

GSoC 2023 Final Project Report - Open Genome Informatics

Project Title: Improved Continuous Integration for Reactome

GSoC 2023 Contributor: Sukanya Krishna

Email: sskrishn@ucsd.edu

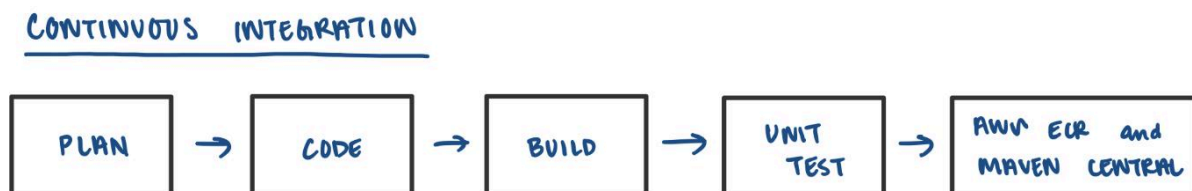
Github: sukikrishna

Background:

Jenkins is being used to run the Reactome release pipelines and this project proposes moving the system into a more automated format – to use Docker containers with a CI/CD-based system [1]. In the standard release process, as project code will be built and pushed to the main branch for each GitHub repository, as a result, multiple dependencies would also be triggered to be built as well. In this way, we will know if a change in a dependency has correctly integrated into the broader code base. There is a release pipeline process that can take on average 3 weeks in total to be released. This time frame should ideally be compressed, and the production should go through the multiple pipeline processes and pull in files from the internet that may or may not have changed, wanting to trigger subsequent changes automatically. Before making an official release, have a system that will run tests, check if they're done correctly, and then try to automate as much of this system as possible. Multiple repositories depend on one another, when a change has been made, need to perform tests of the singular feature, and then perform integration tests to check whether each feature is performing properly, to make releases of these changes. A lot of this testing is done manually, so automating this system can make the time frame for making a release much smaller. With a continuous integration system, we will integrate these artifacts with the release ETL (Extract Transform Load) pipelines.

Currently, Reactome is hosting the Java artifacts on a Nexus server they have internally maintained. They are currently in the process of moving to Maven Central. The system that I am proposing will primarily work with Maven Central as that is the direction they are moving in. Take part of the curation database and part that wants to be released to the public (for Reactome everything is curated to humans), and then run through the slice of the database and run through different testing files on how this should perform. These pipelines can be continuously integrated and continuously deployed when possible.

Proposed Work Flow Chart:



Part I: Continuous Integration (CI) system. Taking the changed code in the main branch on GitHub, there is some environment listing for that change on GitHub, which will build a jar file or docker container

around the jar file depending on the kind of features that need to be run. This process essentially is when making a change, it will continuously be making artifacts according to changes made to a repository.

CONTINUOUS DEPLOYMENT



Part 2: Continuous Deployment (CD) system. Dependent on the CI system. From the AWS ECR and Maven Central will eventually get pushed to S3, which stores flat files, or EKS which stores docker containers (stores the artifacts). This process essentially works on preparing the artifacts from CI to make them available (ready for deployment in a production environment).

RELEASE TO PRODUCTION



Part 3: Release to Production. From the CD system, sending all of the files to S3, and hosting them on the release.reactome.org folder structure of all the artifacts and files used in production. Production would ideally point at the files in S3. Some components on the production server may include but are not limited to the Neo4J database, Java APIs, Apache, and Solr.

Currently, there are automated tests along with manual tests. Although not all tests can be fully automated enough could be automated to significantly impact how the pipelines are run. Changes are made to the curation database when certain things are put in incorrectly. When all of these changes have been solved, then a final slice of the data is used for the whole release. Building software in a systematic way in an integrated system so developers can see their issues earlier before the release process can be a use case of this development system. One approach is to have the jar files dockerized and stored in AWS as the intermediate docker containers of each of the steps involved in the processes. For continuous integration software, Kubernetes can be used to switch between the docker containers and appropriate testing environments that can be run. There are continuous deployment packages built around Kubernetes UIs, for instance, GoCD, ArgoCD, AWS Fargate, and Jenkins X. As the current system is using Jenkins and Java, these UIs may be good considerations for integrating the continuous integration deployment process.

Goals of the Project:

The primary goal of the project was to establish a robust Continuous Integration and Continuous Deployment (CI/CD) pipeline for the Reactome organization. The pipeline was intended to integrate Argo CD, Argo Workflows, GitHub, and Jenkins to automate application deployment and achieve a streamlined development process.

