# SV-Comp Cloud Track Instructions

The goal of the SV-Comp Cloud Track infrastructure is to make it straightforward to build solvers that will be used in the SV-Comp competition, and to run them at scale on Amazon Web Services (AWS) resources.

Preparing tools and testing them involves four phases, which will be described in greater detail later on in the document:

1. Creation of a test AWS Account for building your solver and running limited-scale experiments.
2. Creating the build pipeline to build your solver
3. Creating the AWS Batch execution pipeline to run your solver at limited scale in your test account.
4. Sharing your tools with us to run tests at scale

We will pay for the first $200 in AWS account fees per month, which should be more than enough to run several dozen builds and also to run tests at limited scale.

For each step, we have a CloudFormation template that can be used to set up the account for use with minimal effort. These files are available in the github project at:

<https://www.github.com/mww-aws/configure> (which also contains this README file). CloudFormation is AWS’s approach to “infrastructure as code”, and it allows bulk creation of AWS resources using a relatively straightforward (if verbose) YAML format to construct resources. It is not necessary that teams become expert CloudFormation users, but some understanding of the rudiments of CloudFormation are useful. See: <https://aws.amazon.com/cloudformation/> for information and a tutorial on CloudFormation. At each stage, we will tell you how to invoke CloudFormation to install resources into the account.

Parallel solvers are constructed by running multiple copies of a single Docker image that can communicate with one another using IP, TCP, SSH, or any number of higher-level protocols such as MPI. We provide a Docker container image that by default has support for the use of MPIover SSH.

The containers are hosted on AWS resources using a service called [AWS Batch](https://docs.aws.amazon.com/batch/latest/userguide/what-is-batch.html), where solver jobs run as multi-node jobs: <https://docs.aws.amazon.com/batch/latest/userguide/multi-node-parallel-jobs.html>. We describe the process to build the docker image and execute it on AWS Batch in the next two sections of this document.

When solvers are stable, we will use the scripts and github information to build the solver and run tests at scale. This will allow us to report results to you as to how well the solver is performing on a set of test benchmarks when running at scale.

## Creating the Test Account

Please create a “fresh” account in order to simplify billing for the account. If you have not created an AWS previously, it is straightforward to do, requiring a cell phone #, credit card, and address. Please navigate to aws.amazon.com and follow the instructions on the web site to create an account.

If you have already created an account based on your email address, please create a separate AWS account for managing the SV-Comp tool construction and testing. This makes it straightforward for us to manage account credits and billing. **Once the account is created please email us the account number at sv-comp-2020@amazon.com** so that we can apply credits to your account.

**N.B.:** It is veryimportant that you tell us your account number immediately after creating the account, so that we can assign you a resource budget for your experiments. Once we hear from you, we will email you in acknowledgement that the accounts have been set up with resources.

### Installing the AWS CLI.

In order to work with AWS, you must install the AWS CLI for your platform.

To use the AWS CLI, please follow the directions for your operating system here:   
 <https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html>

When setting up an account, we recommend the use of named profiles as they allow some flexibility in later connecting to multiple accounts:   
 <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html>

The rest of this document assumes that you have configured the CLI with named profiles.

After installing the AWS CLI and gaining credentials, make sure that the CLI is installed properly by attempting to run an AWS command. An example command that should work is:

aws --profile [YOUR PROFILE NAME HERE] s3api list-buckets

If it does not work, see the troubleshooting section at the bottom of this document.

Now, we will use CloudFormation to do basic account setup. The account-setup.yaml script sets up notification emails to be sent to the list of email accounts when the account reaches 20%, 40%, 60%, 80%, and 100% of the monthly account budget so that you have a window into the current spend rate for building and testing your solver. Here is the command to run it:

aws --profile [YOUR PROFILE NAME HERE] cloudformation create-stack --stack-name "setup-account-stack" --template-body file://setup-account.yaml --parameters ParameterKey=emailAddress,ParameterValue=[ENTER EMAIL ADDRESS HERE]

The --profile argument should be the profile associated with the account, and the emailAddress parameter is the email address that notification messages related to budgeting and account spending will be sent.

After running the aws cloudformation command, you can monitor the installation process from the CloudFormation console. Log into your AWS account and navigate to the CloudFormation console. You should see a stack named “setup-account-stack”.

