Genomics Secondary Analysis Using AWS Step Functions and AWS Batch

AWS Developer Guide

Lee Pang

Ryan Ulaszek

Michael Miller

April 2020

Contents

About This Guide

This developer guide provides information about customizing and extending the Genomics Secondary Analysis Using AWS Step Functions and AWS Batch solution. It includes information about the custom configuration package structure, bioinformatics tools and workflows, configuration templates, pipeline deployment stages, and dashboards.

The guide is intended for IT infrastructure architects, administrators, bioinformatics scientists, DevOps professionals, and software engineers who want to customize and extend the Genomics Secondary Analysis Using AWS Step Functions and AWS Batch solution for their company or customers.

Customizations for Solution Deployment

The Genomics Secondary Analysis Using AWS Step Functions and AWS Batch solution creates AWS CodeCommit repositories containing the source codes that define (1) the continuous integration and continuous delivery (CI/CD) pipeline for the genomics workflow and (2) the containerized tools (Dockerfiles), workflow definitions (such as the Amazon States Language source for an AWS Step Functions state machine), and job execution resources including AWS Batch job definitions, job queues, and compute environments.

This solution also provides example code to run an example bioinformatics workflow and dataset. You can replace this example code and push your code into the AWS CodeCommit `code` repository, which triggers associated AWS CodePipeline pipelines. The following sections detail how to replace the example code and customize the solution to fit your needs.

Add New Bioinformatics Tools

Open source bioinformatics tools are built as container images, which are pushed to an Amazon Elastic Compute Cloud (Amazon EC2) Container Registry (ECR) in your AWS account. Use the following steps to add a new bioinformatics tool.

1. Update the CI/CD pipeline:
    a. Clone source code or pull source code updates from the CI/CD pipeline source code repository in AWS CodeCommit.
    b. Create an AWS CodeBuild project for the tool’s container image.
    c. Create an AWS CodePipeline stage action to build the tool’s container image.
    d. Commit and push changes to the AWS CodeCommit `GenomicsWorkflowPipe` repository to trigger an update of the CI/CD pipeline.

2. Update the workflow code:
    a. Clone source code or pull source code updates from the workflow source code repository in AWS CodeCommit.
    b. Create a Dockerfile that defines how to create the tool's container image.
    c. Create an AWS Batch job definition that references the tool's container image.
    d. Commit and push changes to the CodeCommit `GenomicsWorkflowCode` repository to trigger an execution of the updated CI/CD pipeline and build of the new tool(s).

The following topics provide more detail about the process for adding new bioinformatics tools.

Update the CI/CD Pipeline

The solution's CI/CD pipeline is defined by source code contained in the AWS CloudFormation template. This source code is stored in an AWS CodeCommit repository and defines the AWS CodeBuild projects that build the solution's artifacts (for example, container images). The source code also defines an AWS CodePipeline pipeline to coordinate multiple build projects and automate the deployment (using AWS CloudFormation) of the secondary analysis workflow and supporting resources. You can enable new tools for workflows by updating the CI/CD pipeline and creating a CodeBuild project and a CodePipeline stage action for each tool you wish to add. After you make all requisite CI/CD pipeline code changes, committing and pushing changes to the CodeCommit `GenomicsWorkflowPipe` repository triggers CI/CD pipeline redeployment.

Clone Source Code or Pull Source Code Updates

You can retrieve and edit a local copy of the CI/CD pipeline source code from AWS CodeCommit using Git, an open source software version management system. You must create appropriate credentials in your AWS account to enable connecting to AWS CodeCommit repositories. For information about setting up Git, see Setting Up Using Git Credentials in the *AWS CodeCommit User Guide*. After you set up Git, you can use the following command to clone the CI/CD pipeline repository.

```
git clone <repository-url>
```

This command creates a local copy of the source code in your environment. You only need to clone a repository once. If you have already cloned the repository, it is recommended that you use the following command to pull updates from AWS CodeCommit prior to making any code changes.

```
cd <path/to/code>
git pull origin
```

