLI?

Amazon Genomics CLI uses the same [AWS credential chain](#) used by the AWS CLI to determine what account should be used and with what credentials. All that is required is that you have an existing AWS account (or create a new one) which contains at least one IAM Principal (User/ Role) that you can access.

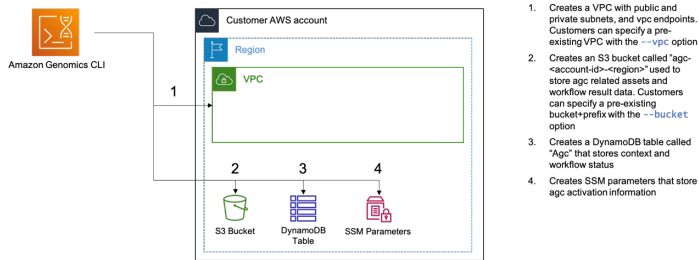
### Which Region is Used by Amazon Genomics CLI?

Much like accounts and credentials, Amazon Genomics CLI uses the same chain used by the AWS CLI to determine the region that is being targeted. For example, if your AWS profile uses `us-east-1` then Amazon Genomics CLI will use the same. Likewise, if you set the `AWS_REGION` environment variable to `eu-west-1` then that region will be used by Amazon Genomics CLI for all subsequent commands in that shell.

## Shared Infrastructure

When a region is first activated for Amazon Genomics CLI, some basic infrastructure is deployed including a [VPC](#), which is used for the compute infrastructure that will be deployed in a [context](#), and an [S3](#) bucket which will be used to store workflow intermediates and results. This core infrastructure will be shared by all Amazon Genomics CLI users and projects in that region.

The following diagram shows the infrastructure deployed when the command `agc account activate` is run:



Note that context specific infrastructure is not shared and is unique and namespaced by user and project.

## Bring your Own VPC and S3 Bucket

During account [activation](#) you may specify an existing VPC ID or S3 bucket name for use by Amazon Genomics CLI. If you do not these will be created for you. Although we use AWS best practices for these, if your organization has specific security requirements for networking and storage this may be the easiest way to activate Amazon Genomics CLI in your environment.

## Account Commands

A full reference of the account commands is [here](#)

### activate

You can activate an account using `agc account activate`. An account must be activated before any contexts can be deployed or workflows run.

Activating an account will also bootstrap the AWS Environment for CDK app deployments.

### Using an Existing S3 Bucket

Amazon Genomics CLI requires an S3 bucket to store workflow results and associated information. If you prefer to use an existing bucket you can use the form `agc account activate --bucket my-existing-bucket`. If you do this the AWS [IAM](#) role used to run Amazon Genomics CLI must be able to write to that bucket.

### Using an Existing VPC

To use an existing VPC you can use the form `agc account activate --vpc my-existing-vpc-id`. This VPC must have at least 3 availability zones each with at least one private subnet. The private subnets must have connectivity to the internet, such

as via a NAT gateway, and connectivity to AWS services either through VPC endpoints or the internet. Amazon Genomics CLI will not modify the network topology of the specified VPC.

## Specifying Subnets

When using an existing VPC you may need to specify which subnets of the VPC can be used for infrastructure. This is useful when only some private subnets have internet routing. To do this you can supply a comma separated list of subnet IDs using the `--subnets` flag, or repeat the flag multiple times. For example:

```
agc account activate --vpc my-existing-vpc-id --subnets
subnet-id-1,subnet-id-2 --subnets subnet-id-3
```

We recommend a minimum of 3 subnets across availability zones to take advantage of EC2 instance availability and to ensure high availability of infrastructure.

## Using a Specific AMI for Compute Environments

Some organizations restrict the use of AMIs to a pre-approved list. By default, Amazon Genomics CLI uses the most recent version of the Amazon Linux 2 ECS Optimized AMI. To change this behavior you can supply the ID of an alternative AMI at account activation. This AMI will then be used for all compute environments used by all newly deployed contexts.

```
agc account activate --ami <ami-id>
```

There are some specific requirements that the AMI must comply with. It must be a private AMI from the same account that you will use for deploying Amazon Genomics CLI infrastructure. It must also be capable of successfully running all parts of the [LaunchTemplate](#) executed at startup time including the [ecs-additions](#) dependencies. We recommend an ECS optimized image based on Amazon Linux 2, RHEL, Fedora or similar.

If the LaunchTemplate cannot complete successfully it will result in an EC2 instance that cannot join a compute-cluster and cannot complete workflow tasks. A common symptom of this is workflow tasks that become stuck in a “runnable” state but are never assigned to a cluster node.

## Using Only Public Subnets

Amazon Genomics CLI can create a new VPC with only public subnets to use for its infrastructure using the `--usePublicSubnets` flag.

```
agc account activate --usePublicSubnets
```

This can reduce costs by removing the need for NAT Gateways and VPC Gateway Endpoints to route internet traffic from private subnets. It can also reduce the number of Elastic IP Addresses consumed by your infrastructure.

## Warning

When using a VPC with only public subnets, you will need to ensure that the contexts defined in `agc-project.yaml` files declare that they will use public subnets. For example:

```
contexts:
  myContext:
    usePublicSubnets: true
    engines:
      - type: nextflow
        engine: nextflow
```

## Warning

Currently, use of public subnets is only supported for contexts that use the Nextflow engine. Use of public IPs with the Cromwell server creates a security risk and will fail. Assignment of public IPs to AWS Batch Fargate tasks (as used by miniwdl and SnakeMake) is possible but will require changes to the WES adapters of those engines. If you need this please file a [feature request](#) with your use case

## Security Considerations

Although your infrastructure will be protected by security groups you should be aware that any manual modification of these may result in exposing your infrastructure to the internet. For this reason *we do **not** recommend using this configuration in production or with sensitive data.*

## Updating

Issuing `account activate` commands more than once effectively updates the core infrastructure with the difference between the two commands according to the rules below.

### Updating the VPC

You may change the VPC used by issuing the command `agc account activate --vpc <vpc-id>`. If a `--vpc` argument is *not* provided as part of an `agc account activate` command then the last VPC used will be 'remembered' and used by default.

If you wish to change to use a new default VPC created by Amazon Genomics CLI you must deactivate ( `agc account deactivate` ) and reactivate with no `--vpc` flag.

```
agc account activate           # VPC 1 created.
agc account activate --vpc-id abc # VPC 1 destroyed and
agc account activate           # VPC 2 created. Custom
agc account deactivate         # AGC core infrastru
```

## Updating to Use Public Subnets Only

If you wish to change the VPC to use public subnets only, or change it from public subnets to private subnets you must deactivate the account and reactivate it with (or without) the `--usePublicSubnets` flag. For example:

```
agc account activate --usePublicSubnets # New VPC with
agc account deactivate                 # VPC destroyed
agc account activate                   # New VPC with
```

## Updating Selected Subnets

To change a VPC to use a different selection of subnets you must supply both the VPC id and the required subnet IDs. If you omit the `--subnets` flag, then future context deployments will use *all* private subnets of the VPC.

```
agc account activate --vpc <vpc-id> --subnets <subnet1,
agc account activate --vpc <vpc-id> --subnets <subnet1,
agc account activate --vpc <vpc-id>
```

## Updating the Compute-Environment AMI

The compute-environment AMI can be changed by re-issuing the `account activate` command with (or without) the `--ami` flag. If the flag is not provided the latest Amazon Linux 2 ECS optimized image will be used.

```
agc account activate           # Latest Amazon
agc account activate --ami <ami-1234> # AMI 1234 used
agc account activate           # Latest Amazon
```

## deactivate

The `deactivate` command is used to remove the core infrastructure deployed by Amazon Genomics CLI in the current region when an account is activated. The S3 bucket deployed by Amazon Genomics CLI and its contents are retained. If a VPC

and/ or S3 bucket were specified by the user during account activation these will also be retained. Any CloudWatch logs produced by Amazon Genomics CLI will also be retained.

If there are existing deployed contexts the command will fail, however, you can force the removal of these at the same time with the `--force` flag. Note that this will also interrupt any running workflow of any user in that region.

The deactivate command will only operate on infrastructure in the current region.

If the deployed infrastructure has been modified through the console or the AWS CLI rather than through Amazon Genomics CLI deactivation may fail due to the infrastructure state being inconsistent with the [CloudFormation](#) state. If this happens you may need to manually clean up through the CloudFormation console.

## Costs

Core infrastructure deployed for Amazon Genomics CLI is [tagged](#) with the `application-name: agc` tag. This tag can be activated for cost tracking in [AWS CostExplorer](#). The core infrastructure is shared and *not* tagged with any username, context name or project name.

While an account region is activated there will be ongoing charges from the core infrastructure deployed including things such as VPC NAT gateways and VPC Endpoints. If you no longer use Amazon Genomics CLI in a region we recommend you deactivate it. You may also wish to remove the S3 bucket along with its objects as well as the [CloudWatch](#) logs produced by Amazon Genomics CLI. These are retained by default so that you can view workflow results and logs even after deactivation.

However, if you wish to have this infrastructure remain deployed, you are able to significantly reduce ongoing costs by using `agc account activate --usePublicSubnets`. This prevents the creation of private subnets with NAT gateways, and the use of VPC endpoints, both of which have associated ongoing costs. Please note that **you must also set `usePublicSubnets: true` in your `agc-config.yaml` if you choose to use this option**. Please also note that this is not recommended for security-critical deployments, as it means that any edits to the stack security groups risk exposing worker nodes to the public internet.

## Network traffic

When running genomics workflows, network traffic can become a significant expense when the traffic is routed through NAT gateways into private subnets (where worker nodes are usually located). To minimize these costs we recommend the use of VPC

Endpoints ([see below](#)) as well as activating Amazon Genomics CLI and running your workflows in the same region as your S3 bucket holding your genome files. VPC Gateway endpoints are regional so cross region S3 access will *not* be routed through a VPC gateway.

If you make use of large container images in your workflows (such as the GATK images) we recommend copying these to a private [ECR](#) repository in the same region that you will run your analysis to use ECR endpoints and avoid traffic through NAT gateways.

## VPC Endpoints

When Amazon Genomics CLI creates a VPC it creates the following VPC endpoints:

- `com.amazonaws.{region}.dynamodb`
- `com.amazonaws.{region}.ecr.api`
- `com.amazonaws.{region}.ecr.dkr`
- `com.amazonaws.{region}.ecs`
- `com.amazonaws.{region}.ecs-agent`
- `com.amazonaws.{region}.ecs-telemetry`
- `com.amazonaws.{region}.logs`
- `com.amazonaws.{region}.s3`
- `com.amazonaws.{region}.ec2`

If you provide your own VPC we recommend that the VPC has these endpoints. This will improve the security posture of Amazon Genomics CLI in your VPC and will also reduce NAT gateway traffic charges which can be substantial for genomics analyses that use large S3 objects and/ or large container images.

If you are using Amazon Genomics CLI client on an EC2 instance in a subnet with no access to the internet you will need to have a VPC endpoint to `com.amazonaws.{region}.execute-api` so that the client can make calls to the REST services deployed during account activation.

## Technical Details.

Amazon Genomics CLI core infrastructure is defined in code and deployed by [AWS CDK](#). The CDK app responsible for creating the core infrastructure can be found in [packages/cdk/apps/core/](#).



## 4.2 - Users

How Amazon Genomics CLI identifies users

When the CLI is set up, the user of the CLI is defined using the `agc configure email` command. This email should be unique to the individual user. This email address is used to determine a unique user ID which will be used to uniquely identify infrastructure belonging to that user.

### Amazon Genomics CLI Users are Not IAM Users (or Principals)

Amazon Genomics CLI users are primarily used for identification and as a component of namespacing. They are not a security measure, nor are they related to IAM users or roles. All AWS activities carried out by Amazon Genomics CLI will be done using the AWS credentials in the environment where the CLI is installed and are *not* based on the Amazon Genomics CLI username.

For example. If Amazon Genomics CLI is installed on an EC2 instance and configured with the email `someone@company.com` Amazon Genomics CLI will interact with AWS resources based solely on the IAM Role assigned to that EC2 via it's instance profile. Like wise if you use Amazon Genomics CLI on your laptop then the IAM role that you use will be determined by the same process as is used by the AWS CLI.

### Who am I?

To find out what username and email has been configured in your current environment you can use the following command:

```
agc configure describe
```

### Changing user

If you update your configured email, a new user identity is generated. If this is done while infrastructure is deployed, Amazon Genomics CLI may no longer be able to identify that infrastructure as belonging to your project. We strongly recommend stopping all running workflows and destroying all your deployed contexts from *all* projects before changing user.

If you do not do this, you or an account administrator will need to identify any un-needed infrastructure in the CloudFormation console and remove it from there.

## 4.3 - Projects

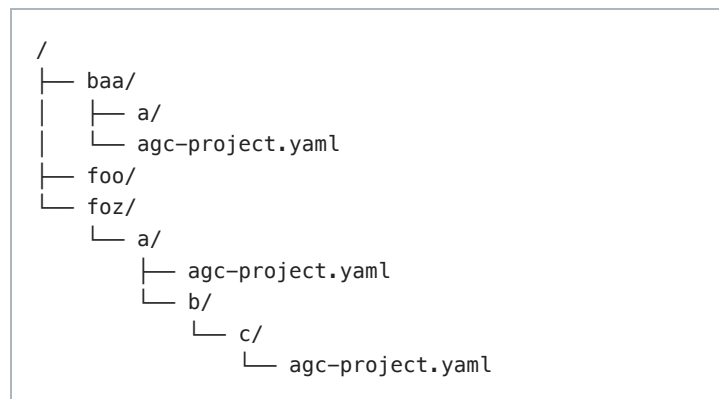
A project defines the contexts, engines, data and workflows that make up a genomics analysis

An Amazon Genomics CLI project defines the [projects](#), [contexts](#), [data](#) and [workflows](#) that make up a genomics analysis. Each project is defined in a project file named `agc-project.yaml`.

### Project File Location

To find the project definition, Amazon Genomics CLI will look for a file named `agc-project.yaml` in the current working directory. If the file is not found, Amazon Genomics CLI will traverse up the file hierarchy until the file is found or until the root of the file system is reached. If no project definition can be found an error will be reported. All Amazon Genomics CLI commands operate on the project identified by the above process.

Consider the example directory structure below:



- If the current working directory is `/baa` or `/baa/a` then `/baa/agc-project.yaml` will be used for definitions,
- If the current working directory is `/foo` an error will be reported as no project file is found before the root,
- If the current working directory is `/foz` an error will be reported as no project file is found before the root,
- If the current working directory is `/foz/a` or `/foz/a/b` then `/foz/a/agc-project.yaml` will be used for definitions.
- If the current working directory is `/foz/a/b/c` then `/foz/a/b/c/agc-project.yaml` will be used for definitions.

### Relative Locations

The location of resources declared in a project file are resolved relative to the location of the project file *unless* they are declared using an absolute path. If the project file in `/baa`

declared that there was a workflow definition in `a/b/` then Amazon Genomics CLI will search for that definition in `/baa/a/b/`.

## Project File Structure

A minimal project file can be generated using the `agc project init myProject --workflow-type nextflow`. Using `myProject` as a project name and workflow type `nextflow` will result in the following:

```
name: myProject
schemaVersion: 1
contexts:
  ctx1:
    engines:
      - type: nextflow
        engine: nextflow
```

This is fully usable project called “myProject” with a single context named “ctx1”. At this point “ctx1” can be [deployed](#) however, there are currently no workflows defined.

### name

A string that identifies the project

### schemaVersion

An integer defining the schema version. Version numbers will be incremented when changes are made to the project schema that are not backward compatible.

### contexts

A map of context names to context definitions. Each context in the project must have a unique name. The [contexts](#) documentation provides more details.

### workflows

A map of workflow names to workflow definitions. Workflow names must be unique in a project. The [workflows](#) documentation provides more details.

### data

An array of data sources that the contexts of the project have access to. For example:

```
data:
- location: s3://gatk-test-data
  readOnly: true
- location: s3://broad-references
  readOnly: true
- location: s3://1000genomes-dragen-3.7.6
  readOnly: true
```

You can use S3 prefixes to be more restrictive about access to data. For example, if you want to allow access to the `foo` folder of `my-bucket` and its sub-folders you would declare the location as:

```
data:
- location: s3://my-bucket/foo/*
```

You can also grant access to a specific object (only) by providing the full path of the object. For example:

```
data:
- location: s3://my-bucket/foo/object
```

## Commands

A full reference of project commands are available [here](#)

### init

The `agc project init <project-name> --workflow-type <worklow-type>` command can be used to initialize a minimal `agc-project.yaml` file in the current directory. Alternatively project yaml files can be created with any text editor.