[GRAPHIC HERE]

By clicking on this stack, and choosing “events”, you can see the resources associated with the stack. After a short time, you should see the “CREATE\_SUCCEEDED” event. If not (e.g., the email address was not valid email address syntax), you will see a “CREATE\_FAILED” event. In this case, delete the stack and try again. If you have trouble, please email us at: [sv-comp-2020@amazon.com](mailto:sv-comp-2020@amazon.com) and we will walk you through the process.

Although it is handy to get emails when certain account budget thresholds have been met, it is also possible to check by-the-minute account spending on the console: <https://console.aws.amazon.com/billing/home>

## Building the Solver

Solvers should be buildable from source code using a standard process. It is expected that you will provide (either via github or using a .zip file) a directory tree with a “build.sh” file at the top level that can build the project from source files that are contained within the repository structure. It is also expected that a Dockerfile is present at the top-level of the directory tree to construct the Docker image. You can also build the project inside the Dockerfile, in which case you can simply leave your build.sh blank. We have created a Dockerfile that contains MPI, SSH, and other tools and utilities that should be sufficient for competition purposes [HERE]. Copy this file into the top-level directory for your project.

The outcome of that build process should be a binary file that is stored in the /bin directory underneath the container root. Files in the /bin directory will be copied into the /bin directory of the created Docker container (using the provided Dockerfile).



*Figure 1: Overview of the Build Process*

An overview of the build process is shown in Figure 1. The build mechanism creates a docker container from provided source files and stores it in the Amazon ECR (Elastic Container Repository) service.

We support two different mechanisms for build. First, you can set up an external github account containing the source code and the pipeline will automatically build a new version of the solver each time a commit is made. Alternately, you can upload a directory structure as a .zip file into S3, and set up the build system to build each time a new .zip file is loaded. **N.B.: You must choose one or the other approach. It is not possible to have a pipeline that builds both from GitHub and from S3.**

### Building Sources from GitHub

To build sources from GitHub, a GitHub OAuth token is required. [MORE HERE ABOUT GETTING THE TOKEN].

This OAuth token allows CodePipeline to query the state of the GitHub repository and to rebuild the solver when changes occur.

We have a shell script that, for Mac OS and linux, can create the pipeline. It requires 6 arguments (sorry!) and creates the architecture shown in Figure 1. Here is the syntax for the command:

**build-solver-pipeline-github.sh PROFILE PROJECT\_NAME GITHUB\_TOKEN REPOSITORY\_OWNER REPOSITORY\_NAME REPOSITORY\_BRANCH\_NAME  
where:**

**PROFILE is a AWS CLI profile with administrator access to the account**

**PROJECT\_NAME is the name of the project. MUST BE ALL LOWERCASE.**

**Regular expression for names is:**

**(?:[a-z0-9]+(?:[.\_-][a-z0-9]+)\*/)\*[a-z0-9]+(?:[.\_-][a-z0-9]+)\***

**GITHUB\_TOKEN is a GITHUB OAuth token that can clone public repositories**

**REPOSITORY\_OWNER is the owner of the repository**

**REPOSITORY\_NAME is the name of the repository**

**REPOSITORY\_BRANCH\_NAME is the branch of the repository to use**

**N.B.: The Cloudformation script will fail if the “PROJECT\_NAME” parameter is not a lower-case string that follows the regular expression shown in the command arguments above.**

After running the shell command, you can monitor the progress of the CloudFormation account by logging into the CloudFormation console (as described above for the account setup script). The stack name will be **build-PROJECT\_NAME**, where PROJECT\_NAME is the name of the project that you used in the shell script.

If errors occur, please delete the stack (as described above for the account setup script), determine the error in parameters (e.g., did you use lower-case for the PROJECT\_NAME?) and re-run the build-solver-pipeline-github.sh script. If you encounter difficulties, please email us at: [sv-comp-2020@amazon.com](mailto:sv-comp-2020@amazon.com) and we will walk you through the process.

### Building Sources from S3

To build sources from S3, the script sets up an S3 “bucket” where you store a file named [PROJECT\_NAME].zip. Whenever a new version of this file is uploaded, the build system will attempt to re-build the docker container.