Create a CodeBuild Project

AWS CodeBuild projects that build container images are defined in the `template_cfn.yml` AWS CloudFormation template found in the CodeCommit `GenomicsWorkflowPipe` repository. The following code example shows the structure of the definitions.

```
   StackBuildContainer<toolname>:
     Type: "AWS::CloudFormation::Stack"
     Properties:
       Parameters:
         Project: !Ref ProjectLowerCase
         ImageName: <toolname>
         ImageTag: "<tool-version>"
         BuildSpec: ./containers/buildspec.yml
         ProjectPath: ./containers/<toolname>
         CodeBuildRoleArn: !GetAtt CodeBuildRole.Arn
         UseProjectPrefix: "yes"
       TemplateURL:
         Fn::Sub:
           - ${TemplateRootUrl}/container-buildproject.cfn.yaml
           - TemplateRootUrl:
               Fn::Sub:
                 -
 "https://${ZoneBucket}.s3.${AWS::Region}.amazonaws.com/zone"
                 - ZoneBucket:
                     Fn::ImportValue:
                       !Sub ${ZoneStackName}-ZoneBucket
```

Modify the following variables for use with your tools:

- **Logical-ID:** A unique identifier within the AWS CloudFormation template. The naming convention used in the example project is `StackBuildContainer<toolname>`, where `<toolname>` is replaced with the name of the tool used to customize this solution (for example, `StackBuildContainerBwa)`.
- **ImageName:** The name used for the resultant Amazon ECR image repository (for example, `ImageName: bwa`).
- **ImageTag:** A tag passed to the `ARG VERSION` parameter in the image's Dockerfile. This tag is also used to tag the final container image (the version of the tool that is being containerized).
- **ProjectPath:** The relative path in the source code to find the Dockerfile used to create the tool's base image (for example, `ProjectPath: ./containers/bwa`).

Create a CodePipeline Stage Action

The AWS CodeBuild projects are invoked using an AWS CodePipeline pipeline defined in the AWS CloudFormation template. AWS CodeBuild projects are called in the *build* stage of the pipeline. The following code example shows container image build actions.

```
    CodePipeline:
      Type: AWS::CodePipeline::Pipeline
      Properties:
        # ...
        Stages:
          # ...
          - Name: Build
            Actions:
              # ...
              - Name: <toolname>
                ActionTypeId:
                  Category: Build
                  Owner: AWS
                  Provider: CodeBuild
                  Version: "1"
                Configuration:
                  ProjectName: !GetAtt
  "StackBuildContainer<toolname>.Outputs.Name"
                InputArtifacts:
                - Name: SourceStageOutput
```

Modify the following parts of the pipeline stage action for new tools:

- **Name:** The name used for this AWS CodePipeline must match the name of the tool you are building (for example, `- Name: Bwa`).
- **Configuration.ProjectName:** The project name must correspond to the AWS CodeBuild project for the tool. The naming convention is `StackBuildContainer<toolname>.Outputs.Name` (for example, `StackBuildContainerBwa.Outputs.Name`).

Commit and Push Changes to the CodeCommit Repository

Commit and push changes to the CodeCommit `GenomicsWorkflowPipe` repository to trigger a CI/CD pipeline update. Use the following Git commands to commit code changes to your local repository.

```
git add <file1> <file2> ...
git commit
```

An editor opens. Enter a commit message to describe the changes you made. After you finish committing your local code changes, you can use the following command to push the changes to the remote repository on AWS CodeCommit.

```
git push origin
```

This command triggers the execution of the AWS CodePipeline pipeline associated with the `GenomicsWorkflowPipe` repository and subsequent redeployment of the CI/CD pipeline.

Update the Workflow Code

An AWS Step Functions state machine implements a simple genomics variant calling workflow using BWA-MEM, SAMtools, and BCFtools. The workflow aligns raw FASTQ files to a reference sequence, and call variants on a specified list of chromosomes using dynamic parallelism. Use following process to update the workflow code.