### describe

The `agc project describe <project-name>` command will provide basic metadata about the 'local' project file. See [above](#) for details on how project files are located.

### validate

Using `agc project validate` you can quickly identify any syntax errors in your local project file.

## Versioning and Sharing

We recommend placing a project under source version control using a tool like [Git](#). The folder containing the `agc-project.yaml` file is a natural location for the root of a Git repository. Workflows relating to the project would naturally be located in sub-folders of the same repository allowing those to be versioned as well. Alternatively, more advanced Git users may consider storing workflows as a Git [sub-module](#) allowing them to be independent of the project and reused among projects.

Projects and associated workflows can then be shared by “pushing” the project’s Git repository to a website such as [GitHub](#), [GitLab](#), or [BitBucket](#) or hosted on a private Git Server like [AWS Code Commit](#). To facilitate sharing you should ensure that any file paths in your definitions are relative to the project and not absolute. You will also need to make sure that data locations are appropriately shared.

## Costs

A project itself doesn’t have infrastructure. It is not deployed and therefore has no direct costs. If the contexts defined by an infrastructure are deployed or the workflows run then those *will* incur costs.

## Tags

The project `name` will be [tagged](#) on any deployed contexts or workflows defined in this project allowing costs to be aggregated to the project level.

## Technical Details

A project is purely a [YAML](#) definition. The values in the `agc-project.yaml` file are used by CDK when Amazon Genomics CLI deploys contexts and when Amazon Genomics CLI runs workflows. The project itself has no direct infrastructure. The project `name` is used to help namespace context infrastructure.

## 4.4 - Contexts

Contexts are the set of cloud resources used to run a workflow

### What is a Context?

A context is a set of cloud resources. Amazon Genomics CLI runs [workflows](#) in a context. A deployed context will include an [engine](#) that can interpret and manage the running of a workflow along with compute resources that will run the individual tasks of the workflow. The deployed context will also contain any resources needed by the engine or compute resources including any security, permissions and [logging](#) capabilities. Deployed contexts are [namespaced](#) based on the user, project and context name so that resources are isolated, preventing collisions.

When a workflow is run the user will decide which context will run it. For example, you might choose to submit a workflow to a context that uses "Spot priced" resources or one that uses "On Demand" priced resources.

When deployed context resources that require a VPC will be deployed into the VPC that was specified when the [account](#) was activated.

### How is a Context Defined?

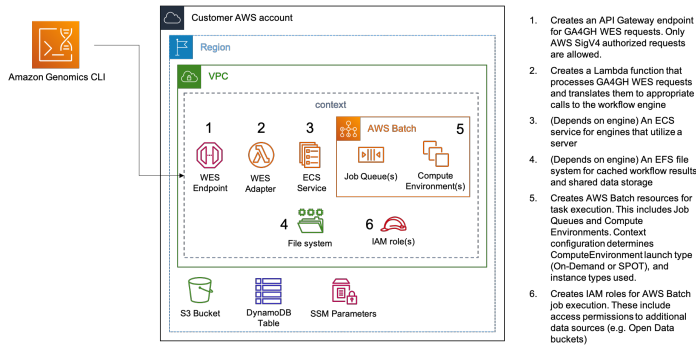
A context is defined in the YAML file that defines the [project](#). A project has at least one context but may have many. Contexts must have unique names and are defined as YAML maps.

A context may request use of [Spot priced](#) compute resources with `requestSpotInstances: true`. The default value is `false`.

A context must define an array of one or more `engines`. Each engine definition must specify the workflow language that it will interpret. For each language Amazon Genomics CLI has a default engine however, users may specify the exact engine in the `engine` parameter.

### General Architecture of a Context

The exact architecture of a context will depend on the context properties described below and defined in their `agc-project.yaml`. However, the architecture deployed on execution of `agc context deploy` is shown in the following diagram:



## Context Properties

### Instance Types

You may optionally specify the instance types to be used in a context. This can be a specific type such as `r5.2xlarge` or it can be an instance family such as `c5` or a combination. By default, a context will use instance types up to `4xlarge`

Note, if you only specify large instance types you will be using those instances for running even the smallest tasks so we recommend including smaller types as well.

Ensure that any custom types you list are available in the region that you're using with Amazon Genomics CLI or the context will fail to deploy. You can obtain a list using the following command

```
aws ec2 describe-instance-type-offerings \
  --region <region_name>
```

### Examples

The following snippet defines two contexts, one that uses spot resources and one that uses on demand. Both contain a WDL engine.

```
...
contexts:
  # The on demand context uses on demand EC2 instances
  onDemandCtx:
    requestSpotInstances: false
    engines:
      - type: wdl
        engine: cromwell
```



```
# The spot context uses EC2 spot instances which are
spotCtx:
  requestSpotInstances: true
  engines:
    - type: wdl
      engine: cromwell
  ...
```

The following context may use any instance type from the m5 , c5 or r5 families

```
contexts:
  nfLargeCtx:
    instanceTypes: [ "c5", "m5", "r5" ]
    engines:
      - type: nextflow
        engine: nextflow
```

## Max vCpus

*default: 256*

You may optionally specify the maximum number of vCpus used in a context. This is the max total amount of vCpus of all the jobs currently running within a context. When the max has been reached, additional jobs will be queued.

*note:* if your vCPU limit is lower than maxVCpus then you won't get as many as requested and would need to make a limit increase.

```
contexts:
  largeCtx:
    maxVCpus: 2000
    engines:
      - type: nextflow
        engine: nextflow
```

## Public Subnets

In the interest of saving money, in particular if you intend to have the AGC stack deployed for a long period, you may choose to deploy in “public subnet” mode. To do this, you must first set up the core stack using `aws configure --usePublicSubnets` , which will disable the creation of the NAT gateway and VPC endpoints which present an ongoing cost unrelated to your use of compute resources. After you have done this, you must also set `usePublicSubnets: true` in all contexts you use:

```
contexts:
  someCtx:
    usePublicSubnets: true
  engines:
    - type: nextflow
      engine: nextflow
```

This ensures that the AWS batch instances are deployed into a public subnet, which has no additional cost associated with it. However note that while these instances are given a security group that will block all incoming traffic, this is not as secure as using the default private subnet mode.

## Context Commands

A full reference of context commands is [here](#)

### describe

The command `agc context describe <context-name> [flags]` will describe the named context as defined in the project YAML as well as other relevant account information.

### list

The command `agc context list [flags]` will list the names of all contexts defined in the project YAML file along with the name of the engine used by the context.

### deploy

The command `agc context deploy <context-name> [flags]` is used to deploy the cloud infrastructure required by the context. If the context is already running the existing infrastructure will be updated to reflect changes in project YAML. For example if you added another `data` definition in your project and run `agc context deploy <context-name>` then the deployed context will be updated to allow access to the new data.

All contexts defined in the project YAML can be deployed or updated using the `--all` flag.

Individually named contexts can be deployed or updated as positional arguments. For example: `agc context deploy ctx1 ctx2` will deploy the contexts `ctx1` and `ctx2`.

The inclusion of the `--verbose` flag will show the full CloudFormation output of the context deployment.

### destroy

A contexts cloud resources can be “destroyed” using the `agc context destroy <context-name>` command. This will remove any infrastructure artifacts associated with the context unless they are defined as being retained. Typically, things like logs and workflow outputs on S3 are retained when a context is destroyed.

All deployed contexts can be destroyed using the `--all` flag.

Multiple contexts can be destroyed in a single command using positional arguments. For example: `agc context destroy ctx1 ctx2` will destroy the contexts `ctx1` and `ctx2`.

## status

The status command is used to determine the status of a *deployed* context or context instance. This can be useful to determine if an instance of a particular context is already deployed. It can be used to determine if the deployed context is consistent with the defined context in the project YAML file. For example, if you deploy a context instance and later change the definition of the context in the project YAML file then the running instance will no longer reflect the definition. In this case you may choose to update the deployed instance using the `agc context deploy` command.

Status will only be shown for contexts for the current user in the current AWS region for the current project. To show contexts for another project, issue the command from that project’s home folder (or subfolder). To display contexts for another AWS region, you can use a different AWS CLI profile or set the `AWS_PROFILE` environment variable to the desired region (e.g `export AWS_REGION=us-west-2` ).

### Warning

Because the `status` command will only show contexts that are listed in the project YAML you should take care to `destroy` any running contexts before deleting them from the project YAML.

## Costs

Infrastructure deployed for a context is tagged with the context name as well as username and project name. These tags can be used with AWS CostExplorer to identify the costs associated with running contexts.

A deployed context will incur charges based on the resources being used by the context. If a workflow is running this will include compute costs for running the workflow tasks but some

contexts may include infrastructure that is always “on” and will incur costs even when no workflow is running. If you no longer need a context we recommend pausing or destroying it.

If `requestSpotInstances` is true, the context will use spot instances for compute tasks. The context will set the max price to 100% although if the current price is lower you will pay the lower price. Note that even at 100% spot instances can still be interrupted if total demand for on demand instances in an availability zone exceeds the available pool. For full details see [Spot Instance Interruptions](#) and [EC2 Spot Pricing](#).

## Ongoing Costs

Until a context is destroyed resources that are deployed can incur ongoing costs even if a workflow is not running. The exact costs depend on the configuration of the context.

Amazon Genomics CLI version 1.0.1 and earlier used an AWS Fargate based WES service for each deployed context. The service uses 0.5 vCPU, 4 GB memory and 20 GB base instance storage. Fargate pricing varies by region and is detailed [here](#). The estimated cost is available via [this link](#)

After version 1.0.1, the WES endpoints deployed by Amazon Genomics CLI are implemented with AWS Lambda and therefore use a [pricing model](#) based on invocations.

Contexts using a Cromwell engine run an additional AWS Fargate service for the engine with 2 vCPU, 16 GB RAM and 20 GB of base storage. Additionally, Cromwell is deployed with a standard EFS volume for storage of metadata. EFS [costs](#) are volume based. While relatively small the amount of metadata will expand as more workflows are run. The volume is destroyed when the context is destroyed. An estimated cost for both components is available via [this link](#)

Contexts using the “miniwdl” or “snakemake” engines use EFS volumes as scratch space for workflow intermediates, caches and temporary files. Because many genomics workflows can accumulate several GB of intermediates per run we recommend destroying these contexts when not in use. An estimated cost assuming a total of 500 GB of workflow artifacts is available via [this link](#)

Refer to the [public subnets section](#) if you are concerned about reducing these ongoing costs.

## Tags

All context infrastructure is [tagged](#) with the context name, username and project name. These tags may be used to help differentiate costs.

## Technical Details

Context infrastructure is defined as code as [AWS CDK](#) apps. For examples, take a look at the `packages/cdk` folder. When deployed a context will produce one or more stacks in Cloudformation. Details can be viewed in the Cloudformation console or with the AWS CLI.

A context includes an endpoint compliant with the [GA4GH WES API](#). This API is how Amazon Genomics CLI submits workflows to the context. The context also contains one or more workflow engines. These may either be deployed as long-running services as is the case with Cromwell or as “head” jobs that are responsible for a single workflow, as is the case for NextFlow. Engines run as “head” jobs are started and stopped on demand thereby saving resources.

## Updating Launch Templates

Changes to EC2 LaunchTemplates in CDK result in a new LaunchTemplate version when the infrastructure is updated. Currently, CDK is unable to also update the default version of the template. In addition, any existing AWS Batch Compute Environments will not be updated to use the new LaunchTemplate version. Because of this, whenever a LaunchTemplate is updated in CDK code we recommend destroying any relevant running contexts and redeploying them. An update will *NOT* be sufficient.

## 4.5 - Data

### Data sets

To run an analysis you need data. In the `agc-project.yaml` file of an Amazon Genomics CLI [project](#) `data` is a list of data locations which can be used by the [contexts](#) of the project.

In the example data definition below we are declaring that the project's contexts will be allowed to access the three listed S3 bucket URIs.

```
data:
  - location: s3://gatk-test-data
    readOnly: true
  - location: s3://broad-references
    readOnly: true
  - location: s3://1000genomes-dragen-3.7.6
    readOnly: true
```

The contexts of the project will be *denied* access to all other S3 location except for the S3 bucket created or associated when the [account](#) was initialized by Amazon Genomics CLI.

Declaring access in the project will only ensure your infrastructure is correctly configured to access the bucket. If the target bucket is further restricted, such as by an access control list or bucket policy, you will still be denied access. In these cases you should work with the bucket owner to facilitate access.

### Read and Write

The default value of `readOnly` is `true`. At the time of writing, write access is not supported (except for the Amazon Genomics CLI core S3 bucket)

### Access to a Prefix

The above examples will grant read access to an entire bucket. You can grant more granular access to a prefix within a bucket, for example:

```
data:
  - location: s3://my-bucket/my/prefix/
```

### Cross Account Access

A bucket in another AWS account can be accessed if the owner has set up access, and you are using a role that is allowed access. See [cross account access](#) for details.

## Updating Data Sources

If data definitions are added to or removed from a project definition the change will *not* be reflected in deployed contexts until they are updated. This can be done with `agc context deploy --all` for all contexts or by using a context name to update only one. See [context deploy](#) for details.

### Warning

Removing access to S3 buckets while there are running workflows in a project may cause the workflow to fail if it depends on access to data in those buckets.

## Technical Details

When a context is deployed, IAM roles used by the infrastructure of the context will be granted s3 permissions to perform some S3 read (or read and write) actions on the listed locations. The permissions are defined in CDK code in `/packages/cdk/apps/`. The CDK code does not modify any data in the buckets or any other bucket policies or configurations.

## 4.6 - Workflows

A Workflow is a series of steps or tasks to be executed as part of an analysis.

A Workflow is a series of steps or tasks to be executed as part of an analysis. To run a workflow using Amazon Genomics CLI, first you must have deployed a context with suitable compute resources and with a workflow engine that can interpret the language of the workflow.

### Specification in Project YAML

In an Amazon Genomics CLI project you can specify multiple workflows in a YAML map. The following example defines four WDL version 1.0 workflows. The `sourceURL` property defines the location of the workflow file. If the location is relative then the relevant file is assumed to be relative to the location of the project YAML file. Absolute file locations are also possible although this may reduce the portability of a project if it is intended to be shared. Web URLs are supported as locations of the workflow definition file.

At this time Amazon Genomics CLI does *not* resolve path aliases so, for example, a `sourceURL` like `~/workflows/workflow.wdl` is not supported.

The `type` object declares the `language` of the workflow (eg, `wdl`, `nextflow` etc). The run a workflow there must be a deployed [context](#) with a matching language. The `version` property refers to the workflow language version.

```
workflows:
  hello:
    type:
      language: wdl
      version: 1.0
      sourceURL: workflows/hello.wdl
  read:
    type:
      language: wdl
      version: 1.0
      sourceURL: workflows/read.wdl
  words-with-vowels:
    type:
      language: wdl
      version: 1.0
      sourceURL: workflows/words-with-vowels.wdl
```

### Multi-file Workflows



Some workflow languages allow for the import of other workflows. To accommodate this, Amazon Genomics CLI supports using a directory as a source URL. When a directory is supplied as the `sourceURL`, Amazon Genomics CLI uses the following rules to determine the name of the main workflow file and any supporting files:

1. If the source URL resolves to a single non-zipped file, then the file is assumed to be a workflow file. Dependent resources (if any) are hardcoded in the file and must be resolvable by the Wes adapter or implicitly the workflow engine (e.g the Wes adapter figures out if the engine can resolve them and if not it resolves them itself).
2. The source URL resolves to a zipped file ( `.zip` ). The zip may contain a manifest.
  1. If the zip file does *not* contain a file named `MANIFEST.json` :
    1. The zip file must contain one workflow file with the prefix `main` followed by the conventional suffix for the workflow, e.g. `main.wdl`
    2. Any sub-workflows or tasks referenced by the main workflow must either be in the zip at the appropriate relative path or they must be referenced by URLs that are resolvable by the workflow engine. The WesAdapter may attempt to resolve them for the engine but this is a convenience and not required.
    3. Any variables not defined in the workflows must be provided in an inputs file that is referenced via the input argument in AGC. For workflow engines that support multiple input files an index suffix must be provided (e.g. `inputs_a.json` or `inputs_1.json`) if there is more than one inputs file.
    4. A workflow options file may be included and must be named with the options prefix followed by the conventional suffix of the workflow. The WesAdapter may chose to make use of this depending on the context of the workflow engine. It may also choose to pass this to the workflow engine or pass a modified copy to the workflow engine.
  2. If the zip file *does* contain a manifest:
    1. The manifest must contain a parameter called `mainWorkflowURL`. If it does then the value of the parameter must either be a URL, including the relevant protocol, or the name of a file present in the zip archive. Any subworkflows or tasks imported by the main workflow must either be referenced as URLs in the workflow or be present in the archive as described above.
    2. The manifest may contain an array of URLs to inputs files called `inputFileURLs`. The

WesAdapter must decide if it should resolve these or let the workflow engine resolve them.