We have a shell script that, for Mac OS and linux, can create the pipeline. It requires 2 arguments and creates the architecture shown in Figure 1. Here is the syntax for the command:

**build-solver-pipeline-s3.sh PROFILE PROJECT\_NAME  
where:   
 PROFILE is a AWS CLI profile with administrator access to the account**

**PROJECT\_NAME is the name of the project. MUST BE ALL LOWERCASE.**

**Regular expression for names is:**

**(?:[a-z0-9]+(?:[.\_-][a-z0-9]+)\*/)\*[a-z0-9]+(?:[.\_-][a-z0-9]+)\*"**

**N.B.: The Cloudformation script will fail if the “PROJECT\_NAME” parameter is not a lower-case string that follows the regular expression shown in the command arguments above.**

After running the shell command, you can monitor the progress of the CloudFormation account by logging into the CloudFormation console (as described above for the account setup script). The stack name will be **build-PROJECT\_NAME**, where PROJECT\_NAME is the name of the project that you used in the shell script.

The script will build an S3 bucket named

**AWS\_REGION-AWS\_ACCOUNT\_ID-PROJECT\_NAME**

Where:

AWS\_REGION is the region in which the setup script was run.

AWS\_ACCOUNT\_ID is the account number of the account

PROJECT\_NAME is the name of the project that you created.

To trigger a build, upload a file called PROJECT\_NAME.zip (where PROJECT\_NAME is the name of the project you created) containing the project source code.

If errors occur, please delete the stack (as described above for the account setup script), determine the error in parameters (e.g., did you use lower-case for the PROJECT\_NAME?) and re-run the build-solver-pipeline-github.sh script. If you encounter difficulties, please email us at: [sv-comp-2020@amazon.com](mailto:sv-comp-2020@amazon.com) and we will walk you through the process.

### Debugging the Build Process

It is unlikely that the solver build will work the first time. Fortunately, CodePipeline and CodeBuild, the services that perform the build, have good logging and error reporting.

To get started, log in to your account and navigate to the CodePipeline console. You should see a pipeline that looks like the following:

[IMAGE HERE]

The two phases of the build process are the “Source” stage, where artifacts are read from S3 or from GitHub, and the “Build” stage, where the code is built and a docker container is stored in ECR.

**Debugging the Source Stage:**

For “Source”, if the build process was able to access GitHub (resp. S3), you should see a green check. If you see a red ‘X’ instead, there is some misconfiguration.

If building from GitHub, examine the error that is returned and carefully check that the OAuth token, account owner, project name, and branch correspond to the actual GitHub project. In this case, navigate to the CloudFormation console, delete the **build-PROJECT\_NAME** stack, and rebuild the stack using the correct github information.

If building from S3, it is unlikely to have a red ‘X’, but if the uploaded file name does not match **PROJECT\_NAME.zip** (where ‘PROJECT\_NAME’ is the name of your project) then the project will not build. If no builds have occurred, check to make sure that the uploaded file name and location is correct.

**Debugging the Build Stage:**

The build stage builds the tool Docker image using the build commands in a file called build.sh stored in the top level of the directory structure in the repo/zip file, and builds a Docker image using a file called Dockerfile at the top level of the directory structure. As previously discussed, we have provided a Dockerfile at: [DOCKER FILE LOCATION] that should be sufficient for the solvers that you produce. Include this Dockerfile at the top-level of your directory structure.

The container used by CodeBuild to build your tool contains recent versions of GCC, Java, Scala, Ruby, Python, and many standard build utilities for linux. When performing a build, it creates a log of the build activity. To debug the build process, please navigate to the CodeBuild service from the AWS Console. From there, it is possible to see success/failure of recent builds and logs of the build.

Successful builds will create a Docker container that is stored in ECR. If a build is successful, you should see a container image in ECR. To verify a build was successful, navigate to ECR and check to see whether a container was created.

If you encounter difficulties with the debugging process, please email us at: [sv-comp-2020@amazon.com](mailto:sv-comp-2020@amazon.com) and we will walk you through the process.

## Building the AWS Batch Pipeline

The next step is to build the Batch environment that will run the solver. This is relatively straightforward, and a script is provided to construct the batch environment.

The batch environment is designed to allow testing at small scale, and consists of four 16-core machines. This should be sufficiently large to allow testing that the communication between containers works properly and that parallel solving is working, but not so large as to become expensive for testing. For large-scale testing, we ask that, after the solver is stable, you provide us with links to the repository/S3, and we will run your solver at large-scale.