Accomplishments:

During the project, progress was made towards achieving the project goals for integrating a continuous integration and continuous deployment (CI/CD) pipeline for the Reactome organization. The main focus was on implementing the continuous integration component of the pipeline. A dedicated effort was put into dockerizing existing repositories within the Reactome organization. This process involved creating a separate branch to work on dockerization, ensuring that it wouldn't disrupt the ongoing release cycle. A comprehensive spreadsheet listing all repositories in the Reactome organization was generated (linked below) documenting progress. Additionally, a clear process for dockerizing and integrating other repositories was defined for future work.

A major milestone was successfully establishing a fully functional CI pipeline for two repositories, with the potential to expand to approximately 20 repositories by the end of the month. The code for the two repositories, `statistics_generator` and `release_download_directory`, will be included for the upcoming release. The CI pipeline was designed to be triggered by GitHooks, responding to both pull requests and pushes to the main branch. This achievement marked a pivotal step in automating the development process and streamlining application deployment. The first repository, `statistics-generator`, newly implemented this process and demonstrated the ability to seamlessly integrate and automate the CI process from the ground up. The second repository, `release-download-directory`, was historically challenging to operate on the release server but was effectively tackled by dockerizing. This dockerization not only automated but also enhanced the work curators were doing to generate images.

Additionally, by dockerizing the `release-download-directory`, a critical segment of the pipeline, the project contributed to the elimination of manual intervention. This not only enhances pipeline efficiency but also significantly reduces the workload for developers, aligning with the project's overarching goal of improving automation and productivity.

The additional repositories will be on their way to being integrated into a CI pipeline and in the subsequent release if this release goes well. The goal was to not move all changes into this upcoming release, but to see how well these repositories perform for the upcoming release so in the subsequent one, the organization can pull more in. The timing of the release cycles are quarterly so it can give the opportunity to work through any bugs. The pull requests and github repositories that have been integrated in CI are linked below.

Additionally, an essential accomplishment was the successful setup of Argo CD, which marked a pivotal step toward the implementation of the complete CI/CD pipeline. Despite resource constraints, Argo CD was configured to facilitate automated application deployment and management. However, due to limitations in available resources, the utilization of this pipeline was temporarily paused, particularly in running the clusters on AWS.

This combination of dockerization progress and the establishment of Argo CD laid a strong foundation for the Reactome organization to transition towards a more streamlined development process with automated deployment capabilities.

Link to a spreadsheet with repositories for which commits have been created: [📄 Reactome Repositories](#)

Linking some of the repositories for which I created commits for:

- `graph_importer`: <https://github.com/reactome/graph-importer/tree/sk-gsoc>
- `data_export`: <https://github.com/reactome/data-export/tree/sk-gsoc>

Link to `statistics_generator`: <https://github.com/reactome/statistics-generator/tree/main>

Link to `release_download_directory`: <https://github.com/reactome/release-download-directory/tree/main>

Current Status of Continuous Integration and Deployment Pipeline:

As of the conclusion of the project, the continuous integration component of the CI/CD pipeline has been significantly improved. A total of 17 repositories have been dockerized and have potential to be integrated. While Argo CD has been successfully set up, the usage has been temporarily halted due to resource constraints in running clusters on AWS.

Future Tasks:

While progress has been made, there are remaining tasks and future steps that need to be addressed to complete the CI/CD pipeline. One crucial step is the full implementation of Argo CD, which was not fully accomplished during this project. This includes connecting Argo CD to AWS resources, establishing connections between Argo CD and Argo Workflows, and fine-tuning the deployment process. Further dockerization of repositories and integration into the pipeline is also required to achieve comprehensive coverage. Additionally, addressing resource constraints to resume utilization of the pipeline is an important next step.

Code Merged Upstream:

The code for dockerizing the 17 repositories has not yet been merged into the main repositories due to the ongoing release cycle. However, this code is readily available on a separate branch and can be incorporated into the main repositories in future releases. The setup of Argo CD and other related configurations are documented and available for reference.

Challenges and What I Have Learned:

Throughout the project, several challenges were encountered. One significant hurdle was the fact that this was the first experience with using Docker with Jenkins pipelines for the project. The learning curve associated with these tools added an extra layer of complexity to the project. Gaining proficiency in Docker's containerization and Jenkins' automation capabilities required dedicated effort and exploration.

Another challenge was effectively managing concurrent releases while implementing new CI/CD processes. Balancing the ongoing release cycle with the introduction of new automation processes required careful planning and consideration. The need to maintain a stable release environment while incorporating new changes highlighted the importance of robust branching and versioning strategies.

Despite these challenges, the project provided valuable lessons. First and foremost, it emphasized the significance of hands-on experience with emerging technologies and the ability to adapt to unfamiliar tools. It also underscored the importance of clear documentation and communication within the team, particularly when navigating complex processes involving multiple tools.

In retrospect, the experience gained from tackling these challenges and overcoming any learning curve has proven to be an invaluable asset. It has not only expanded my skill set but also deepened their understanding of CI/CD principles, Docker containerization, and Jenkins automation, which will be applicable in future projects.