3. The manifest may contain a URL reference to an options files name optionFileURL. The WesAdapter may choose to make use of this depending on the context of the workflow engine. It may also choose to pass this to the workflow engine or pass a modified copy to the workflow engine.
3. If the source URL points to a directory then Amazon Genomics CLI will zip the directory before uploading it. The directory must follow the same conventions stated above for zip files.

The following snippet demonstrates a possible declaration of a multi-file workflow:

```
workflows:
  gatk4-data-processing:
    type:
      langauge: wdl
      version: 1.0
      sourceURL: ./gatk4-data-processing
```

The following snippet demonstrates a valid MANIFEST.json file:

```
{
  "mainWorkflowURL": "processing-for-variant-discovery-
  "inputFileURLs": [
    "processing-for-variant-discovery-gatk4.hg38.wgs.ir
  ],
  "optionFileURL": "options.json"
}
```

MANIFEST.json Structure

The following keys are allowed in the MANIFEST.json

Key	Required	Purpose
mainWorkflowURL	Yes	Points to the workflow definition that is the main entrypoint of the workflow.
inputFileURLs	No	An array of URLs to one or more JSON files that define the inputs to the workflow in the format expected by the relevant engine. inputFile URLs are resolved relative to the location of the MANIFEST.json  If multiple files are listed in inputFiles URLs, they will be passed to Cromwell in

the order specified as workflowInput.json, workflowInput\_1.json, ... , workflowInput\_5.json. (Note: Cromwell only supports up to 5 input.json files). If there are any properties in common between the files, values in higher numbered files will take precedence. See: [Cromwell Docs](#)

optionFileURL	No	A URL pointing to a JSON file containing engine options applied to a workflow instance. This is only used when engines run in <a href="#">server mode</a> . Options are interpreted by the engine and so must be in the form expected by the engine. The URL is resolved relative to the location of the MANIFEST.json .
engineOptions	No	A string appended to the command line of the engine's run command. The string may contain any flags or parameters relevant to the engine of the context used to run the workflow. It should not be used to declare inputs (use inputFileURLS instead). This parameter is only relevant for engines that run as <a href="#">head processes</a> .

## Engine Selection

When a workflow is submitted to run, Amazon Genomics CLI will match the workflow type with the map of engines in the context. For example, if the workflow type is wdl Amazon Genomics CLI will attempt to identify an engine designated as the engine for that type. There may only be one engine per type. If no suitable engine is found in the context an error will be reported.

## Workflow Instances

Any defined project workflow can be run multiple times. Each run is called an instance and assigned a unique instance ID. When referring to a specific run of a workflow you should use the instance ID rather than the workflow name. It is possible to submit multiple instances of the same workflow and to have these run concurrently.

## Context

All workflows are coordinated by the engine, they are submitted to and executed in the context that is specified at submission

time. The workflow engine decides how the workflow is to be run and the context provides compute resources to run the workflow.

## Commands

A full reference of workflow commands is available [here](#)

### run

Invoking `agc workflow run <workflow-name> -c <context-name>` will run the named workflow in a specific context. The unique ID of that workflow instance run will be returned if the submission is successful.

### workflow arguments

Workflow arguments such as inputs file can be specified at submission time using the `i` or `--inputsFile` flag. For example:

```
agc workflow run my-workflow --inputsFile inputs.json
```

If the inputs file references local files, these will be synced with S3 and those files in S3 will be used when the workflow instance is run.

### workflow optionFileUrl

An additional `optionFileUrl` can be provided using the `'o'` or `'--optionFileUrl'` flag. For example:

```
agc workflow run my-workflow --optionFileUrl optionFile
```

`OptionFileUrl` is only for use with engines that run in server mode (e.g. Cromwell).

Example `option.json`

```
{
  "option_name_1": "option value 1",
  "option_name_2": "option value 2"
}
```

### list

The `agc workflow list` command can be used to list all workflows that are specified in the current project.

## describe

The `agc workflow describe <workflow-name>` command will return detailed information about the named workflow based on the specification in the current project YAML file.

## status

To find out the status of workflow instances that are running, or have been run you can use the `agc workflow status` command. This will display details on 20 recent workflows from the project, to display more, or fewer you can use the `--limit number` flag where the `number` may be as many as 1000.

To list the status of workflows run or running in a specific context use the `--context-name` flag and provide the name of one of the contexts of the project.

You may get the status of workflow instances by workflow name using the `--workflow-name` flag.

To display the status of a specific workflow instance you can provide the id of the desired workflow instance with the `--instance-id` flag.

## stop

A running workflow *instance* can be stopped at any time using the `agc workflow stop <instance-id>` command. When issued, Amazon Genomics CLI will look up the appropriate context and engine using the `instance-id` of the workflow and instruct the engine to stop the workflow. What happens next depends on the actual workflow engine. For example, in the case of the Cromwell WDL engine, any currently executing tasks will halt, any pending tasks will be removed from the work queue and no further tasks will be started for that workflow instance.

## output

You can obtain the output (if any) of a completed workflow run using the `output` command and supplying the workflow run id. Typically, this is useful for locating the files produced by a workflow, although the actual output generated depends on the workflow specification and engine.

If the workflow declares outputs you may also obtain these using the command:

```
agc workflow output <workflow_run_id>
```

The following is an example of output from the “CramToBam” workflow run in a context using the Cromwell engine.

```
OUTPUT id aaba95e8-7512-48c3-9a61-1fd837ff6099
OUTPUT outputs.CramToBamFlow.outputBai s3://agc-123456
OUTPUT outputs.CramToBamFlow.outputBam s3://agc-123456
OUTPUT outputs.CramToBamFlow.validation_report s3://agc-123456
```

## Cost

Your account will be charged based on actual resource usage including compute time, storage, data transfer charges etc. The resources used will depend on the resources requested in your workflow definition as interpreted by the workflow engine according the resources made available in the context in which the workflow is run. If a spot context is used then the costs of the spot instances will be determined by the rules governing spot instance charges.

## Tags

Resources used by Amazon Genomics CLI are tagged including the username, project name and the context name. Currently, tagging is *not* possible at the level of an individual workflow.

## 4.7 - Engines

Workflow engines parse and manage the tasks in a workflow

A workflow engine is defined as part of a [context](#). A context is currently limited to one workflow engine. The workflow engine will manage the execution of any [workflows](#) submitted by Amazon Genomics CLI. When the context is deployed, an endpoint will be made available to Amazon Genomics CLI through which it will submit workflows and workflow commands to the engine according to the WES API specification.

### Supported Engines and Workflow Languages

Currently, Amazon Genomics CLI's officially supported engines can be used to run the following workflows:

Engine	Language	Language Versions	Run Mode
<a href="#">Cromwell</a>	<a href="#">WDL</a>	All versions up to 1.0	Server
<a href="#">Nextflow</a>	<a href="#">Nextflow DSL</a>	Standard and DSL 2	Head Process
<a href="#">miniwdl</a>	<a href="#">WDL</a>	<a href="#">documented here</a>	Head Process
<a href="#">Snakemake</a>	<a href="#">Snakemake</a>	All versions	Head Process
<a href="#">Toil</a>	<a href="#">CWL</a>	All versions up to 1.2	Server

Overtime we plan to add additional engine and language support and provide the ability for third party developers to develop engine plugins.

### Run Mode

#### Server

In server mode the engine runs as a long-running process that exists for the lifetime of the context. All workflow instances sent to the context are handled by that server. The server resides on on-demand instances to prevent Spot interruption even if the workflow tasks are run on Spot instances

#### Head Process

Head process engines are run when a workflow is submitted, manage a single workflow and only run for the lifetime of the workflow. If multiple workflows are submitted to a context in parallel then multiple head processes are spawned. The head processes always run on on-demand resources to prevent Spot interruption even if the workflow tasks are run on Spot instances.

## Engine Definition

An engine is defined within a `context` definition of the [project YAML file](#) as a map. For example, the following snippet defines a WDL engine of type `cromwell` as part of the context named `onDemandCtx`. There may be one engine defined for each supported language.

```
contexts:
  onDemandCtx:
    requestSpotInstances: false
    engines:
      - type: wdl
        engine: cromwell
```

## Commands

There are no commands specific to engines. Engines are [deployed](#) along with contexts by the [context commands](#) and workflows are run using the [workflow commands](#).

## Costs

The costs associated with an engine depend on the actual infrastructure required by the engine. In the case of the Cromwell, the engine runs in “server” mode as an [AWS ECS Fargate](#) container using an [Amazon Elastic File System](#) for metadata storage. The container will be running for the entire time the context is deployed, even when no workflows are running. To avoid this cost we recommend destroying the context when it is not needed. The Nextflow engine runs as a single batch job per workflow instance and is only running when workflows are running.

In both cases a serverless WES API endpoint is deployed through [Amazon API Gateway](#) to act as the interface between Amazon Genomics CLI and the engine.

## Tags



Being part of a context, engine related infrastructure is [tagged](#) with the context name, username and project name. These tags may be used to help differentiate costs.

## Technical Details

Supported engines are currently deployed with configurations that allow them to make use of files in S3 and submit workflows as jobs to AWS Batch. Because the current generation of engines we support do not directly support the [WES API](#), adapters are deployed as Fargate container tasks. AWS API Gateway is used to provide a gateway between Amazon Genomics CLI and the WES adapters.

When `workflow` commands are issued on Amazon Genomics CLI, it will send WES API calls to the appropriate endpoint. The adapter mapped to that endpoint will then translate the WES command and either send the command to the engines REST API for Cromwell, or spawn a Nextflow engine task and submit the workflow with that task. At this point the engine is responsible for creating controlling and destroying the resources that will be used for task execution.

## 4.8 - Logs

Logs are produced by contexts, engines and workflow tasks. Understanding how to access them is critical to monitoring and debugging workflows.

The infrastructure deployed by Amazon Genomics CLI records logs for many activities including the workflow runs, workflow engines as well as infrastructure. The logs are recorded in CloudWatch but are accessible through the CLI.

When debugging or reviewing a workflow run, the engine logs and workflow logs will be the most useful. For diagnosing infrastructure or access problems the adapter logs and access logs will be informative.

### Engine Logs

Engine logs are the logs produced by a workflow engine in a context. The logs produced depend on the engine implementation. Engines that run in “server” mode, such as Cromwell, produce a single log for the lifetime of the context that encompass all workflows run through that engine. Engines that run as “head node” will produce discrete engine logs for each run.

### Workflow Logs

Workflow logs are the aggregate logs for all steps in a workflow run (instance). Any workflow steps that are retrieved from a call cache are not run so there will be no workflow logs for these steps. Consulting the engine logs may show details of the call cache. If a previously successful workflow is run with no changes in inputs or parameters it may have all steps retrieved from the cache in which case there will be no workflow logs although the workflow instance will be marked as a success and engine logs will be produced. The outputs for a completely cached workflow will also be available.

### Adapter Logs

Adapter logs consist of any logs produced by a WES adapter for a workflow engine. They can reveal information such as the WES API calls that are made to the engine by Amazon Genomics CLI and any errors that may have occurred.

### Access Logs

Amazon Genomics CLI talks to an engine via API Gateway which routes to the WES adapter. If an expected call does not appear in the adapter logs it may have been blocked or incorrectly routed in the API Gateway. The API Gateway access logs may be informative in this case.

## Commands

A full reference of Amazon Genomics CLI `logs` commands are available [here](#)

## Costs

Amazon Genomics CLI logs are stored in CloudWatch and accessed using the CloudWatch APIs. Standard CloudWatch charges apply. All logs are retained permanently, even after a context is destroyed and other Amazon Genomics CLI infrastructure is removed from an account. If they are no longer needed they may be removed using the AWS Console or the AWS CLI.

## 4.9 - Namespaces

Amazon Genomics CLI uses namespacing to prevent conflicts

Amazon Genomics CLI uses namespacing to prevent conflicts when there are multiple [users](#), [contexts](#), and [projects](#) in the same AWS account and region.

In any given account and region, an individual user may have many projects with many deployed contexts all running at the same time without conflict as long as:

- 1. No other user with the same Amazon Genomics CLI username exists in the same account and region.
- 2. All projects, used by that user, have a unique name.
- 3. All contexts within a project have a unique name.

## Shared Project Definitions

Project definitions can be shared between users. A simple way to achieve this is by putting the project YAML file and associated workflow definitions into a source control system like Git. If two users in the same account and region start contexts from the same project definition, these contexts are discrete and include the Amazon Genomics CLI username in the names of their respective infrastructures.

Therefore, the following combination are allowed:

```
userA -uses-> ProjectA -to-deploy-> ContextA
userB -uses-> ProjectA -to-deploy-> ContextA
```

In the above example it is useful to think of these as two instances of Context A. Both share the same definition but the instances do not have the same infrastructure.

## Tags

All Amazon Genomics CLI infrastructure is tagged with the `application-name` key and a value of `agc`. Aside from the core account infrastructure, all deployed infrastructure is tagged with the following key value pairs:

Key	Value
agc-project	The name of the project in which the context is defined
agc-user-id	The unique username

agc-user-email	The users email
agc-context	The name of the context in which the infrastructure is deployed
agc-engine	The name of the engine being run in the context
agc-engine-type	The workflow language run by the engine

## 5 - Tutorials

Work through some end to end examples.

### Attention

**The Amazon Genomics CLI project has entered its End Of Life (EOL) phase.** The code is no longer actively maintained and the **Github repository will be archived on May 31 2024**. During this time, we encourage customers to migrate to [AWS HealthOmics](#) to run their genomics workflows on AWS, or [reach out to their AWS account team](#) for alternative solutions. While the source code of AGC will still be available after the EOL date, we will not make any updates inclusive of addressing issues or accepting Pull Requests.

The following are **complete worked examples** made up of **multiple tasks** that guide you through a relatively simple but realistic scenarios using Amazon Genomics CLI.

## 5.1 - Walk through

Demonstrates installation and using the essential functions of Amazon Genomics CLI

### Prerequisites

Ensure you have completed the [prerequisites](#) before beginning.

### Download and install Amazon Genomics CLI

Download the Amazon Genomics CLI according to the [installation](#) instructions.

### Setup

Ensure you have initialized your account and created a username by following the [setup](#) instructions.

### 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 for running WDL based workflows do the following:

```
mkdir myproject
cd myproject
agc project init myproject --workflow-type wdl
```

NOTE: for a Nextflow based project you can substitute `--workflow-type wdl` with `---workflow-type nextflow`.

Projects may have workflows from different languages, so the `--workflow-type` flag is simply to provide the stub for an initial workflow engine.

This will create a config file called `agc-project.yaml` with the following contents:

```
name: myproject
schemaVersion: 1
contexts:
  ctx1:
    engines:
      - type: wdl
        engine: cromwell
```

This config file will be used to define aspects of the project - e.g. the contexts and named workflows the project uses. For a more representative project config, look at the projects in `~/agc/examples`. Unless otherwise stated, command line activities for the remainder of this document will assume they are run from within the `~/agc/examples/demo-wdl-project/` 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. In the `demo-wdl-project` folder, after running the following:

```
agc context list
```

You should see something like:

```
2021-09-22T01:15:41Z i Listing contexts.
CONTEXTNAME    cromwell    myContext
CONTEXTNAME    cromwell    spotCtx
```

In this project there are two contexts, one configured to run with On-Demand instances (`myContext`), and one configured to use SPOT instances (`spotCtx`).

You need to have a context running to be able to run workflows. To deploy the context `myContext` in the `demo-wdl-project`, run:

```
agc context deploy myContext
```

This will take 10-15min to complete.