Clone Source Code or Pull Source Code Updates

You can retrieve and edit a local copy of the secondary analysis workflow source code from AWS CodeCommit using Git. As discussed in the *Update the CI/CD Pipeline* section of this document, ensure that you have the appropriate credentials to enable connecting to AWS CodeCommit repositories. After you have the appropriate credentials, use the following command to clone the secondary analysis workflow repository.

```
git clone <repository-url>
```

This command creates a local copy of the source code in your environment. You only need to clone a repository once. If you have already cloned the repository, it is recommended that you use the following command to pull updates from AWS CodeCommit prior to making any code changes.

```
cd <path/to/code>
git pull origin
```

Create a Dockerfile

The solution includes example Dockerfiles for the tools it deploys in the example workflow. Container Dockerfiles are located in `/containers/<toolname>` folders in the CodeCommit `GenomicsWorkflowCode` repository.

Each Dockerfile specifies a multi-stage container build with, at minimum, a build stage. Dockerfiles also define an **ARG** parameter called `VERSION` used to specify which version of the tool to build. This value is used to tag the container image. The following is an example of a minimal Dockerfile.

```
FROM ubuntu:18.04 AS build
ARG VERSION=1.2.3

# build steps ...
```

This example minimal Dockerfile defines the base image used to build the container. This example assumes that base images are based on Ubuntu. Unless you have special requirements, this should be the only Dockerfile you need to create for each new tool you add.

Prior to pushing to Amazon Elastic Container Registry (Amazon ECR), a final container image is built by combining the base image with a custom entry point script that handles data staging from and to Amazon Simple Storage Service (Amazon S3). This container image uses specific environment variables to identify AWS Batch and AWS Step Functions. For more information, see Defining Workflow Steps.

Creating an AWS Batch Job Definition

Workflows reference AWS Batch job definitions when submitting jobs to AWS Batch for execution. These definitions are created in the `template_cfn.yml` AWS CloudFormation template in the `GenomicsWorkflowCode` CodeCommit repository as shown in the following code example.

```
  BatchJobDefinition<toolname>:
    Type: "AWS::Batch::JobDefinition"
    Properties:
      JobDefinitionName: !Sub ${ProjectLowerCase}-<toolname>
      Type: container #required
      ContainerProperties:
        Image: !Sub
${AWS::AccountId}.dkr.ecr.${AWS::Region}.amazonaws.com/${ProjectLowerCase
}-<toolname>
        Vcpus: 8
        Memory: 16000
        Volumes:
          - Host:
              SourcePath: /opt/miniconda
            Name: awscli
        MountPoints:
          - ContainerPath: /opt/miniconda
            SourceVolume: awscli
```

Modify the following parts of the job definition for use with your tools:

- **Logical-ID:** This ID must be unique within the AWS CloudFormation template. The naming convention used is `BatchJobDefinition<toolname>` (for example, `BatchJobDefinitionBwa`).
- **JobDefinitionName:** The name of the job definition that is unique within the AWS Region where the solution is deployed. The project name is used as a namespace. If the job definition already exists, a new job definition revision is created (for example, `JobDefinitionName: !Sub ${ProjectLowerCase}-bwa`).
- **ContainerProperties.Image:** The URI for the container image used when submitting jobs with this job definition. This should correspond to the URI for the container image created by the AWS CodeBuild project for the tool (for example, `${AWS::AccountId}.dkr.ecr.${AWS::Region}.amazonaws.com/${ProjectLowerCase}-bwa`).

Commit and Push Changes to the CodeCommit Repository

Committing and pushing changes to the CodeCommit `GenomicsWorkflowCode` repository triggers a secondary analysis workflow update. Use the following Git commands to commit code changes to your local repository.

```
git add <file1> <file2>...
git commit
```

An editor opens. Enter a commit message to describe the changes you made. After you commit your local code changes, you can use the following command to push the changes to the remote repository on AWS CodeCommit.

```
git push origin
```

This command triggers the execution of the AWS CodePipeline pipeline associated with the `GenomicsWorkflowPipe` repository and subsequent redeployment of the secondary analysis workflow.