Documentation for Framing Future Progress:

The documentation provided in this report serves as a guide for framing useful documentation and next steps for future work. Individuals aiming to extend the project can refer to the existing work on dockerization, connecting Argo CD and Argo Workflows to AWS, and configuring Jenkins for CI/CD.

In the future, addressing resource constraints and completing the implementation of Argo CD are recommended. Further dockerization efforts and the incorporation of additional repositories into the pipeline are also essential to achieve comprehensive coverage of the Reactome organization's efforts. If the continuous integration and deployment effort can continue, then that can allow OICR employees to focus their time on other research and development rather than on monitoring and testing scripts for future releases as this process will become more automated. Regularly updating and maintaining the documentation will be important as the project progresses.

Future goals:

1. Complete the full implementation of Argo CD, ensuring connections with AWS resources and integration with Argo Workflows.
2. Continue dockerizing repositories and integrating them into the CI/CD pipeline.
3. Fine-tune and optimize the pipeline for efficiency and reliability.
4. Regularly update and maintain the documentation to reflect the evolving state of the project.


The provided documentation and the work completed so far lay the foundation for a robust CI/CD pipeline that will streamline the development process and enhance collaboration within the Reactome organization.


[1] <http://www.ncbi.nlm.nih.gov/pubmed/34788843>

Notes regarding CI/CD Implementation using ArgoCD and Jenkins:

Documentation on Connecting Argo CD and Argo Workflows to AWS and Configure Jenkins and Argo CI-CD Pipeline


Goal: Want to be able to connect Argo CD and Argo Workflows to AWS, as well as integrate GitHub, Argo CD, and Jenkins for continuous integration and continuous deployment (CI/CD) of the application pipelines. Steps can be configured in different ways specific to project requirements.

1. Tool Installation and Familiarization
 - a. You need an AWS account with appropriate permissions to create resources.
 - b. Install Argo CD, Argo Workflows, and Kubernetes.
 - i. For Argo CD: https://argo-cd.readthedocs.io/en/stable/getting_started/
 - ii. For Argo Workflows: <https://argoproj.github.io/argo-workflows/quick-start/>
 1. <https://github.com/awslabs/data-on-eks/tree/main>
 - iii. For Kubernetes: <https://kubernetes.io/docs/tasks/tools/>
 - c. Familiarity with the basics of AWS, Argo CD, and Argo Workflows.
 - i. Some useful resources:
 1. AWS
 - a.  [AWS In 10 Minutes | AWS Tutorial For Beginners | AWS ...](#)
 - b. <https://cloudacademy.com/learning-paths/aws-fundamentals-1/>
 - c. <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>

- d. <https://www.simplilearn.com/tutorials/aws-tutorial/aws-ec2>
 2. Argo CD
 - a.  [ArgoCD Tutorial for Beginners | GitOps CD for Kubernetes](#)
 - b. <https://www.harness.io/blog/what-is-argo-cd#:~:text=Why%20Argo%20CD%3F,helps%20to%20remediate%20configuration%20drift.>
 - c. <https://successive.cloud/10-advantages-argocd-with-kubernetes/>
 - d. <https://www.nine.ch/en/engineering-logbook/what-is-argo-cd>
 3. Argo Workflows
 - a. [https://pipekit.io/blog/what-is-argo-workflows#:~:text=With%20Argo%20Workflows%2C%20you%20can,CD\)%2C%20and%20infrastructure%20automation.](https://pipekit.io/blog/what-is-argo-workflows#:~:text=With%20Argo%20Workflows%2C%20you%20can,CD)%2C%20and%20infrastructure%20automation.)
 - b. <https://codefresh.io/learn/argo-workflows/>
 - c. <https://www.getambassador.io/docs/argo/latest/concepts/argo>
 - d. <https://medium.com/atlantbh/implementing-ci-cd-pipeline-using-argo-workflows-and-argo-events-6417dd157566>
 4. Kubernetes
 - a. <https://kubernetes.io/docs/setup/>
 - b. <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>
 - c. <https://www.digitalocean.com/community/tutorials/how-to-deploy-to-kubernetes-using-argo-cd-and-gitops>
 - d. <https://codefresh.io/learn/argo-cd/>
2. Set up AWS resources
 - a. Create an Amazon Elastic Kubernetes Service (EKS) cluster using the AWS Management Console or the AWS CLI.
 - i. EKS cluster allows to run Kubernetes on both Amazon ECG and AWS Fargate: [https://aws.amazon.com/eks/features/#:~:text=Amazon%20EKS%20lets%20you%20run,Amazon%20EC2\)%20and%20AWS%20Fargate.](https://aws.amazon.com/eks/features/#:~:text=Amazon%20EKS%20lets%20you%20run,Amazon%20EC2)%20and%20AWS%20Fargate.)
 - ii. <https://docs.aws.amazon.com/eks/latest/userguide/create-cluster.html>
 - iii. Ensure that the EKS cluster has relevant permissions and IAM roles configured.
 - b. Deploy Argo CD and Argo Workflows resources on the EKS cluster (information should be provided in the installation guides).
 - i. Can deploy using Linus Screen: <https://linuxize.com/post/how-to-use-linux-screen/>
 - ii. Can connect Argo and Argo Workflows to deploy off of the same port
 1. AWS Route 53: <https://aws.amazon.com/route53/>
 2. <https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-to-ec2-instance.html>
3. Connect Argo CD to AWS
 - a. Resources:
 - i. <https://aws.amazon.com/blogs/opensource/connecting-aws-managed-services-to-your-argo-cd-pipeline-with-open-source-crossplane/>