If you have more than one context configured, and want to deploy them all at once, you can run:



```
agc context deploy --all
```

Contexts have read-write access to a context specific prefix in the S3 bucket Amazon Genomics CLI creates during account activation. You can check this for the `myContext` context with:

```
agc context describe myContext
```

You should see something like:

```
CONTEXT      myContext      false      STARTED
OUTPUTLOCATION s3://agc-123456789012-us-east-2/proje
WESENDPOINT   https://a1b2c3d4.execute-api.us-east-
```

You can add more data locations using the `data` section of the `agc-project.yaml` config file. All contexts will have an appropriate access policy created for the data locations listed when they are deployed. For example, the following config adds three public buckets from the Registry of Open Data on AWS:

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

Note, you need to redeploy any running contexts to update their access to data locations. Do this by simply (re)running.

```
agc context deploy myContext
```

Contexts also define what types of compute your workflow will run on - i.e. if you want to run workflows using SPOT or On-demand instances. By default, contexts use On-demand instances. The configuration for a context that uses SPOT instances looks like the following:

```
contexts:
  # The spot context uses EC2 spot instances which are
  spotCtx:
    requestSpotInstances: true
```

You can also explicitly specify what instance types contexts will be able to use for workflow jobs. By default, Amazon Genomics CLI will use a select set of instance types optimized for running genomics workflow jobs that balance data I/O performance and mitigation of workflow failure due to SPOT reclamation. In short, Amazon Genomics CLI uses AWS Batch for job execution and selects instance types based on the requirements of submitted jobs, up to `4xlarge` instance types. If you have a use case that requires a specific set of instance types, you can define them with something like:

```
contexts:
  specialCtx:
    instanceTypes:
      - c5
      - m5
      - r5
```

The above will create a context called `specialCtx` that will use any size of instances in the C5, M5, and R5 instance families. Contexts are elastic with a minimum vCPU capacity of 0 and a maximum of 256. When all vCPUs are allocated to jobs, further tasks will be queued.

Contexts also launch an engine for specific workflow types. You can have one engine per context and, currently, engines for WDL and Nextflow are supported.

A contexts configured with WDL and Nextflow engines respectively look like:

```
contexts:
  wdlContext:
    engines:
      - type: wdl
        engine: cromwell

  nfContext:
    engines:
      - type: nextflow
        engine: nextflow
```

## Workflows

### Add a workflow

Bioinformatics workflows are written in languages like WDL and Nextflow in either single script files, or in packages of multiple files (e.g. when there are multiple related workflows that leverage reusable elements). Currently, Amazon Genomics CLI supports both WDL and Nextflow. To learn more about WDL

workflows, we suggest resources like the [OpenWDL - Learn WDL course](#). To learn more about Nextflow workflows, we suggest [Nextflow's documentation](#) and [NF-Core](#).

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, which Amazon Genomics CLI will use to execute it on appropriate compute resources.

There is a "hello" workflow definition in the `~/agc/examples/demo-wdl-project/workflows/hello` folder that looks like:

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

The workflow specification for this workflow in the project config looks like:

```
workflows:
  hello:
    type:
      language: wdl
      version: 1.0
      sourceURL: workflows/hello.wdl
```

Here the workflow is expected to conform to the `WDL-1.0` specification. A specification for a "hello" workflow written in Nextflow DSL1 would look like:

```
workflows:
  hello:
    type:
      language: nextflow
      version: 1.0
      sourceURL: workflows/hello
```

For Nextflow DSL2 workflows set `type.version` to `dsl2`.

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

You can quickly get a list of available configured workflows with:

```
agc workflow list
```

For the `demo-wdl-project`, this should return something like:

```
2021-09-02T05:14:47Z i Listing workflows.
WORKFLOWNAME    hello
WORKFLOWNAME    read
WORKFLOWNAME    words-with-vowels
```

The `hello` 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 would simply point to the parent directory:

```
workflows:
  hello-dir-abs:
    type:
      language: wdl
      version: 1.0
    sourceURL: /abspath/to/hello-dir
  hello-dir-rel:
    type:
      language: wdl
      version: 1.0
    sourceURL: relpath/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:
      language: wdl
      version: 1.0 # this is the WDL spec version
      sourceURL: https://raw.githubusercontent.com/openw...
```

NOTE: remote sourceURLs for Nextflow workflows can be GitHub repo URLs like: <https://github.com/nextflow-io/rnaseq-nf.git>

## 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 “myContext” context, run:

```
agc workflow run hello --context myContext
```

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
"06604478-0897-462a-9ad1-47dd5c5717ca"
```

The last line is the workflow run 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    myContext    66826672-778e-449d-8f2
```

By default, the `workflow status` command will show the state of all workflows across all running contexts.

To show only the status of workflow instances of a specific workflow you can use:

```
agc workflow status -n <workflow-name>
```

To show only the status of workflows instances in a specific context you can use:

```
agc workflow status -c <context-name>
```

If you want to check the status of a specific workflow you can do so by referencing the workflow execution by its run id:

```
agc workflow status -r <workflow-run-id>
```

If you need to stop a running workflow instance, run:

```
agc workflow stop <workflow-run-id>
```

## Using workflow inputs

You can provide runtime inputs to workflows at the command line. For example, the `demo-wdl-project` has a workflow named `read` that requires reading a data file. The specification of `read` 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 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": "data.txt"}
EOF

```

Finally, you would submit the workflow with its corresponding inputs file with:

```

agc workflow run read --inputsFile inputs/read.inputs.json

```

Amazon Genomics CLI will scan the file provided to `--inputsFile` for local paths, sync those files to S3, and rewrite the inputs file in transit to point to the appropriate S3 locations. Paths in the `*.inputs.json` file provided as `--inputsFile` are referenced relative to the `*.inputs.json` file.

## 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 run id, and task name. The `workflow` prefix is where named workflows are cached when you run workflows definitions stored in your local environment.

If a workflow declares workflow outputs then these can be obtained using `agc workflow output <run_id>`

The following is example output from the “cram-to-bam” workflow

```
OUTPUT id      aaba95e8-7512-48c3-9a61-1fd837ff6099
OUTPUT outputs.CramToBamFlow.outputBai s3://agc-123456
OUTPUT outputs.CramToBamFlow.outputBam s3://agc-123456
OUTPUT outputs.CramToBamFlow.validation_report s3://agc-123456
```

## Accessing workflow logs

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

```
agc logs workflow <workflow-name>
```



This will return the logs for all runs of the workflow. If you just want the logs for a specific workflow run, you can use:

```
agc logs workflow <workflow-name> -r <workflow-instance>
```

This will print out the `stdout` generated by each workflow task.

For the hello workflow above, this would look like:

```
Fri, 10 Sep 2021 22:00:04 +0000    download: s3://agc-1
Fri, 10 Sep 2021 22:00:04 +0000    *** LOCALIZING INPUT
Fri, 10 Sep 2021 22:00:05 +0000    download: s3://agc-1
Fri, 10 Sep 2021 22:00:05 +0000    *** COMPLETED LOCALI
Fri, 10 Sep 2021 22:00:05 +0000    Hello Amazon Genomic
Fri, 10 Sep 2021 22:00:05 +0000    *** DELOCALIZING OUT
Fri, 10 Sep 2021 22:00:05 +0000    upload: ./hello-rc.t
Fri, 10 Sep 2021 22:00:06 +0000    upload: ./hello-stdc
Fri, 10 Sep 2021 22:00:06 +0000    upload: ./hello-stdc
Fri, 10 Sep 2021 22:00:06 +0000    *** COMPLETED DELOCA
```

If your workflow fails, useful debug information is typically reported by the workflow engine logs. These are unique per context. To get those for a context named `myContext`, you would run:

```
agc logs engine --context myContext
```

You should get something like:

```
Fri, 10 Sep 2021 23:40:49 +0000    2021-09-10 23:40:49,
Fri, 10 Sep 2021 23:40:52 +0000    2021-09-10 23:40:52,
Fri, 10 Sep 2021 23:40:52 +0000    2021-09-10 23:40:52,
Fri, 10 Sep 2021 23:40:52 +0000    2021-09-10 23:40:52,
Fri, 10 Sep 2021 23:40:52 +0000    2021-09-10 23:40:52,
Fri, 10 Sep 2021 23:40:52 +0000    2021-09-10 23:40:52,
Fri, 10 Sep 2021 23:40:52 +0000    2021-09-10 23:40:52,
Fri, 10 Sep 2021 23:40:52 +0000    2021-09-10 23:40:52,
Fri, 10 Sep 2021 23:40:53 +0000    2021-09-10 23:40:53,
Fri, 10 Sep 2021 23:40:54 +0000    2021-09-10 23:40:54,
Fri, 10 Sep 2021 23:40:55 +0000    2021-09-10 23:40:55,
Fri, 10 Sep 2021 23:40:56 +0000    2021-09-10 23:40:56,
Fri, 10 Sep 2021 23:40:57 +0000    2021-09-10 23:40:57,
Fri, 10 Sep 2021 23:40:57 +0000    {
Fri, 10 Sep 2021 23:40:57 +0000    "hello_agc.hello.c
Fri, 10 Sep 2021 23:40:57 +0000    }
Fri, 10 Sep 2021 23:40:59 +0000    2021-09-10 23:40:59,
```

You can filter logs with the `--filter` flag. The filter syntax adheres to [CloudWatch's filter and pattern syntax](#). For example, the following will give you all error logs from the workflow engine:

```
agc logs engine --context myContext --filter ERROR
```

## Additional workflow examples

The Amazon Genomics CLI installation also includes a set of typical genomics workflows for raw data processing, germline variant discovery, and joint genotyping based on [GATK Best Practices](#), developed by the [Broad Institute](#). More information on how these workflows work is available in the [GATK Workflows Github repository](#).

You can find these in:

```
~/agc/examples/gatk-best-practices-project
```

These workflows come pre-packaged with `MANIFEST.json` files that specify example input data available publicly in the [AWS Registry of Open Data](#).

Note: these workflows take between 5 min to ~3hrs to complete.

## Cleanup

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

Stop a context with:

```
agc context destroy <context-name>
```

This will destroy all compute resources in a context, but retain any data in S3. If you want to destroy all your running contexts at once, you can use:

```
agc context destroy --all
```

Note, you will not be able to destroy a context that has a running workflow. Workflows will need to complete on their own or stopped before you can destroy the context.

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.

# 6 - Best Practices

Best practices when using Amazon Genomics CLI

Things to consider so that you can get the most out of Amazon Genomics CLI

## 6.1 - IAM Permissions

Minimum IAM Permissions required to use AGC

Amazon Genomics CLI is used to deploy and interact with infrastructure in an AWS account. Amazon Genomics CLI will use the permissions of the current profile to perform its actions. The profile would either be the users profile or, if being run from an EC2 instance, the attached profile of the instance. No matter the source of the role it must have sufficient permissions to perform the necessary tasks. In addition, best practice recommends that the profile only grant minimal permissions to maintain security and prevent unintended action.

### Recommended Minimal Permissions

As part of the Amazon Genomics CLI repository we have included a CDK project that can be used by an account administrator to generate the necessary minimum policies.

### Pre-requisites

Before generating the policies you need to do the following:

1. Install `node` and `npm` . We recommend using node v14.17 installed via `nvm`
2. Install Amazon CDK ( `npm install -g aws-cdk@latest` )
3. An AWS account where you will use Amazon Genomics CLI
4. A role in that account that allows the creation of IAM roles and policies

### Generate Roles and Policies

1. Clone the Amazon Genomics CLI repository locally: `git clone https://github.com/aws/amazon-genomics-cli.git`
2. `cd amazon-genomics-cli/extras/agc-minimal-permissions/`
3. `npm install`
4. `cdk deploy`

You will see output similar to the following:

```
✨ Synthesis time: 2.91s

AgcPermissionsStack: deploying...
AgcPermissionsStack: creating CloudFormation changeset.

✔ AgcPermissionsStack

✨ Deployment time: 44.39s

Stack ARN:
arn:aws:cloudformation:us-east-1:123456789123:stack/Agc

✨ Total time: 47.3s
```

Using the emitted Stack ARN you can identify the policies created. You can also inspect the stack in the CloudFormation console.

For example:

```
aws cloudformation describe-stack-resources --stack-name
```

with output similar to:

```
{
  "StackResources": [
    {
      "StackName": "AgcPermissionsStack",
      "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/AgcPermissionsStack/12345678-9012-3456-7890-123456789012",
      "LogicalResourceId": "CDKMetadata",
      "PhysicalResourceId": "6ace55f0-b67c-11ec-8000-0242ac120000",
      "ResourceType": "AWS::CDK::Metadata",
      "Timestamp": "2022-04-07T14:10:30.922000+00:00",
      "ResourceStatus": "CREATE_COMPLETE",
      "DriftInformation": {
        "StackResourceDriftStatus": "NOT_CHECKED"
      }
    },
    {
      "StackName": "AgcPermissionsStack",
      "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/AgcPermissionsStack/12345678-9012-3456-7890-123456789012",
      "LogicalResourceId": "agcadminpolicy2500318",
      "PhysicalResourceId": "arn:aws:iam::123456789012:policy/agcadminpolicy2500318",
      "ResourceType": "AWS::IAM::ManagedPolicy",
      "Timestamp": "2022-04-07T14:10:41.597000+00:00",
      "ResourceStatus": "CREATE_COMPLETE",
      "DriftInformation": {
        "StackResourceDriftStatus": "NOT_CHECKED"
      }
    },
    {
      "StackName": "AgcPermissionsStack",
      "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/AgcPermissionsStack/12345678-9012-3456-7890-123456789012",
      "LogicalResourceId": "agcuserpolicy346A2D4",
      "PhysicalResourceId": "arn:aws:iam::123456789012:policy/agcuserpolicy346A2D4",
      "ResourceType": "AWS::IAM::ManagedPolicy",
      "Timestamp": "2022-04-07T14:10:41.981000+00:00",
      "ResourceStatus": "CREATE_COMPLETE",
      "DriftInformation": {
        "StackResourceDriftStatus": "NOT_CHECKED"
      }
    },
    {
      "StackName": "AgcPermissionsStack",
      "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/AgcPermissionsStack/12345678-9012-3456-7890-123456789012",
      "LogicalResourceId": "agcuserpolicycdk27FA",
      "PhysicalResourceId": "arn:aws:iam::123456789012:policy/agcuserpolicycdk27FA",
      "ResourceType": "AWS::IAM::ManagedPolicy",
      "Timestamp": "2022-04-07T14:10:41.747000+00:00",
      "ResourceStatus": "CREATE_COMPLETE",
      "DriftInformation": {
        "StackResourceDriftStatus": "NOT_CHECKED"
      }
    }
  ]
}
```

Three resources of type `AWS::IAM::ManagedPolicy` are created:

- The resource with a name similar to `agcadminpolicy25003180` identify policies which grant sufficient permission to run `agc account activate` and

`agc account deactivate` and should be attached to the profile of users who need to perform that action

- Two resources with names similar to `agcuserpolicy346A2D4F` and `agcuserpolicycdk27FA61BC` identify policies which allow all other Amazon Genomics CLI actions. These should be attached to profiles that will use Amazon Genomics CLI day to day.

## 6.2 - Controlling Costs

Monitoring costs and design considerations to reduce costs

When you begin to run large scale workflows frequently it will become important to be able to understand the costs involved and how to optimize your workflow and use of Amazon Genomics CLI to reduce costs.

### Use AWS Cost Explorer to Report on Costs

AWS Cost Explorer has an easy-to-use interface that lets you visualize, understand, and manage your AWS costs and usage over time. We recommend you use this tool to gain sight into the costs of running your genomics workflows. At the time of writing AWS Cost Explorer can only be enabled using the AWS Console so Amazon Genomics CLI won't be able to set this up for you. As a first step you will need to [enable cost explorer](#) for your AWS account.

Amazon Genomics CLI will tag the infrastructure it creates with tags. Application, user, project and context tags are all generated as appropriate and these can be used as [cost allocation tags](#) to determine which account costs are coming from Amazon Genomics CLI and which user, context and project.

Within Cost Explorer the Amazon Genomics CLI tags will be referred to as ["User Defined Cost Allocation Tags"](#). Before a tag can be used in a cost report it must be [activated](#). Costs associated with tags are only available for infrastructure used *after* activation of a tag, so it will not be possible to retrospectively examine costs.

### Optimizing Requested Container Resources

Tasks in a workflow typically run in Docker containers. Depending on the workflow language there will be some kind of `runtime` definition that specifies the number of vCPUs and amount of RAM allocated to the task. For example, in WDL you could specify