Using APN Partner, AWS Marketplace, and Third-Party Tools

APN Partners offer several containerized bioinformatics tools available in the AWS Marketplace, or as third-party (for example, open source) solutions that can be used in genomics secondary analysis workflows. The following examples highlight how such tools might integrate with the solution.

> **Note:** Integration of any of these tools with your workflow depends on the distribution and licensing mechanisms for the tool you choose to use.

Example: Sentieon

Sentieon provides complete solutions for secondary DNA analysis that improve upon BWA, GATK, HaplotypeCaller, Mutect, and Mutect2 based pipelines. The solutions are deployable on any generic-CPU-based computing system. AWS compatible solutions are provided as Docker container images available at DockerHub. To build a container image using Sentieon with AWS Step Functions and AWS Batch for your workflow, create a new Dockerfile with the following code (see Create a Dockerfile for the location of the Dockerfiles).

```
FROM sentieon/sentieon-aws AS build
```

The container image automatically obtains a two-week evaluation license when run on Amazon Web Services. The base image configures environmental variables with an `ENTRYPOINT` script at `/opt/sentieon/cloud_auth.sh`. The solution's build and deployment pipeline override the `ENTRYPOINT` with an entry point that enables data staging from Amazon S3. Command overrides to the container should source the `cloud_auth.sh` script so that your environment is properly configured prior to running any Sentieon tooling. For example:

```
source /opt/sentieon/cloud_auth.sh; <toolname> <tool-arguments> ...
```

Example: BioContainers

BioContainers is a community-driven project that provides the infrastructure and basic guidelines to create, manage, and distribute bioinformatics packages and containers. BioContainers is based on the popular frameworks Conda, Docker, and Singularity.

Note: This solution currently only supports Docker containers.

There are several pre-built BioContainers container images for popular bioinformatics tools available on DockerHub. Use the following code to create a new Dockerfile and build a container image for a BioContainers tool that can be used with AWS Step Functions and AWS Batch.

```
FROM biocontainers/<toolname>:<tool-version> AS build
USER root
```

BioContainers container images use a non-privileged user by default. The default container user must be changed to `root`. When containers are built using the Genomics Secondary Analysis Using AWS Step Functions and AWS Batch solution, support scripts are added to locations within the container that require root access (for example, `/opt`, and `/usr/bin`).

To add Sentieon, BioContainers, or any other tool to the solution, see Update the Workflow Code.

Adding New Bioinformatics Workflows

This solution installs an example workflow for initial testing and to serve as a template to build complex workflows. The workflow definition is written in Amazon States Language (ASL), which is embedded inline within the AWS CloudFormation template that deploys it as an AWS Step Functions state machine. For modularity, the workflow definition template is a nested template. There is a 1:1 relationship between workflows and nested definition templates.

Workflow definition templates are located in `GenomicsWorkflowCode/cfn/workflow-*.cfn.yaml` and have the following structure.

- Parameters:
  - **Input Data S3 URI:** The default location for source data for the workflow, used to construct example state machine input.
  - **Output Data S3 URI:** The default location for output data for the workflow, used to construct example state machine input.
  - **Batch Job Definition ARNs:** Used by the workflow.
  - **Batch Job Queue ARN:** Used by default for the workflow.
  - **State Machine Execution Role ARN**: State machine resource definition with inline ASL.
- Outputs: **State Machine Name**, **State Machine ARN**, and **Example State Machine Input**.