- ii. <https://dev.to/awsccommunity-asean/ci-cd-with-argocd-on-aws-eks-cluster-3e09>
 - b. In the Argo CD interface, navigate to the Applications page and click "New Application."
 - c. Fill in the necessary details, such as the application name and the Git repository URL.
 - d. Under the "Project" section, select where to deploy the application.
 - e. In the "Sync Policy" section, choose the appropriate sync options based on your requirements.
 - f. Under the "Destination" section, select "Cluster" and provide the necessary details such as the Kubernetes cluster name and the namespace.
 - g. In the "Source" section, finish the application configurations and provide the necessary details.
 - h. Click "Create" to deploy the application onto the AWS cluster. Argo CD will automatically synchronize and deploy the application.
4. Connect Argo Workflows to AWS
- a. Follow the instructions here:
 - i. <https://awslabs.github.io/data-on-eks/docs/blueprints/job-schedulers/argo-workflows-eks>
 - ii. <https://github.com/awslabs/data-on-eks/tree/main>
 - b. Use the Argo Workflows CLI or the Argo Workflows UI to submit the workflow for execution.
 - c. Argo Workflows will create the necessary resources and interact with AWS services based on the workflow definition.

Connecting GitHub, Argo CD, and Jenkins for Continuous Integration and Continuous Deployment (CI/CD)

5. Requirements
- a. A GitHub repository containing your application code.
 - b. An Argo CD installation, Jenkins installation, and appropriate Kubernetes cluster.
 - i.  [How to Deploy To Kubernetes with Jenkins GitOps GitHub Pipeline - Tut...](#)
6. Connect Argo CD to GitHub
- a. Resources:
 - i. <https://levelup.gitconnected.com/connect-argocd-to-your-private-github-repository-493b1483c01e> ← Example of how to connect Argo to repository
 - b. Log into the Argo CD application
 - i. Username: admin
 - ii. Password: ... (can get first-time password off of command line)
 - c. In the Argo CD interface, navigate to "Settings" and then "Repositories."
 - d. Click "New Repository" and provide the necessary details such as the repository name and the repository URL.
 - e. Configure the repository settings
 - f. Argo CD will automatically sync with the configured GitHub repository and deploy any changes detected in the repository (after they have been pushed).

7. Configure Jenkins with Argo CD integration
 - a. Jenkins at this stage has already been configured for CICD because all of the build processes and steps have been built for Reactome repositories/pipelines. The goal is just to be able to connect Jenkins with Argo and run these pipelines.
 - b. Resources
 - i. <https://medium.com/@abhishek261291/cicd-using-jenkins-and-argocd-cb0b9fa63aa0>
 - ii. <https://cloudyuga.guru/blog/jenkins-argo>
 - c. In the Jenkins pipeline, add a stage or step to trigger the deployment in Argo CD.
 - d. Use the Argo CD CLI or Argo CD REST API to deploy the application.
 - e. Configure the appropriate Argo CD credentials or authentication to interact with the Argo CD server.
 - i. Can incorporate auth0 authentication:
https://www.google.com/aclk?sa=l&ai=DChcSEwj90sSCmJaAAxVXIa0GHcd1BE4YABACGgJwdg&sig=AOD64_2-O3O_vxH58N2Tajzs8U5OWRjL4A&q&a_durl&ved=2ahUKEwjP8buCmJaAAxUbIEQIHWHOcngQ0Qx6BAgLEAE
8. Test the CI/CD Pipeline
 - a. Make a code change in your GitHub repository and push the changes.
 - i. <https://github.com/joshnh/Git-Commands>
 - b. Observe the Jenkins pipeline being triggered automatically or manually.
 - i. You can see the entire pipeline and steps (e.g. jar creation, files being added or destroyed)
 - ii. Color-coded (green/red) to indicate if the process is continuing correctly

If the process is working, then ideally, Jenkins will build, test, and package the application, and then deploy it using Argo CD onto the Kubernetes cluster.