```
runtime {  
  docker: "biocontainers/plink1.9:v1.90b6.6-181012-1-  
  memory: "8 GB"  
  cpu: 2  
}
```

The amount of resource allocated to each container ultimately impacts the cost to run a workflow. Optimally allocating resources leads to cost efficiency.

## Optimize the longest running, and most parallel tasks first

When optimizing a workflow, focus on those tasks that run the longest as well as those that have the largest number of parallel tasks as they will make up the majority of the workflow runtime and contribute most to the cost.

## Consider CPU and memory ratios

EC2 workers for Cromwell AWS Batch compute environments are `c`, `m`, and `r` instance families that have vCPU to memory ratios of 1:2, 1:4 and 1:8 respectively. Engines that run container based workflows will typically attempt to fit containers to instances in the most optimal way depending on cost and size requirements, or they will delegate this to a service like AWS Batch. Given that a task requiring 16 GB of RAM that could make use of all available CPUs, then to optimally pack the containers you should specify either 2, 4, or 8 vCPU. Other values could lead to inefficient packing meaning the resources of the EC2 container instance will be paid for but not optimally used.

NOTE: Fully packing an instance can result in it becoming unresponsive if the tasks in the containers use 100% (or more if they start swapping) of the allocated resources. The instance may then be unresponsive to its management services or the workflow engine and may time out. To avoid this, always allow for a little overhead, especially in the smaller instances.

The largest instance types deployed by default are from the `4xlarge` size which have 16 vCPU and up to 128 GB of RAM.

## Consider splitting tasks that pipe output

If a workflow task consists of a process that pipes `STDOUT` to another process then both processes will run in the same container and receive the same resources. If one task requires more resources than the other this might be inefficient, it may be better divided into two tasks each with its own `runtime` configuration. Note that this will require the intermediate `STDOUT` to be written to a file and copied between containers so if this output is very large then keeping the processes in the same task may be more efficient. Piping very large outputs may a lot of memory so your container will need an appropriate allocation of memory.

## Use the most cost-effective instance generation

When you specify the `instanceTypes` in a context, as opposed to letting Amazon Genomics CLI do it for you, consider the cost and performance of the instance types with respect to your workflow requirements. Fifth generation EC2 types ( `c5` , `m5` , `r5` ) have a lower on-demand price and have higher clock speeds than their 4th generation counterparts ( `c4` , `m4` , `r4` ). Therefore, for on-demand compute environments, those instance types should be preferred. In spot compute environments we suggest using both 4th and 5th generation types as this increases the pool of available types meaning Batch will be able to choose the instance type that is cheapest and least likely to be interrupted.

## Deploy Amazon Genomics CLI where your S3 data is

Genomics workflows may need to access considerable amounts of data stored in S3. Although S3 uses global namespaces, buckets do reside in regions. If you access a lot of S3 data it makes sense to deploy your Amazon Genomics CLI infrastructure in the same region to avoid cross region data charges.

Further, if you use a custom VPC we recommend deploying a VPC endpoint for S3 so that you do not incur NAT Gateway charges for data coming from the same region. If you do not you might find that NAT Gateway charges are the largest part of your workflow run costs. If you allow Amazon Genomics CLI to create your VPC (the default), appropriate VPC endpoints will be setup for you. Note that VPC endpoints cannot avoid cross region data charges, so you will still want to deploy in the region where most of your data resides.

## Use Spot Instances

The use of Spot instances can significantly reduce costs of running workflows. However, spot instances may be interrupted when EC2 demand is high. Some workflow engines, such as Cromwell, can support retries of tasks that fail due to Spot interruption (among other things). To enable this for Cromwell, include the `awsBatchRetryAttempts` parameter in the `runtime` section of a WDL task with an integer number of attempts.

Even with retries, there is a risk that spot interruption will cause a task or entire workflow to fail. Use of an engine's caching capabilities (if available) can help avoid repeating work if a partially complete workflow needs to be restarted due to Spot instance interruption.

## Use private ECR registries

Each task in a workflow requires access to a container image, and some of these images can be several GB if they contain large packages like GATK. This can lead to large NAT Gateway traffic charges. To avoid these charges, we recommend deploying copies of frequently used container images into your account's private ECR registry.

Amazon Genomics CLI deployed VPCs use a VPC gateway to talk to private ECR registries in your account thereby avoiding NAT Gateway traffic. The gateway is limited to registries in the same region as the VPC, so to avoid cross-region traffic you should deploy images into the region(s) that you use for Amazon Genomics CLI.

## 6.3 - Scaling Workloads

Making workflows run at scale

Workflows with considerable compute requirements can incur large costs and may fail due to infrastructure constraints. The following considerations will help you design workflows that will perform better at scale.

### Large compute requirements

By default, contexts created by AGC will allocate compute nodes with a size of up to `4xlarge`. These types have 16 vCPU and up to 128 GB of RAM. If an individual task requires additional resources you may specify these in the `instanceTypes` array of the project context. For example:

```
contexts:
  prod:
    requestSpotInstances: false
    instanceTypes:
      - c5.16xlarge
      - r5.24xlarge
```

### Large data growth

When using the Nextflow or Cromwell engines the EC2 container instances that carry out the work use a script to detect and automatically expand disk capacity. Generally, this will allow disk space to increase to the amount required to hold inputs, scratch and outputs. However, it can take up to a minute to attach new storage so events that fill disk space in under a minute can result in failure.

### Large numbers of inputs/outputs

Typically, genomics files are large and best stored in S3. However, most applications used in genomics workflows cannot read directly from S3. Therefore, these inputs must be localized from S3. Compute work will not be able to begin until localization is complete so “divide and conquer” strategies are useful in these cases.

Whenever possible compress inputs (and outputs) appropriately. The CPU overhead of compression will be low compared to the network overhead of localization and delocalization.

Localization of large numbers of large files from S3 will put load on the network interface of the worker nodes and the node may experience transient network failures or S3 throttling. While we have included retry-with-backoff logic for localization it is not impossible that downloads may occasionally fail. Failures (and retries) will be recorded in the workflow task logs.

## Parallel Steps

Workflows often contain parallel steps where many individual tasks are computed in parallel. Amazon Genomics CLI makes use of elastic compute clusters to scale to these requirements. Each context will deploy an elastic compute cluster with a minimum of 0 vCPU and a maximum of 256 vCPU. No individual task may use more than 256 vCPU. Smaller tasks may be run in parallel up to the maximum of 256 vCPU. Once that limit is met, additional tasks will be queued to run when capacity becomes free.

Each parallel task is isolated meaning each task will need a local copy of its inputs. When large numbers of parallel tasks, require the same inputs (for example reference genomes) you may observe contention for network resources and transient S3 failures. While we have included retry with backoff logic we recommend keeping the number of parallel tasks requiring the same inputs below 500. Fewer, if the tasks inputs are large.

An extreme example is Joint Genotyping. This type of analysis benefits from processing large numbers of samples at the same. Further, the user may wish to genotype many intervals concurrently. Finally, the step of merging the variant calls will import the variants from all intervals. In our experience, a naive implementation calling 100 samples over 100 intervals is feasible. Also, feasible is calling ~20 samples over 500 intervals. At larger scales it would be worth considering dividing tasks by chromosome or batching inputs.

## Container throttling

Some container registries will throttle container access from anonymous accounts. Because each task in a workflow uses a container large or frequently run workflows may not be able to access their required containers. While compute clusters deployed by Amazon Genomics CLI are configured to cache containers this is only available on a per-instance basis. Further, due to the elastic nature of the clusters instances with cached container images are frequently shutdown. All of this will potentially lead to an excess of requests. To avoid this we recommend using registries that don't impose these limits, or using images hosted in an ECR registry in your AWS account.

# 7 - Workflow Engines

## Supported Workflow Engines

### Attention

**The Amazon Genomics CLI project has entered its End Of Life (EOL) phase.** The code is no longer actively maintained and the **Github repository will be archived on May 31 2024**. During this time, we encourage customers to migrate to [AWS HealthOmics](#) to run their genomics workflows on AWS, or [reach out to their AWS account team](#) for alternative solutions. While the source code of AGC will still be available after the EOL date, we will not make any updates inclusive of addressing issues or accepting Pull Requests.

The following pages provide details on the workflow engines that are currently supported by Amazon Genomics CLI.

# 7.1 - Filesystems

## Workflow Filesystems

### Attention

**The Amazon Genomics CLI project has entered its End Of Life (EOL) phase.** The code is no longer actively maintained and the **Github repository will be archived on May 31 2024**. During this time, we encourage customers to migrate to [AWS HealthOmics](#) to run their genomics workflows on AWS, or [reach out to their AWS account team](#) for alternative solutions. While the source code of AGC will still be available after the EOL date, we will not make any updates inclusive of addressing issues or accepting Pull Requests.

The tasks in a workflow require a common filesystem or scratch space where the outputs of tasks can be written so they are available to the inputs of dependent tasks in the same workflow. The following pages provide details on the engine filesystems that can be deployed by Amazon Genomics CLI.

## 7.1.1 - EFS Workflow Filesystem

### Amazon EFS Workflow Filesystem

Workflow engines that support it may use Amazon EFS as a shared “scratch” space for hosting workflow intermediates and outputs. Initial inputs are localized once from S3 and final outputs are written back to S3 when the workflow is complete. All intermediate I/O is performed against the EFS filesystem.

#### Advantages

1. Compared with the [S3 Filesystem](#) there is no redundant I/O of inputs from S3.
2. Each tasks individual I/O operations tend to be smaller than the copy from S3 so there is less network congestion on the container host.
3. Option to use provisioned IOPs to provide high sustained throughput.
4. The volume is elastic and will expand and contract as needed.
5. It is simple to start an Amazon EC2 instance from the AWS console and connect it to the EFS volume to view outputs as they are created. This can be useful for debugging a workflow.

#### Disadvantages

1. Amazon EFS volumes are more expensive than storing intermediates and output in S3, especially when the volume uses provisioned throughput.
2. The volume exists for the lifetime of the context and will incur costs based on its size for the lifetime of the context. If you no longer need the context we recommend destroying it.
3. Call caching is only possible for as long as the volume exists, i.e. the lifetime of the context.

#### Provisioned Throughput

Amazon EFS volumes deployed by the Amazon Genomics CLI use “[bursting](#)” throughput by default. For workflows that have high I/O throughput or in scenarios where you may have many workflows running in the same context at the same time, you may exhaust the burst credits of the volume. This might cause a



workflow to slow down or even fail. Available volume credits can be [monitored](#) in the Amazon EFS console, and/ or Amazon CloudWatch.

If you observe the exhaustion of burst credits you may want to consider deploying a context with [provisioned throughput](#). Throughput is provisioned in MiB/s and may be upto 1024 MiB/s. Note that this is an **additional expense** for the EFS volume and is charged even when the volume has no data stored in it. If you choose an EFS volume with provisioned throughput we encourage you to destroy the context whenever it is not in use to minimize costs. To determine the costs of provisioned throughput you may use the [AWS Price Calculator for EFS](#)

The following fragment of an `agc-project.yaml` file is an example of how to configure provisioned throughput for the Amazon EFS volume used by `miniwdl` in an Amazon Genomics CLI context.

```
myContext:
  engines:
    - type: wdl
      engine: miniwdl
      filesystem:
        fsType: EFS
        configuration:
          provisionedThroughput: 100
```

## Supporting Engines

The use of Amazon EFS as a shared file system is supported by the [miniwdl](#) and [Snakemake](#) engines. Both use EFS with bursting throughput by default and both support provisioned throughput.

## 7.1.2 - S3 Workflow Filesystem

### Amazon S3 Workflow Filesystem

Some workflow engines deployed by Amazon Genomics CLI can use S3 as their shared “filesystem”. Because S3 is not a POSIX compliant filesystem and most of the applications run by workflow tasks will require POSIX files, inputs will be localized from Amazon S3 and outputs will be delocalized to Amazon S3.

#### Advantages

1. Inputs are read into each task’s container and are not available by a common container mount so there is no possibility of containers on the same host over-writing or accessing another tasks inputs
2. No shared file system needs to be provisioned for a contexts compute environment thereby reducing ongoing costs.
3. All intermediate task outputs and all workflow outputs are persisted to the S3 bucket provisioned by Amazon Genomics CLI and this bucket will remain after contexts are destroyed and even after Amazon Genomics CLI is deactivated in the account.
4. Container hosts use an auto-expansion strategy for their local EBS volumes so disk sizes don’t need to be stated.

#### Disadvantages

1. Container hosts running multiple tasks may exhaust their aggregate network bandwidth (see below).
2. It is assumed that no other external process will be making changes to the S3 objects during a workflow run. If this does happen, the run may fail or be corrupted.

#### Network Bandwidth Considerations

During workflows with large numbers of concurrent steps that all rely on large inputs you may observe that the localization of inputs to the containers will become very slow. This is because a single EC2 container host may have multiple containers all competing for limited bandwidth. In these cases we recommend the following possible mitigations:

1. Consider using a shared filesystem such as EFS for your engine or an engine that supports EFS
2. Configure your `agc-project.yaml` such that a context is

available that uses instance types that are [network optimized](#). For example used `m5n` instance types rather than `m5` and use instance types that offer sustained throughput rather than [bursting throughput](#) such as instances with more than 16 vCPU.

3. Consider modifying your workflow to request larger memory and vCPU amounts for these tasks. This will tend to ensure AWS Batch selects larger instances with better performance as well as placing fewer containers per host resulting in less competition for bandwidth.

These mitigations may result in the use of more expensive infrastructure but can ultimately save money by completing the workflow quicker. The best price-performance configuration will vary by workflow.

## Supporting Engines

The [Cromwell](#) and [Nextflow](#) engines both support the use of Amazon S3 as a filesystem. Contexts using these engines will use this filesystem by default.

## 7.2 - miniwdl

Details on the miniwdl engine deployed by Amazon Genomics CLI

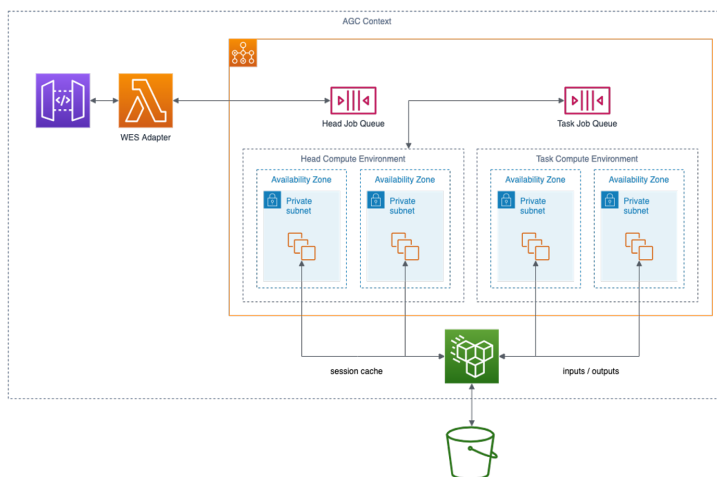
### Description

[miniwdl](#) is free open source software distributed under the MIT licence developed by the [Chan Zuckerberg Initiative](#).

The source code for miniwdl is available on [GitHub](#). When deployed with Amazon Genomics CLI miniwdl makes use of the [miniwdl-aws extension](#) which is also distributed under the MIT licence.

### Architecture

There are four components of a miniwdl engine as deployed in an Amazon Genomics CLI context:



### WES Adapter

Amazon Genomics CLI communicates with the miniwdl engine via a GA4GH [WES](#) REST service. The WES Adapter implements the WES standard and translates WES calls into calls to the miniwdl head process.

### Head Compute Environment

For every workflow submitted, the WES adapter will create a new AWS Batch Job that contains the miniwdl process responsible for running that workflow. These miniwdl “head” jobs are run in an “On-demand” AWS Fargate compute environment even when the actual workflow tasks run in a Spot environment. This is to prevent Spot interruptions from terminating the workflow coordinator.

## Task Compute Environment

Workflow tasks are submitted by the `miniwdl` head job to an AWS Batch queue and run in containers using an AWS Compute Environment. Container characteristics are defined by the resources requested in the workflow configuration. AWS Batch coordinates the elastic provisioning of EC2 instances (container hosts) based on the available work in the queue. Batch will place containers on container hosts as space allows.

## Session Cache and Input Localization

Any context with a `miniwdl` engine will use an Amazon Elastic File System (EFS) volume as scratch space. Inputs from S3 are localized to the volume by jobs that the `miniwdl` engine spawns to copy these files to the volume. Outputs are copied back to S3 using a similar process. Workflow tasks access the EFS volume to obtain inputs and write intermediates and outputs.

The EFS volume is used by all `miniwdl` engine “head” jobs to store metadata necessary for call caching.

The EFS volume will remain in your account for the lifetime of the context and are destroyed when contexts are destroyed. Because the volume will grow in size as you run more workflows we recommend destroying the context when done to avoid on going EFS charges.

## Using `miniwdl` as a Context Engine

You may declare `miniwdl` to be the `engine` for any contexts `wdl` type engine. For example:

```
contexts:
  onDemandCtx:
    requestSpotInstances: false
    engines:
      - type: wdl
        engine: miniwdl
```

## Call Caching

Call caching is enabled by default for `miniwdl` and because the metadata is stored in the contexts EFS volume call caching will work across different engine “head” jobs.

To disable call caching you can provide the `--no-cache` engine option. You may do this in a workflows `MANIFEST.json` by adding the following key/ value pair.

