ASD2018, Jenkins Documentation

Contributors: Tianyi Cheng, Ali Alhady

Jenkins Build Server

The Jenkins server runs on asd1.ccs.neu.edu, port 8080 (link: http://129.10.111.209:8080/). This server resides behind the NU firewall and as such can only be reached when on the NU network. As of this writing, the administrator credentials for the Jenkins server are (user: admin/pass: admin) Jenkins itself is housed within a Docker Container, which provides many advantages including easy migration of components from one server to another if ever need be.

Deployment projects within the Jenkins server itself are split into multiple folders that correspond to the four main project components (Deployment, Frontend, Backend and Machine Learning). The component deployment projects are all further highlighted below in this document.

Deployment

This folder contains Jenkins maintenance and backup jobs. There are daily backup jobs implemented for both the Jenkins server itself and also database components.

- backupMariaDBFiles
  - Using the *MySQLDump* tool, takes a backup in .sql format of each database instance (student/public/admin) and stores them in a private repository

- jenkinsBackup
  - Compresses and pushes a backup of core files within jenkins_home and publishes to a private repository

- printRunningDockerContainers
  - *Prints to console output ALL running Docker containers on each of the server instances (asd1.., asd2.., asd3.., asd4.ccs.neu.edu)*

FrontEnd

This folder contains all deployment jobs for front end web interfaces (public, student & admin). Front end jobs will clone latest code from the provided scm repository and install any required dependencies through npm. Below is more granular information on the deployment for each of the frontend components:

- Admin
    - Utilizing npm, gets the latest **Admin** front end project code and any dependencies
    - Compresses project source to tar format for distribution
    - Copies the compressed file to the remote NU server host (in this case, **asd2**)
    - Checks for any existing Docker containers for the **Admin** front end and stops/removes if found
    - Deploys the admin front end project within a new httpd container and prints machine status to console
- Public
    - Utilizing npm, gets the latest **Public** front end project code and any dependencies
    - Compresses project source to tar format for distribution
    - Copies the compressed file to the remote NU server host (in this case, **asd4**)
    - Checks for any existing Docker containers for the **Public** front end and stops/removes if found
    - Deploys the **Public** front end project within a new httpd container and prints machine status to console
- Student
    - Using npm, gets the latest **Student** front end project and dependencies.
    - Compresses this to tar format for distribution
    - Copies the compressed archive to the remote NU server host (in this case, **asd4**)
    - Checks for any existing Docker containers for the **Student** front end and stops/removes if found
    - Deploys the **Student** front end project within a new httpd container and prints machine status to console

BackEnd

Database

The database system selected for the ASD project is MariaDB. A Jenkins project was implemented to deploy the three MariaDB instances (student, admin & public) to isolated Docker containers on asd3. This process is highlighted in more detail below:

- Database Deployment
    - The deployment job will first check to see if any database containers are running on the remote server (asd3)
        - If any jobs are found, they will be force stopped and removed

- A new set of mariadb containers are then spun up which house each database instance individually
- Databases live in their own containers and do not interact with each other
- The database always persists on the local file system and gets mounted to a Docker container as an external volume

Services

These projects utilize maven to resolve dependencies and for building the services infrastructure, there is a pre-deployment phase for each individual project which will run tests with a newly provisioned containizered test database on a Docker virtual network. If the tests pass, the projects will continue on with deployment of the web application package. The full deployment process per component is highlighted below:

- Admin
  - Post-completion of the services test deployment, the project will move forward with the production deploy portion
  - The **Admin** project .war file will be uploaded to the remote server host (asd2 in this case)
  - A check will be done to ensure there are no already deployed **Admin** services Docker containers
  - A new tomcat Docker container housing the **Admin** services will be spun up onto asd2
- Public
  - Post-completion of the services test deployment, the project will move forward with the production deploy portion
  - The **Public** project .war file will be uploaded to the remote server (asd4 in this case)
  - A check will be done to ensure there are no already deployed **Public** services Docker containers
  - A new tomcat Docker container housing the **Public** services will be spun up onto asd4
- Student
  - Post-completion of the services test deployment, the project will move forward with the production deploy portion
  - The **Student** project .war file will be uploaded to the remote server (asd4 in this case)
  - A check will be done to ensure there are no already deployed **Student** services Docker containers
  - A new tomcat Docker container housing the **Student** services will be spun up onto asd4