To set up the batch pipeline, run the job-queue.sh file:

**build-job-queue.sh PROFILE PROJECT\_NAME  
where:   
 PROFILE is a AWS CLI profile with administrator access to the account**

**PROJECT\_NAME is the name of the project. MUST BE ALL LOWERCASE.**

**Regular expression for names is:**

**(?:[a-z0-9]+(?:[.\_-][a-z0-9]+)\*/)\*[a-z0-9]+(?:[.\_-][a-z0-9]+)\*"**

PROJECT\_NAME must be the same name that you used earlier for the build-solver-pipeline script.

Once again, monitor the creation of resources from the script by navigating to the CloudFormation console.

## Running the Solver

To run the solver, we have to point it at a test directory and pass in arguments. The solver will run in AWS Batch as a multi-node parallel job: <https://docs.aws.amazon.com/batch/latest/userguide/multi-node-parallel-jobs.html>

This approach allocates resources for parallel execution, sets up a ‘main node’ with an IP address that is started first. Once executing, a set of ‘child nodes’ is started, and the IP address of the main node is passed to all child nodes, so that they can communicate back to the main node.

The standard way to pass in arguments for AWS Batch multi-node jobs is to use environment variables. We add additional environment variables describing the job to be solved and any additional flags to be passed to the solver.

### Environment Variables:

At runtime, in addition to the standard environment variables that all AWS Batch jobs receive, each node is configured with the following environment variables that are specific to multi-node parallel jobs:

AWS\_BATCH\_JOB\_MAIN\_NODE\_INDEX

This variable is set to the index number of the job's main node. Your application code can compare the AWS\_BATCH\_JOB\_MAIN\_NODE\_INDEX to the AWS\_BATCH\_JOB\_NODE\_INDEX on an individual node to determine if it is the main node.

AWS\_BATCH\_JOB\_MAIN\_NODE\_PRIVATE\_IPV4\_ADDRESS

This variable is only set in multi-node parallel job child nodes (it is not present on the main node). This variable is set to the private IPv4 address of the job's main node. Your child node's application code can use this address to communicate with the main node.

AWS\_BATCH\_JOB\_NODE\_INDEX

This variable is set to the node index number of the node. The node index begins at 0, and each node receives a unique index number. For example, a multi-node parallel job with 10 children has index values of 0-9.

AWS\_BATCH\_JOB\_NUM\_NODES

This variable is set to the number of nodes that you have requested for your multi-node parallel job.

We augment this list with data related to the solver and task to be performed:

COMP\_S3\_PROBLEM\_PATH

The location in S3 (bucket and path) of a .zip file containing the problem to be solved. This .zip file should be downloaded and expanded into the top-level directory of the container.

COMP\_S3\_RESULT\_PATH

The location in S3 (bucket and path) where the result of the analysis should be stored. The result (whether the property was violated or not) is competition-specific and final format is TBD.

COMP\_OPTIONS

A list of competition-specific options passed to the solver

COMP\_TASKS

A list of tasks to be performed by the solver

COMP\_RESOURCE\_LIMITS

A json-formatted string containing resource limits for the run. Examples: time-limit, soft-time-limit, hard-time-limit, memory-limit, cpu-core-limit

For each node in the multi-node job (i.e., each container), the process will be roughly:

1. Read the environment variables
2. Download the problem to be analyzed using S3 using COMP\_S3\_PROBLEM\_PATH
3. Determine whether this node is the ‘main’ node or ‘child’ node using environment variables: AWS\_BATCH\_JOB\_NODE\_INDEX and AWS\_BATCH\_JOB\_MAIN\_NODE\_INDEX
4. Child node sets up communications with main using the AWS\_BATCH\_JOB\_MAIN\_NODE\_PRIVATE\_IPV4\_ADDRESS
5. Main node distributes portions of the problem to children as they connect.
6. Child node works on their portion of the problem.
7. Once results are known, solving node notifies main node. One of these nodes writes to COMP\_S3\_RESULT\_PATH, and main node terminates.

When the main node terminates, the solving process is finished. If nodes complete normally, then the exit code for the container should be SUCCEEDED. The results of the properties should be written to the COMP\_S3\_RESULT\_PATH S3 location.

### Monitoring and Logging

The AWS Batch console allows you to observe the status of batch jobs. Each job goes through stages: SUBMITTED, RUNNABLE, STARTING, RUNNING, and either SUCCEEDED or FAILED. You can examine all jobs in each stage using the console application.

For a given job, each output to stdio/stderr by a node is logged. To examine the logs, choose the job, then navigate to the bottom of the information page, and there will be a link to view the logs.

## Example

An example of the

## Troubleshooting

TBD