```
"engineOptions": "--no-cache"
```

## 7.3 - Toil

Details on the Toil engine (CWL mode) deployed by Amazon Genomics CLI

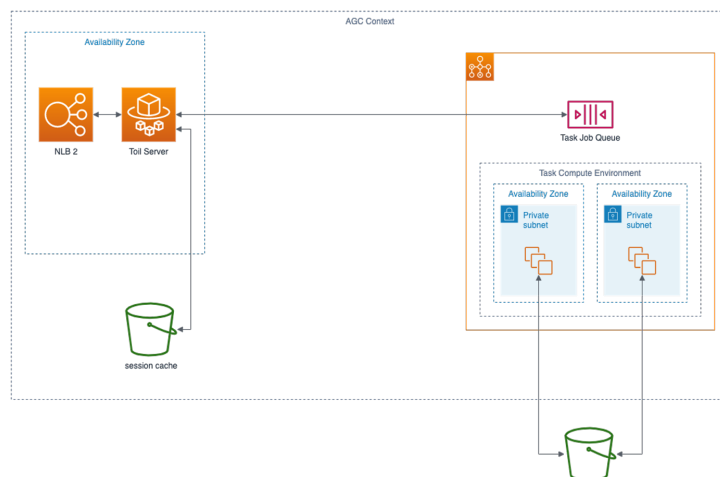
### Description

[Toil](#) is a workflow engine developed by the [Computational Genomics Lab](#) at the [UC Santa Cruz Genomics Institute](#). In Amazon Genomics CLI, Toil is an engine that can be deployed in a [context](#) as an [engine](#) to run workflows written in the [Common Workflow Language](#) (CWL) standard, version [v1.0](#), [v1.1](#), and [v1.2](#) (or mixed versions).

Toil is an open source project distributed by UC Santa Cruz under the [Apache 2 license](#) and available on [GitHub](#).

### Architecture

There are two components of a Toil engine as deployed in an Amazon Genomics CLI context:



### Toil Server

The Toil engine is run in “server mode” as a container service in ECS. The engine can run multiple workflows asynchronously. Workflow tasks are run in an elastic [compute environment](#) and monitored by Toil. Amazon Genomics CLI communicates with the Toil engine via a GA4GH [WES](#) REST service which the server offers, available via API Gateway.

### Task Compute Environment

Workflow tasks are submitted by Toil to an AWS Batch queue and run in Toil-provided containers using an AWS Compute Environment. Tasks which use the [CWL DockerRequirement](#) will

additionally be run in sibling containers on the host Docker daemon. AWS Batch coordinates the elastic provisioning of EC2 instances (container hosts) based on the available work in the queue. Batch will place containers on container hosts as space allows.

## Disk Expansion

Container hosts in the Batch compute environment use EBS volumes as local scratch space. As an EBS volume approaches a capacity threshold, new EBS volumes will be attached and merged into the file system. These volumes are destroyed when AWS Batch terminates the container host. CWL disk space requirements are ignored by Toil when running against AWS Batch.

This setup means that workflows that succeed on AGC may fail on other CWL runners (because they do not request enough disk space) and workflows that succeed on other CWL runners may fail on AGC (because they allocate disk space faster than the expansion process can react).



## 7.4 - Cromwell

Details on the Cromwell engine deployed by Amazon Genomics CLI

### Description

[Cromwell](#) is a workflow engine developed by the [Broad Institute](#). In Amazon Genomics CLI, Cromwell is an engine that can be deployed in a [context](#) as an [engine](#) to run workflows based on the [WDL](#) specification.

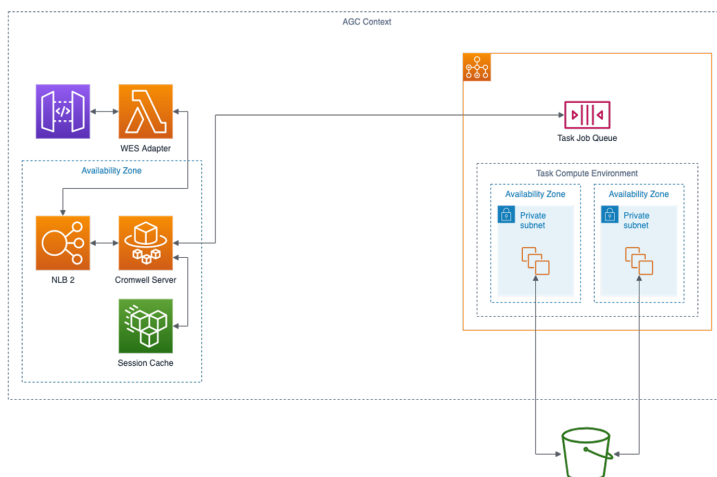
Cromwell is an open source project distributed by the Broad Institute under the [Apache 2 license](#) and available on [GitHub](#).

### Customizations

Some minor customizations were made to the AWS Backend adapter for Cromwell to facilitate improved scalability and cross region S3 bucket access when deployed with Amazon Genomics CLI. The fork containing these customizations is available [here](#) and we are working to contribute these back to the main code base.

### Architecture

There are four components of a Cromwell engine as deployed in an Amazon Genomics CLI context.



### WES Adapter

Amazon Genomics CLI communicates with the Cromwell engine via a GA4GH [WES](#) REST service. The WES Adapter implements the WES standard and translates WES calls into calls to the [Cromwell REST API](#). The adapter runs as an Amazon ECS service available via API Gateway.

## Cromwell Server

The Cromwell engine is run in “server mode” as a container service in ECS and receives instructions from the WES Adapter. The engine can run multiple workflows asynchronously. Workflow tasks are run in an elastic [compute environment](#) and monitored by Cromwell.

## Session Cache

Cromwell can use workflow run metadata to perform call caching. When deployed by Amazon Genomics CLI call caching is enabled by default. Metadata is stored by an embedded HSQL DB with file storage in an attached EFS volume. The EFS volume exists for the lifetime of the context the engine is deployed in so re-runs of workflows within the lifetime can benefit from call caching.

## Task Compute Environment

Workflow tasks are submitted by Cromwell to an AWS Batch queue and run in containers using an AWS Compute Environment. Container characteristics are defined by the `runtime`. AWS Batch coordinates the elastic provisioning of EC2 instances (container hosts) based on the available work in the queue. Batch will place containers on container hosts as space allows.

## Fetch and Run Strategy

Execution of workflow tasks uses a “Fetch and Run” strategy. The commands specified in the `command` section of the WDL task are written as a file to S3 and “fetched” into the container and run. The script is “decorated” with instructions to fetch any `File` inputs from S3 and to write any `File` outputs back to S3.

## Disk Expansion

Container hosts in the Batch compute environment use EBS volumes as local scratch space. As an EBS volume approaches a capacity threshold, new EBS volumes will be attached and merged into the file system. These volumes are destroyed when AWS Batch terminates the container host. For this reason it is not necessary to specify disk requirements for the task `runtime` and these WDL directives will be ignored.

## AWS Batch Retries

The Cromwell AWS Batch backend supports AWS Batch’s task [retry](#) option allowing failed tasks to attempt to run again. This can be useful for adding resilience to a workflow from sporadic infrastructure failures. It is especially useful when using an

Amazon Genomics CLI “spot” context as spot instances can be terminated with minimal warning. To enable retries, add the following option to your `runtime` section of a task:

```
runtime {  
  ...  
  awsBatchRetryAttempts: <int>  
  ...  
}
```

where `<int>` is an integer specifying the number of retries up to a maximum of `10`.

Although similar to the WDL `preemptible` option, `awsBatchRetryAttempts` has differences in how retries are implemented. Notably, the implementation falls back on the AWS Batch retry strategy and will retry a task that fails for **any** reason; whereas the `preemptible` option is more specific to failures caused by preemption. At this time the `preemptible` option is not supported by Amazon Genomics CLI and is ignored.

## 7.5 - Nextflow

Details on the Nextflow engine deployed by Amazon Genomics CLI

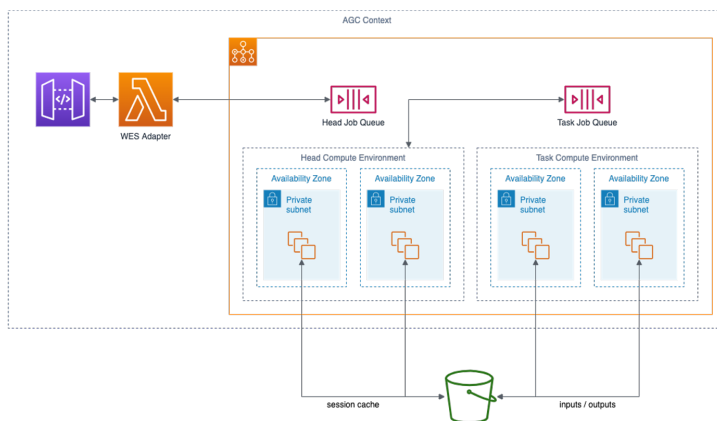
### Description

[Nextflow](#) is free open source software distributed under the Apache 2.0 licence developed by [Sequera Labs](#). The project was started in the Notredame Lab at the [Centre for Genomic Regulation \(CRG\)](#).

The source code for Nextflow is available on [GitHub](#).

### Architecture

There are four components of a Nextflow engine as deployed in an Amazon Genomics CLI context:



### WES Adapter

Amazon Genomics CLI communicates with the Nextflow engine via a GA4GH [WES](#) REST service. The WES Adapter implements the WES standard and translates WES calls into calls to the Nextflow head process.

### Head Compute Environment

For every workflow submitted, the WES adapter will create a new AWS Batch Job that contains the Nextflow process responsible for running that workflow. These Nextflow “head” jobs are run in an “On-demand” compute environment even when the actual workflow tasks run in a Spot environment. This is to prevent Spot interruptions from terminating the workflow coordinator.

### Task Compute Environment

Workflow tasks are submitted by the Nextflow head job to an AWS Batch queue and run in containers using an AWS Compute Environment. Container characteristics are defined by the resources requested in the workflow configuration. AWS Batch coordinates the elastic provisioning of EC2 instances (container hosts) based on the available work in the queue. Batch will place containers on container hosts as space allows.

## Fetch and Run Strategy

Execution of workflow tasks uses a “Fetch and Run” strategy. Input files required by a workflow task are fetched from S3 into the task container. Output files are copied out of the container to S3.

## Disk Expansion

Container hosts in the Batch compute environment use EBS volumes as local scratch space. As an EBS volume approaches a capacity threshold, new EBS volumes will be attached and merged into the file system. These volumes are destroyed when AWS Batch terminates the container host.

## 7.6 - Snakemake

Details on the Snakemake engine deployed by Amazon Genomics CLI

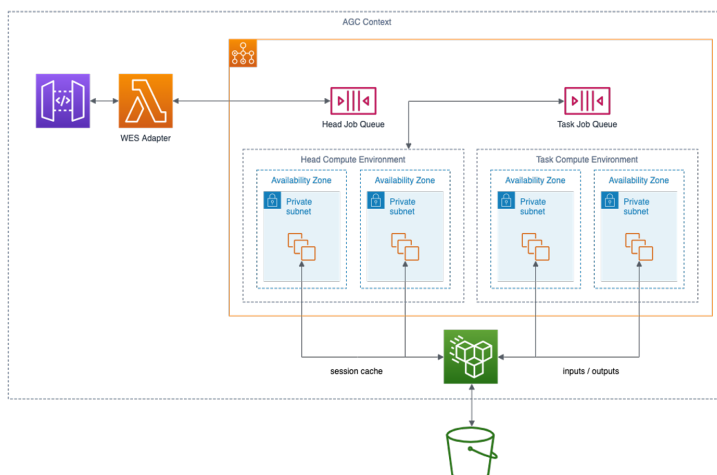
### Description

[Snakemake](#) is free open source software distributed under the MIT licence developed by [Johannes Köster and their team](#).

The source code for snakemake is available on [GitHub](#). When deployed with Amazon Genomics CLI snakemake uses Batch to distribute the underlying tasks.

### Architecture

There are four components of a snakemake engine as deployed in an Amazon Genomics CLI context:



### WES Adapter

Amazon Genomics CLI communicates with the snakemake engine via a GA4GH [WES](#) REST service. The WES Adapter implements the WES standard and translates WES calls into calls to the snakemake head process.

### Head Compute Environment

For every workflow submitted, the WES adapter will create a new AWS Batch Job that contains the snakemake process responsible for running that workflow. These snakemake “head” jobs are run in an “On-demand” AWS Fargate compute environment even when the actual workflow tasks run in a Spot environment. This is to prevent Spot interruptions from terminating the workflow coordinator.

## Task Compute Environment

Workflow tasks are submitted by the snakemake head job to an AWS Batch queue and run in containers using an AWS Compute Environment. Container characteristics are defined by the resources requested in the workflow configuration. AWS Batch coordinates the elastic provisioning of EC2 instances (container hosts) based on the available work in the queue. Batch will place containers on container hosts as space allows.

## Session Cache and Input Localization

Any context with a snakemake engine will use an Amazon Elastic File System (EFS) volume as scratch space. Inputs from the workflow are localized to the volume by jobs that the snakemake engine spawns to copy these files to the volume. Outputs are copied back to S3 after the workflow is complete. Workflow tasks access the EFS volume to obtain inputs and write intermediates and outputs.

The EFS volume can be used by all snakemake engine “head” jobs to store metadata necessary for dependency caching by specifying an argument for the conda workspace that is common across all executions. An example of this is `--conda-prefix /mnt/efs/snakemake/conda`.

The EFS volume will remain in your account for the lifetime of the context and are destroyed when contexts are destroyed. Because the volume will grow in size as you run more workflows we recommend destroying the context when done to avoid on-going EFS charges.

## Using Snakemake as a Context Engine

You may declare snakemake to be the `engine` for any contexts `snakemake` type engine. For example:

```
contexts:
  onDemandCtx:
    requestSpotInstances: false
  engines:
    - type: snakemake
      engine: snakemake
```

## Conda Dependency Caching

Dependency caching is disabled by default so that each workflow can be run independently. If you would like workflow runs to re-use the Conda cache then please specify a folder

under `"/mnt/efs"` which is where the EFS storage space is attached. This will enable snakemake to re-use the dependency which will decrease the time that subsequent workflow runs will take.

To disable call caching you can provide the `--conda-prefix` engine option. You may do this in a workflows `MANIFEST.json` by adding the following key/ value pair.