After creating a workflow definition template, reference it in `GenomicsWorkflowCode/template_cfn.yaml` with the resource definition shown in the following code example.

```
Workflow<workflowname>:
  Type: "AWS::CloudFormation::Stack"
  Properties:
    Parameters:
      InputDataPrefix: !Sub s3://${SamplesBucket}/${SamplesPrefix}
      OutputDataPrefix: !Sub s3://${JobResultsBucket}
      BatchJobDefinitionBwa: !Ref BatchJobDefinitionBwa
      BatchJobDefinitionSamtools: !Ref BatchJobDefinitionSamtools
      BatchJobDefinitionBcftools: !Ref BatchJobDefinitionBcftools
      BatchJobQueue: !Ref LowPriorityQueue
```

```
          StatesExecutionRoleArn: !GetAtt StatesExecutionRole.Arn
        TemplateURL:
          Fn::Sub:
            - "${TemplateRootUrl}/workflow-variantcalling-bwa.cfn.yaml"
            - TemplateRootUrl:
                Fn::Sub:
                  -
"https://${ZoneBucket}.s3.${AWS::Region}.amazonaws.com/code"
                  - ZoneBucket:
                      Fn::ImportValue:
                        !Sub ${ZoneStackName}-ZoneBucket
```

Modify the following parts of the resource definition:

- **Logical-ID:** This ID must be unique within the AWS CloudFormation template. The naming convention used is `Workflow<workflowname>` (for example, `WorkflowBwa`).
- **BatchJobDefinitions:** This parameter references batch job definition resources created elsewhere by the AWS CloudFormation template.
- **TemplateURL:** This parameter references the name of the workflow definition template.

Defining Workflow Steps

Each step in the workflow executing a containerized job is an ASL task that uses the AWS Batch integration for AWS Step Functions. The following is an example of a state machine task for running `bwa-mem`.

```
"BwaMem": {
    "Type": "Task",
    "InputPath": "$",
    "ResultPath": "$.result",
    "Resource": "arn:aws:states:::batch:submitJob.sync",
    "Parameters": {
        "JobName": "bwa-mem",
        "JobDefinition.$": "$.jobdefs.bwa",
        "JobQueue.$": "$.params.queue",
        "ContainerOverrides": {
            "Vcpus": 8,
            "Memory": 8000,
            "Environment": [
                {"Name": "JOB_WORKFLOW_NAME", "Value.$":
"$.workflow.name"},
                {"Name": "JOB_WORKFLOW_EXECUTION_ID", "Value.$":
"$.workflow.execution"},
                {"Name": "JOB_INPUTS", "Value": "s3://broad-
references/hg38/v0/Homo_sapiens_assembly38.fasta*
${!SOURCE_DATA_PREFIX}/${!SAMPLE_ID}*"},
                {"Name": "JOB_OUTPUTS", "Value": "*.sam"},
                {"Name": "JOB_OUTPUT_PREFIX", "Value.$":
"$.params.environment.JOB_OUTPUT_PREFIX"},
                {"Name": "JOB_AWS_CLI_PATH", "Value.$":
"$.params.environment.JOB_AWS_CLI_PATH"},
                {"Name": "SOURCE_DATA_PREFIX", "Value.$":
"$.params.environment.SOURCE_DATA_PREFIX"},
                {"Name": "SAMPLE_ID", "Value.$":
"$.params.environment.SAMPLE_ID"},
                {"Name": "REFERENCE_NAME", "Value.$":
"$.params.environment.REFERENCE_NAME"}
            ],
            "Command": [
                "bwa mem -t 8 -p -o ${!SAMPLE_ID}.sam
${!REFERENCE_NAME}.fasta ${!SAMPLE_ID}_*1*.fastq.gz"
            ]
        }
    },
    "Next": "SamtoolsSort"
}
```

The following parameters reference AWS Batch resources:

- **Parmeters.JobName:** The name used for the submitted AWS Batch job to track the job's status in the AWS Batch console.
- **Parameters.JobDefinition:** The batch job definition ARN can either be a literal value or a reference to a JSON path in the input given to the state machine upon execution.
- **Parameters.JobQueue:** The batch job queue where the task is submitted.

The following variables are defined in `Parameters.ContainerOverrides` and control the task run definitions:

- **Vcpus:** The number of vCPUs allocated to the job container.
- **Memory:** The amount of RAM—in MB—allocated to the job container.
- **Environment:** The environment variables passed to the job container.
- **Command:** The command that the job executes within the job container.