```
"engineOptions": "-j 10 --conda-prefix /mnt/efs/snake"
```



# 8 - Contribution Guidelines

How to contribute to AWS Genomics CLI

## Attention

**The Amazon Genomics CLI project has entered its End Of Life (EOL) phase.** The code is no longer actively maintained and the **Github repository will be archived on May 31 2024**. During this time, we encourage customers to migrate to [AWS HealthOmics](#) to run their genomics workflows on AWS, or [reach out to their AWS account team](#) for alternative solutions. While the source code of AGC will still be available after the EOL date, we will not make any updates inclusive of addressing issues or accepting Pull Requests.

We use Go as the main language of the AWS Genomics CLI. To interact with AWS Services we use the AWS SDK for Go v2. Infrastructure to be deployed into an AWS Account is coded as TypeScript using the AWS CDK.

All submissions, including submissions by project members, require review. We use GitHub pull requests for this purpose. Consult [GitHub Help](#) for more information on using pull requests.

Please refer to our [Contributing](#) document on GitHub for specifics regarding our Pull request procedures, bug reports, feature requests, code of conduct and license terms.

## Useful resources

- [Github Hello World!](#): A basic introduction to GitHub concepts and workflow.
- [Golang](#): Information about the Go language
- [TypeScript](#): Information about the Go language
- [AWS CDK](#): Developer guide to the AWS Cloud Development Kit (CDK)
- [AWS SDK for Go v2](#): Developer guide for the AWS SDK for Go v2
- [Docsy user guide](#): All about Docsy, including how it manages navigation, look and feel, and multi-language support.
- [Hugo documentation](#): Comprehensive reference for Hugo.

## 9 - Reference

Command reference.

### Attention

**The Amazon Genomics CLI project has entered its End Of Life (EOL) phase.** The code is no longer actively maintained and the **Github repository will be archived on May 31 2024**. During this time, we encourage customers to migrate to [AWS HealthOmics](#) to run their genomics workflows on AWS, or [reach out to their AWS account team](#) for alternative solutions. While the source code of AGC will still be available after the EOL date, we will not make any updates inclusive of addressing issues or accepting Pull Requests.

## Command Reference

# 9.1 - agc

## agc

 Launch and manage genomics workloads on AWS.

```
agc [flags]
```

## Examples

```
Displays the help menu for the specified sub-command.  
/code $ agc account --help
```

## Options

<code>--format string</code>	Format option for output. Valid
<code>-h, --help</code>	help for agc
<code>--silent</code>	Suppresses all diagnostic infor
<code>-v, --verbose</code>	Display verbose diagnostic info
<code>--version</code>	version for agc

## SEE ALSO

- [agc account](#) - Commands for AWS account setup. Install or remove AGC from your account.
- [agc completion](#) - generate the autocompletion script for the specified shell
- [agc configure](#) - Commands for configuration. Configuration is stored per user.
- [agc context](#) - Commands for contexts. Contexts specify workflow engines and computational fleets to use when running a workflow.
- [agc logs](#) - Commands for various logs.
- [agc project](#) - Commands to interact with projects.
- [agc workflow](#) - Commands for workflows. Workflows are potentially-dynamic graphs of computational tasks to execute.

## 9.2 - agc account

### agc account

Commands for AWS account setup. Install or remove AGC from your account.

#### Synopsis

Commands for AWS account setup. AGC requires core infrastructure to be running in an account to function. These commands should be used to install or remove AGC from your AWS account.


#### Options

<code>-p, --awsProfile string</code>	Use the provided AWS CLI profile
<code>-h, --help</code>	help for account

#### Options inherited from parent commands

<code>--format string</code>	Format option for output. Valid options are: json, text, table.
<code>--silent</code>	Suppresses all diagnostic information.
<code>-v, --verbose</code>	Display verbose diagnostic information.

#### SEE ALSO

- [agc](#) -  Launch and manage genomics workloads on AWS.
- [agc account activate](#) - Activate AGC in an AWS account.
- [agc account deactivate](#) - Deactivate AGC in an AWS account.

## 9.3 - agc account activate

### agc account activate

Activate AGC in an AWS account.

#### Synopsis

Activate AGC in an AWS account. AGC will use your default AWS credentials to deploy all AWS resources it needs to that account and region.

```
agc account activate [flags]
```

#### Examples

```
Activate AGC in your AWS account with a custom S3 bucket
/code $ agc account activate --bucket my-custom-bucket
```

#### Options

<code>--ami</code> string	The AMI that will be used
<code>--bucket</code> string	The name of an S3 bucket
<code>-h, --help</code>	An autogenerated name will help for activate
<code>--subnets</code> strings	The list of private subnets
	'--vpc' flag. If not supplied
	Each subnet must have access
	Subnet names may be a comma
<code>--tags</code> stringToString	A list of comma separated
	(e.g. <code>--tags "k1=v1","k2=v2"</code> )
	otherwise the parsing will fail
<code>--usePublicSubnets</code>	Do not create a NAT gateway
	You must enable the usePublicSubnets
<code>--vpc</code> string	The ID of a VPC that AGC will use
	A new VPC will be created if not found

#### Options inherited from parent commands

<code>-p, --awsProfile</code> string	Use the provided AWS CLI profile
<code>--format</code> string	Format option for output. Valid values are json, text, and yaml
<code>--silent</code>	Suppresses all diagnostic information
<code>-v, --verbose</code>	Display verbose diagnostic information

## SEE ALSO

- [agc account](#) - Commands for AWS account setup. Install or remove AGC from your account.

## 9.4 - agc account deactivate

### agc account deactivate

Deactivate AGC in an AWS account.

#### Synopsis

Deactivate AGC in an AWS account. AGC will use your default AWS credentials to remove all core AWS resources it has created in that account and region. Deactivation may take up to 5 minutes to complete and return. Buckets and logs will be preserved.

```
agc account deactivate [flags]
```

#### Examples

```
Deactivate AGC in your AWS account.  
/code $ agc account deactivate
```

#### Options

```
-f, --force    Force account deactivation by removing  
              This includes project and context resources.  
              If not specified, only the core resources are removed.  
-h, --help    help for deactivate
```

#### Options inherited from parent commands

```
-p, --awsProfile string  Use the provided AWS CLI profile  
--format string          Format option for output. Valid values are json, text, and yaml.  
--silent                Suppresses all diagnostic information.  
-v, --verbose            Display verbose diagnostic information.
```

#### SEE ALSO

- [agc account](#) - Commands for AWS account setup. Install or remove AGC from your account.

## 9.5 - agc completion

### agc completion

generate the autocompletion script for the specified shell

#### Synopsis

Generate the autocompletion script for agc for the specified shell. See each sub-command's help for details on how to use the generated script.


#### Options

<code>-h, --help</code>	help for completion
-------------------------	---------------------

#### Options inherited from parent commands

<code>--format string</code>	Format option for output. Valid
<code>--silent</code>	Suppresses all diagnostic infor
<code>-v, --verbose</code>	Display verbose diagnostic info

#### SEE ALSO

- [agc](#) -  Launch and manage genomics workloads on AWS.
- [agc completion bash](#) - generate the autocompletion script for bash
- [agc completion fish](#) - generate the autocompletion script for fish
- [agc completion powershell](#) - generate the autocompletion script for powershell
- [agc completion zsh](#) - generate the autocompletion script for zsh



## 9.6 - agc completion bash

### agc completion bash

generate the autocompletion script for bash

#### Synopsis

Generate the autocompletion script for the bash shell.

This script depends on the 'bash-completion' package. If it is not installed already, you can install it via your OS's package manager.

To load completions in your current shell session: \$ source <(agc completion bash)

To load completions for every new session, execute once: Linux: \$ agc completion bash > /etc/bash\_completion.d/agc MacOS: \$ agc completion bash > /usr/local/etc/bash\_completion.d/agc

You will need to start a new shell for this setup to take effect.

```
agc completion bash
```

#### Options

-h, --help	help for bash
--no-descriptions	disable completion descriptions

#### Options inherited from parent commands

--format string	Format option for output. Valid
--silent	Suppresses all diagnostic infor
-v, --verbose	Display verbose diagnostic info

#### SEE ALSO

- [agc completion](#) - generate the autocompletion script for the specified shell

## 9.7 - agc completion fish

### agc completion fish

generate the autocompletion script for fish

#### Synopsis

Generate the autocompletion script for the fish shell.

To load completions in your current shell session: \$ agc completion fish | source

To load completions for every new session, execute once: \$ agc completion fish > ~/.config/fish/completions/agc.fish

You will need to start a new shell for this setup to take effect.

```
agc completion fish [flags]
```

#### Options

-h, --help	help for fish
--no-descriptions	disable completion descriptions

#### Options inherited from parent commands

--format string	Format option for output. Valid
--silent	Suppresses all diagnostic infor
-v, --verbose	Display verbose diagnostic info

#### SEE ALSO

- [agc completion](#) - generate the autocompletion script for the specified shell

## 9.8 - agc completion powershell

### agc completion powershell

generate the autocompletion script for powershell

#### Synopsis

Generate the autocompletion script for powershell.

To load completions in your current shell session: PS C:> agc completion powershell | Out-String | Invoke-Expression

To load completions for every new session, add the output of the above command to your powershell profile.

```
agc completion powershell [flags]
```

#### Options

<code>-h, --help</code>	help for powershell
<code>--no-descriptions</code>	disable completion descriptions

#### Options inherited from parent commands

<code>--format string</code>	Format option for output. Valid values are: json, text, table, yaml
<code>--silent</code>	Suppresses all diagnostic information
<code>-v, --verbose</code>	Display verbose diagnostic information

#### SEE ALSO

- [agc completion](#) - generate the autocompletion script for the specified shell

## 9.9 - agc completion zsh

### agc completion zsh

generate the autocompletion script for zsh

#### Synopsis

Generate the autocompletion script for the zsh shell.

If shell completion is not already enabled in your environment you will need to enable it. You can execute the following once:

```
$ echo "autoload -U compinit; compinit" » ~/.zshrc
```

To load completions for every new session, execute once:

### Linux:

```
$ agc completion zsh > "${fpath[1]}/_agc"
```

### macOS:

```
$ agc completion zsh > /usr/local/share/zsh/site-functions/_agc
```

You will need to start a new shell for this setup to take effect.

```
agc completion zsh [flags]
```

#### Options

<code>-h, --help</code>	help for zsh
<code>--no-descriptions</code>	disable completion descriptions

#### Options inherited from parent commands

<code>--format string</code>	Format option for output. Valid
<code>--silent</code>	Suppresses all diagnostic infor
<code>-v, --verbose</code>	Display verbose diagnostic info

#### SEE ALSO

- [agc completion](#) - generate the autocompletion script for the specified shell

## 9.10 - agc configure

### agc configure

Commands for configuration. Configuration is stored per user.

#### Synopsis

Commands for configuration. Configure local settings and preferences to customize the CLI experience.


#### Options

<code>-p, --awsProfile string</code>	Use the provided AWS CLI profile
<code>-h, --help</code>	help for configure

#### Options inherited from parent commands

<code>--format string</code>	Format option for output. Valid options are 'text', 'table', or 'json'
<code>--silent</code>	Suppresses all diagnostic information
<code>-v, --verbose</code>	Display verbose diagnostic information

#### SEE ALSO

- [agc](#) -  Launch and manage genomics workloads on AWS.
- [agc configure describe](#) - Shows current configuration of the AGC setup for current user
- [agc configure email](#) - Sets user email address to be used to tag AGC resources created in the account
- [agc configure format](#) - Sets default format option for output display of AGC commands. Valid format options are 'text', 'table', or 'json'

## 9.11 - agc configure describe

### agc configure describe

Shows current configuration of the AGC setup for current user

#### Synopsis

Running this command reads current configuration file for AGC and prints out its content. Output of the command has the following format: CONFIG: FORMAT: Name USER: Email Id

```
agc configure describe [flags]
```

#### Options

```
-h, --help    help for describe
```

#### Options inherited from parent commands

```
-p, --awsProfile string  Use the provided AWS CLI profile
--format string          Format option for output. Valid values are json, yaml, and text
--silent                 Suppresses all diagnostic information
-v, --verbose             Display verbose diagnostic information
```

#### SEE ALSO

- [agc configure](#) - Commands for configuration. Configuration is stored per user.

## 9.12 - agc configure email

### agc configure email

Sets user email address to be used to tag AGC resources created in the account

```
agc configure email user_email_address [flags]
```

#### Options

```
-h, --help    help for email
```

#### Options inherited from parent commands

```
-p, --awsProfile string  Use the provided AWS CLI pr
--format string          Format option for output. V
--silent                 Suppresses all diagnostic i
-v, --verbose            Display verbose diagnostic
```

#### SEE ALSO

- [agc configure](#) - Commands for configuration.  
Configuration is stored per user.

## 9.13 - agc configure format

### agc configure format

Sets default format option for output display of AGC commands. Valid format options are 'text', 'table', or 'json'

```
agc configure format output_format [flags]
```

#### Options

```
-h, --help    help for format
```

#### Options inherited from parent commands

```
-p, --awsProfile string  Use the provided AWS CLI pr
    --format string      Format option for output. V
    --silent             Suppresses all diagnostic i
-v, --verbose            Display verbose diagnostic
```

#### SEE ALSO

- [agc configure](#) - Commands for configuration.  
Configuration is stored per user.



## 9.14 - agc context

### agc context

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

### Synopsis

Commands for contexts. Contexts specify workflow engines and computational fleets to use when running a workflow. Users can quickly switch between infrastructure configurations by specifying a particular context.


### Options

<code>-p, --awsProfile string</code>	Use the provided AWS CLI profile
<code>-h, --help</code>	help for context

### Options inherited from parent commands

<code>--format string</code>	Format option for output. Valid options are json, yaml, and text.
<code>--silent</code>	Suppresses all diagnostic information.
<code>-v, --verbose</code>	Display verbose diagnostic information.

### SEE ALSO

- [agc](#) -  Launch and manage genomics workloads on AWS.
- [agc context deploy](#) - Deploy contexts in the current project
- [agc context describe](#) - Show the information for a specific context in the current project
- [agc context destroy](#) - Destroy contexts in the current project.
- [agc context list](#) - List contexts in the project
- [agc context status](#) - Show the status for the deployed contexts in the project

## 9.15 - agc context deploy

### agc context deploy

Deploy contexts in the current project

#### Synopsis

deploy is for deploying one or more contexts. It creates AGC resources in AWS.

Output of the command has following format: DETAIL:

AccessLogGroupName BucketLocation EngineLogGroupName

Status StatusReason WesLogGroupName WesUrl SUMMARY:

IsSpot MaxVCpus Name ENGINE: Engine Type FILESYSTEM:

FSType FSCONFIG: FSProvisionedThroughput STRING

```
agc context deploy {context_name ... | --all} [flags]
```

#### Examples

```
/code agc context deploy context1 context2
```

#### Options

<code>--all</code>	Deploy all contexts in the project
<code>-c, --context strings</code>	Names of one or more contexts to deploy
<code>-h, --help</code>	help for deploy

#### Options inherited from parent commands

<code>-p, --awsProfile string</code>	Use the provided AWS CLI profile
<code>--format string</code>	Format option for output. Valid options are json, text, and table
<code>--silent</code>	Suppresses all diagnostic information
<code>-v, --verbose</code>	Display verbose diagnostic information

#### SEE ALSO

- [agc context](#) - Commands for contexts. Contexts specify workflow engines and computational fleets to use when running a workflow.

## 9.16 - agc context describe

### agc context describe

Show the information for a specific context in the current project

#### Synopsis

describe is for showing information about the specified context.

Output of the command has following format: CONTEXT:  
MaxVCpus Name RequestSpotInstances Status StatusReason  
INSTANCETYPE: Value OUTPUTLOCATION: Url WESENDPOINT:  
Url

```
agc context describe context_name [flags]
```

#### Examples

```
/code agc context describe myCtx
```

#### Options

<code>-c, --context string</code>	Names of one or more contexts
<code>-h, --help</code>	help for describe

#### Options inherited from parent commands

<code>-p, --awsProfile string</code>	Use the provided AWS CLI profile
<code>--format string</code>	Format option for output. Valid values are json, text, and yaml
<code>--silent</code>	Suppresses all diagnostic messages
<code>-v, --verbose</code>	Display verbose diagnostic messages

#### SEE ALSO

- [agc context](#) - Commands for contexts. Contexts specify workflow engines and computational fleets to use when running a workflow.

## 9.17 - agc context destroy

### agc context destroy

Destroy contexts in the current project.

#### Synopsis

destroy is for destroying one or more contexts. It destroys AGC resources in AWS.