Key environment variables:

- **JOB_WORKFLOW_NAME:** Name of the parent workflow for the job. Used with `JOB_WORKFLOW_EXECUTION_ID` to generate a unique prefix for workflow outputs.

- **JOB_WORKFLOW_EXECUTION_ID:** Unique identifier for the current workflow run. Used with `JOB_WORKFLOW_NAME` to generate a unique prefix for workflow outputs.

- **JOB_AWS_CLI_PATH:** Path to add to the `PATH` environment variable so that the AWS Command Line Interface (AWS CLI) can be located.

- **JOB_INPUTS:** A space-delimited list of Amazon S3 object URLs. For example, `s3://<bucket-name>/<key>` for files that the job will use as inputs.

> **Note:** For initial steps of a workflow, these S3 URLs can point to any existing datasets you have in your AWS Account, or publicly available datasets such as those found in the AWS Registry of Open Data.

- **JOB_INPUT_PREFIX:** The Amazon S3 location (for example, `s3://<bucket-name>/<prefix>`) where job inputs are stored. Specified in later steps of a workflow as: `${JOB_OUTPUT_PREFIX}/${JOB_WORKFLOW_NAME}/${JOB_WORKFLOW_EXECUTION_ID}`.

> **Note:** The <prefix> refers to the Amazon S3 prefix used by the solution's workflows to write intermediate data. The first step of a workflow uses the S3 URI for source data. All other steps in a workflow use the pattern provided in the example—`JOB_OUTPUT_PREFIX/JOB_WORKFLOW_NAME/JOB_WORKFLOW_EXECUTION_ID`. Each execution of a workflow must be isolated and must not overwrite data. This is especially important in scenarios that must adhere to HIPAA and 21CFR11 compliance.

- **JOB_OUTPUTS:** A space-delimited list of files that the job generates that are retained and transferred back to Amazon S3.

- **JOB_OUTPUT_PREFIX:** Amazon S3 location (for example, `s3://<bucket-name>/<prefix>`) were job outputs are stored.

> **Note:** By default, workflows created by this solution write intermediate and final results (along with the workflow name and execution ID) to the `JobResults` S3

bucket as described by the Genomics Secondary Analysis Using AWS Step Functions and AWS Batch Implementation Guide.

Adding New Dashboards

This solution provides an example Amazon CloudWatch dashboard to monitor the operational metrics of the example workflow. This dashboard is defined as a generic nested template—`GenomicsWorkflowCode/cfn/cloudwatch-dashboard.cfn.yaml`—and can be associated with any workflow. This dashboard requires that the state machine ARN is monitored as an input parameter.

If you add a new workflow and need to include a dashboard, a resource definition must be added to the `GenomicsWorkflowcode/template_cfn.yaml` file as shown in the following code example.

```
   Dashboard<workflowname>:
     Type: "AWS::CloudFormation::Stack"
     Properties:
       Parameters:
         StateMachineArn: !GetAtt
 Workflow<workflowname>.Outputs.WorkflowArn
       TemplateURL:
         Fn::Sub:
           - "${TemplateRootUrl}/cloudwatch-dashboard.cfn.yaml"
           - TemplateRootUrl:
               Fn::Sub:
                 -
 "https://${ZoneBucket}.s3.${AWS::Region}.amazonaws.com/code"
                 - ZoneBucket:
                     Fn::ImportValue:
                       !Sub ${ZoneStackName}-ZoneBucket
```

Modify the following resource definition parameters:

- **Logical-ID:** A unique ID within the AWS CloudFormation template. The naming convention is `Dashboard<workflowname>` (for example, `DashboardBwa`).
- **StateMachineArn:** The ARN of the state machine that the dashboard is associated with `Workflow<workflowname>` (for example, `WorkflowBwa`).

Document Revisions

Date	Change
April 2020	Initial release