```
agc context destroy {context_name ... | --all} [flags]
```

#### Examples

```
/code agc context destroy context1 context2
```

#### Options

<code>--all</code>	Destroy all contexts in the p
<code>-c, --context strings</code>	Names of one or more contexts
<code>--force</code>	Destroy context and stop runn
<code>-h, --help</code>	help for destroy

#### Options inherited from parent commands

<code>-p, --awsProfile string</code>	Use the provided AWS CLI pr
<code>--format string</code>	Format option for output. V
<code>--silent</code>	Suppresses all diagnostic i
<code>-v, --verbose</code>	Display verbose diagnostic

#### SEE ALSO

- [agc context](#) - Commands for contexts. Contexts specify workflow engines and computational fleets to use when running a workflow.

## 9.18 - agc context list

### agc context list

List contexts in the project

#### Synopsis

list is for showing a combined list of contexts specified in the project specification.

Output of the command has following format:

CONTEXTSUMMARY: EngineName Name

```
agc context list [flags]
```

#### Options

```
-h, --help    help for list
```

#### Options inherited from parent commands

```
-p, --awsProfile string  Use the provided AWS CLI pr
--format string          Format option for output. V
--silent                 Suppresses all diagnostic i
-v, --verbose            Display verbose diagnostic
```

#### SEE ALSO

- [agc context](#) - Commands for contexts. Contexts specify workflow engines and computational fleets to use when running a workflow.

## 9.19 - agc context status

### agc context status

Show the status for the deployed contexts in the project

#### Synopsis

status is for showing the status for the deployed contexts in the project.

Output of the command has following format: INSTANCE:

ContextName ContextReason ContextStatus

IsDefinedInProjectFile

```
agc context status [flags]
```

#### Examples

```
/code agc context status
```

#### Options

```
-h, --help    help for status
```

#### Options inherited from parent commands

```
-p, --awsProfile string  Use the provided AWS CLI pr
--format string          Format option for output. V
--silent                 Suppresses all diagnostic i
-v, --verbose            Display verbose diagnostic
```

#### SEE ALSO

- [agc context](#) - Commands for contexts. Contexts specify workflow engines and computational fleets to use when running a workflow.

## 9.20 - agc logs

### agc logs

Commands for various logs.

#### Synopsis

Commands for various logs. Logs can currently be listed for workflows, workflow engines, and various AGC infrastructure parts. You can also show the content of any CloudWatch log stream that you have access rights to.


#### Options

<code>-p, --awsProfile string</code>	Use the provided AWS CLI profile
<code>-h, --help</code>	help for logs

#### Options inherited from parent commands

<code>--format string</code>	Format option for output. Valid options are json, text, and table.
<code>--silent</code>	Suppresses all diagnostic information.
<code>-v, --verbose</code>	Display verbose diagnostic information.

#### SEE ALSO

- [agc](#) -  Launch and manage genomics workloads on AWS.
- [agc logs access](#) - Show workflow access logs for a given context.
- [agc logs adapter](#) - Show workflow adapter logs for a given context.
- [agc logs engine](#) - Show workflow engine logs for a given context.
- [agc logs workflow](#) - Show the task logs of a given workflow

## 9.21 - agc logs access

### agc logs access

Show workflow access logs for a given context.

#### Synopsis

Show workflow access logs for a given context. If no start, end, or look back periods are set, this command will show logs from the last hour.

```
agc logs access -c context_name [-f filter] [-s start_c
```

#### Examples

```
/code agc logs access -c myCtx -s 2021/3/31 -e 2021/4/1
```

#### Options

<code>-c, --context string</code>	Name of context
<code>-e, --end string</code>	A date to stop displaying logs Supports most date formats, Times respect the system time
<code>-f, --filter string</code>	Match terms, phrases, or values Filters are case sensitive and Use a question mark for OR,
<code>-h, --help</code>	help for access
<code>-l, --look-back string</code>	A period of time to look back Valid time units are "s", "m",
<code>-s, --start string</code>	A date to begin displaying logs Supports most date formats, Times respect the system time
<code>-t, --tail</code>	Follow the log output.

#### Options inherited from parent commands

<code>-p, --awsProfile string</code>	Use the provided AWS CLI profile
<code>--format string</code>	Format option for output. Valid
<code>--silent</code>	Suppresses all diagnostic information
<code>-v, --verbose</code>	Display verbose diagnostic information

#### SEE ALSO

- [agc logs](#) - Commands for various logs.



## 9.22 - agc logs adapter

### agc logs adapter

Show workflow adapter logs for a given context.

#### Synopsis

Show workflow adapter logs for a given context. If no start, end, or look back periods are set, this command will show logs from the last hour.

```
agc logs adapter -c context_name [-f filter] [-s start_
```

#### Examples

```
/code agc logs adapter -c myCtx -s 2021/3/31 -e 2021/4/
```

#### Options

<code>-c, --context string</code>	Name of context
<code>-e, --end string</code>	A date to stop displaying logs Supports most date formats, Times respect the system time
<code>-f, --filter string</code>	Match terms, phrases, or values Filters are case sensitive and Use a question mark for OR,
<code>-h, --help</code>	help for adapter
<code>-l, --look-back string</code>	A period of time to look back Valid time units are "s", "m",
<code>-s, --start string</code>	A date to begin displaying logs Supports most date formats, Times respect the system time
<code>-t, --tail</code>	Follow the log output.

#### Options inherited from parent commands

<code>-p, --awsProfile string</code>	Use the provided AWS CLI profile
<code>--format string</code>	Format option for output. Valid
<code>--silent</code>	Suppresses all diagnostic information
<code>-v, --verbose</code>	Display verbose diagnostic information

#### SEE ALSO

- [agc logs](#) - Commands for various logs.

# 9.23 - agc logs engine

## agc logs engine

Show workflow engine logs for a given context.

### Synopsis

Show workflow engine logs for a given context. If no start, end, or look back periods are set, this command will show logs from the last hour.

```
agc logs engine -c context_name [-r run_id] [-f filter]
```

### Examples

```
/code agc logs engine -c myCtx -r 1234-aed-32db -s 2021
```

### Options

-c, --context string	Name of context
-e, --end string	A date to stop displaying logs. Supports most date formats, Times respect the system time.
-f, --filter string	Match terms, phrases, or values. Filters are case sensitive and use a question mark for OR, help for engine.
-h, --help	help for engine
-l, --look-back string	A period of time to look back. Valid time units are "s", "m", "h", "d", "w", "m", "y".
-r, --run-id string	filter to engine logs to this run_id.
-s, --start string	A date to begin displaying logs. Supports most date formats, Times respect the system time.
-t, --tail	Follow the log output.

### Options inherited from parent commands

-p, --awsProfile string	Use the provided AWS CLI profile.
--format string	Format option for output. Valid values are json, text, and table.
--silent	Suppresses all diagnostic information.
-v, --verbose	Display verbose diagnostic information.

### SEE ALSO

- [agc logs](#) - Commands for various logs.

# 9.24 - agc logs workflow

## agc logs workflow

Show the task logs of a given workflow

### Synopsis

Show the task logs of a given workflow. If the `-run` flag is omitted then the latest workflow run is used.

```
agc logs workflow workflow_name [-r run_id] [--failed_t
```

### Options

<code>--all-tasks</code>	Show logs of all tasks in th
<code>-e, --end string</code>	A date to stop displaying lo Supports most date formats, Times respect the system tim
<code>--failed-tasks</code>	Only show logs of tasks that
<code>-f, --filter string</code>	Match terms, phrases, or val Filters are case sensitive a Use a question mark for OR,
<code>-h, --help</code>	help for workflow
<code>-l, --look-back string</code>	A period of time to look bac Valid time units are "s", "m
<code>-r, --run string</code>	The ID of a workflow run to
<code>-s, --start string</code>	A date to begin displaying l Supports most date formats, Times respect the system tim
<code>-t, --tail</code>	Follow the log output.
<code>--task string</code>	The ID of a single task to r

### Options inherited from parent commands

<code>-p, --awsProfile string</code>	Use the provided AWS CLI pr
<code>--format string</code>	Format option for output. V
<code>--silent</code>	Suppresses all diagnostic i
<code>-v, --verbose</code>	Display verbose diagnostic

### SEE ALSO

- [agc logs](#) - Commands for various logs.

## 9.25 - agc project

### agc project

Commands to interact with projects.

#### Synopsis

Commands to interact with projects. A project is a local configuration file that describes the workflows, data, and contexts you are working with.


#### Options

<code>-h, --help</code>	help for project
-------------------------	------------------

#### Options inherited from parent commands

<code>--format string</code>	Format option for output. Valid
<code>--silent</code>	Suppresses all diagnostic infor
<code>-v, --verbose</code>	Display verbose diagnostic info

#### SEE ALSO

- [agc](#) -  Launch and manage genomics workloads on AWS.
- [agc project describe](#) - Describe a project
- [agc project init](#) - Initialize current directory with a new empty AGC project for a particular workflow type.
- [agc project validate](#) - Validate an agc-project.yaml file

## 9.26 - agc project describe

### agc project describe

Describe a project

#### Synopsis

Describe is for describing the current project specification. The current project specification is determined to be the agc-project.yaml file in the current working directory or a parent of the current directory

Output of the command has following format: PROJECT: Name  
DATA: Location ReadOnly

```
agc project describe [flags]
```

#### Examples

```
/code agc project describe
```

#### Options

```
-h, --help    help for describe
```

#### Options inherited from parent commands

```
    --format string  Format option for output. Valid values are: json, yaml, text, table  
    --silent         Suppresses all diagnostic information  
-v, --verbose       Display verbose diagnostic information
```

#### SEE ALSO

- [agc project](#) - Commands to interact with projects.

## 9.27 - agc project init

### agc project init

Initialize current directory with a new empty AGC project for a particular workflow type.

#### Synopsis

Initialize current directory with a new empty AGC project for a particular workflow type. Project specification file 'agc-project.yaml' will be created in the current directory.

```
agc project init project_name --workflow-type {cwl|nextflow}
```

#### Examples

```
Initialize a new project named "myProject".  
/code $ agc project init myProject --workflow-type my_workflow
```

#### Options

<code>-h, --help</code>	help for init
<code>-w, --workflow-type string</code>	uses the specified workflow type

#### Options inherited from parent commands

<code>--format string</code>	Format option for output. Valid values are json, yaml, and text.
<code>--silent</code>	Suppresses all diagnostic information.
<code>-v, --verbose</code>	Display verbose diagnostic information.

#### SEE ALSO

- [agc project](#) - Commands to interact with projects.

## 9.28 - agc project validate

### agc project validate

Validate an agc-project.yaml file

#### Synopsis

Determines if the current project specification follows the required format and lists any syntax errors. The current project specification is determined to be the agc-project.yaml file in the current working directory or a parent of the current directory. Output of the command has following format: PROJECT: Name  
DATA: Location ReadOnly

```
agc project validate [flags]
```

#### Examples

```
/code agc project describe
```

#### Options

```
-h, --help    help for validate
```

#### Options inherited from parent commands

--format string	Format option for output. Valid
--silent	Suppresses all diagnostic infor
-v, --verbose	Display verbose diagnostic info

#### SEE ALSO

- [agc project](#) - Commands to interact with projects.

## 9.29 - agc workflow

### agc workflow

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

#### Synopsis

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

Workflow specifications are files whose content specify a workflow to execute given a particular set of input parameters to use. Workflow specifications are typed according to which workflow definition language they use (e.g. WDL).


#### Options

<code>-p, --awsProfile string</code>	Use the provided AWS CLI profile
<code>-h, --help</code>	help for workflow

#### Options inherited from parent commands

<code>--format string</code>	Format option for output. Valid options are: json, text, yaml
<code>--silent</code>	Suppresses all diagnostic information
<code>-v, --verbose</code>	Display verbose diagnostic information

#### SEE ALSO

- [agc](#) -  Launch and manage genomics workloads on AWS.
- [agc workflow describe](#) - Show the information for a specific workflow in the current project
- [agc workflow list](#) - Show a list of workflows related to the current project
- [agc workflow output](#) - Show the output for a workflow run in the current project.
- [agc workflow run](#) - Run a workflow
- [agc workflow status](#) - Show the status for workflow run(s) in the current project.
- [agc workflow stop](#) - Stop the workflow with the specified workflow instance id.



## 9.30 - agc workflow describe

### agc workflow describe

Show the information for a specific workflow in the current project

#### Synopsis

describe is for showing details on the specified workflow. It includes workflow specification and list of recent instances of that workflow. An instance is created every time we run a workflow.

Output of the command has following format: WORKFLOW:  
Name Source TypeLanguage TypeVersion

```
agc workflow describe workflow_name [--max_instances] [
```

#### Options

```
-h, --help    help for describe
```

#### Options inherited from parent commands

```
-p, --awsProfile string  Use the provided AWS CLI pr
--format string          Format option for output. V
--silent                 Suppresses all diagnostic i
-v, --verbose            Display verbose diagnostic
```

#### SEE ALSO

- [agc workflow](#) - Commands for workflows. Workflows are potentially-dynamic graphs of computational tasks to execute.

## 9.31 - agc workflow list

### agc workflow list

Show a list of workflows related to the current project

#### Synopsis

list is for showing a combined list of workflows defined in the project specification and workflow instances that were run in this AWS account.

Output of the command has following format:

WORKFLOWNAME: Name

```
agc workflow list [flags]
```

#### Options

```
-h, --help    help for list
```

#### Options inherited from parent commands

```
-p, --awsProfile string  Use the provided AWS CLI pr
--format string          Format option for output. V
--silent                 Suppresses all diagnostic i
-v, --verbose            Display verbose diagnostic
```

#### SEE ALSO

- [agc workflow](#) - Commands for workflows. Workflows are potentially-dynamic graphs of computational tasks to execute.

## 9.32 - agc workflow output

### agc workflow output

Show the output for a workflow run in the current project.

```
agc workflow output run_id [flags]
```

#### Options

```
-h, --help    help for output
```

#### Options inherited from parent commands

```
-p, --awsProfile string  Use the provided AWS CLI pr
    --format string      Format option for output. V
    --silent             Suppresses all diagnostic i
    -v, --verbose         Display verbose diagnostic
```

#### SEE ALSO

- [agc workflow](#) - Commands for workflows. Workflows are potentially-dynamic graphs of computational tasks to execute.

## 9.33 - agc workflow run

### agc workflow run

Run a workflow

#### Synopsis

run is for running the specified workflow in the specified context. This command prints a run Id for the created workflow instance.

```
agc workflow run workflow_name --context context_name [options]
```

#### Examples

```
Run the workflow named "myworkflow", against the "prod"
using input parameters contained in file "/home/ec2-user/inputs/inputs.json"
/code $ agc workflow run myworkflow --context prod --inputsFile inputs.json
```

#### Options

-c, --context string	Name of context
-h, --help	help for run
-i, --inputsFile string	Inputs File Path
-o, --optionsFile string	Options file to use.

#### Options inherited from parent commands

-p, --awsProfile string	Use the provided AWS CLI profile
--format string	Format option for output. Valid options are json, text, and yaml.
--silent	Suppresses all diagnostic messages
-v, --verbose	Display verbose diagnostic messages

#### SEE ALSO

- [agc workflow](#) - Commands for workflows. Workflows are potentially-dynamic graphs of computational tasks to execute.

## 9.34 - agc workflow status

### agc workflow status

Show the status for workflow run(s) in the current project.

```
agc workflow status [flags]
```

#### Options

<code>-c, --context-name string</code>	show status of workflow
<code>-h, --help</code>	help for status
<code>--limit int</code>	maximum number of workfl
<code>-r, --run-id string</code>	show status of specific
<code>-n, --workflow-name string</code>	show status of workflow

#### Options inherited from parent commands

<code>-p, --awsProfile string</code>	Use the provided AWS CLI pr
<code>--format string</code>	Format option for output. V
<code>--silent</code>	Suppresses all diagnostic i
<code>-v, --verbose</code>	Display verbose diagnostic

#### SEE ALSO

- [agc workflow](#) - Commands for workflows. Workflows are potentially-dynamic graphs of computational tasks to execute.

## 9.35 - agc workflow stop

### agc workflow stop

Stop the workflow with the specified workflow instance id.

#### Synopsis

Stop the workflow with the specified workflow instance id. Signals to the workflow engine that all running tasks of the workflow instance should be stopped and any pending tasks should be cancelled.

```
agc workflow stop workflow_instance_id [flags]
```

#### Examples

```
agc workflow stop ae12347654329
```

#### Options

```
-h, --help    help for stop
```

#### Options inherited from parent commands

```
-p, --awsProfile string  Use the provided AWS CLI profile
--format string          Format option for output. Valid values are json,
--silent                 Suppresses all diagnostic messages
-v, --verbose            Display verbose diagnostic messages
```

#### SEE ALSO

- [agc workflow](#) - Commands for workflows. Workflows are potentially-dynamic graphs of computational tasks to execute.