Machine Learning

The Machine Learning component of this project involves implementation of a ‘chatbot’ application which will dynamically answer questions around the ALIGN program. This tool leverages the Google-owned technology called Dialogflow which helps to facilitate more natural conversational behavior for the chatbot. Deployment of the core components for this utility are further described below:

- Chatbot Service
    - The latest available code for the chatbot service is cloned and then compressed into .tar format
    - The package is then deployed to the corresponding server host (asd4 in this case)
    - A search is done on asd4 for any existing chatbot Docker containers. If one is found, it is stopped and then removed
    - A new Docker container starts which will host the latest chatbot release deployment
        - The python flask service is hosted in a gunicorn app server
- Redis Database
    - This component involves both client and server endpoints that live within separate Docker containers
    - The client endpoint is the first sub-component that is built:
        - The latest available code is cloned to the build server
        - The build process is completed within a go Docker container
        - The built app is then deployed to the server host (asd2 in this case)
        - A Docker virtual network named redis-net will be created
        - A redis server container is then created within the redis-net network. Note that it’s only accessible within the redis-net network
        - A check is performed for any existing Docker containers hosting the redis client. If any are found, they are stopped and removed
        - A redis client container is created within redis-net network. This is accessible within the neu network

Design Philosophies

This project was built on Docker virtualization technology. Below are some additional customizations to our project.

- Docker is the only software installed on the servers(asd1-4). Linux Docker group was added.

    - A Jenkins account (user: jenkins/pass: jenkins) is set up on the servers under the Docker group with a home directory of '/data'. This account can communicate with the Docker daemon. All changes we have made to the file system on the servers(asd1-4) are restricted to '/data'.
    - The containers are configured to always start-up immediately if not manually stopped. This is enabled by using '--restart always' option when running Docker container. If the server host was to unexpectedly reboot, the Docker daemon will start on boot and all containers will come online.
    - The jenkins container on asd1 mounts a Docker unix socket so that the user (root user of the container) can talk to the Docker daemon outside the container.
    - Removal of Docker container is done by execute 'Docker rm -v -f <contain_name>'. The -v option ensures all temporary volume gets deleted. Otherwise, those dangling volumes will consume disk space(/var).

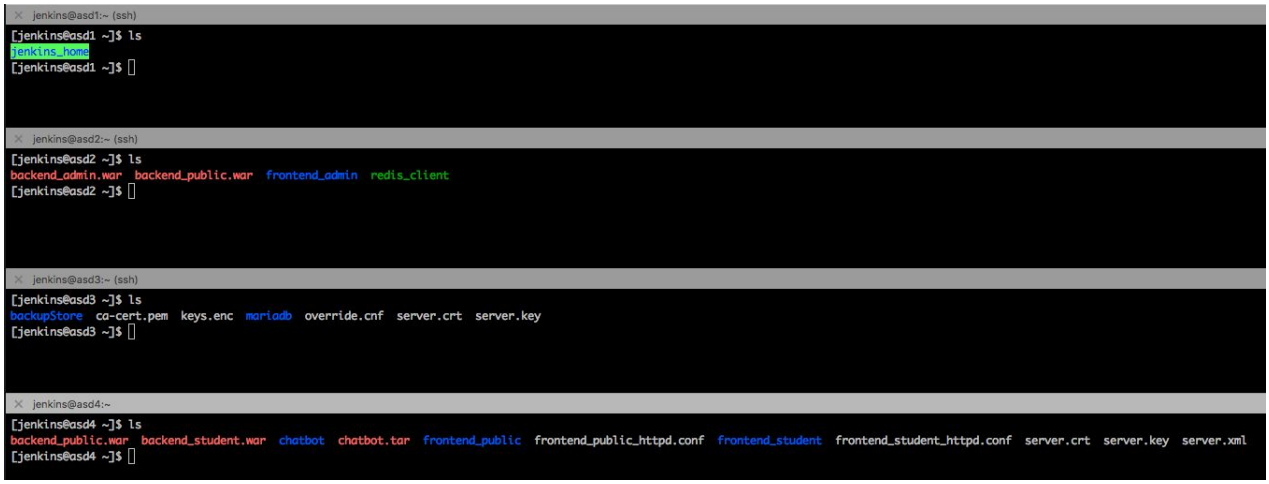
The CCIS signed certificate and private key resides on the corresponding host:

- key pair on asd1 for HTTPS
- key pair on asd4 for TCP SSL

The service config stays on the corresponding host.

- asd1: config for jenkins (jenkins_home)
- asd3 : config for mariadb
- asd4 : config for httpd, config for tomcat

Jenkins Home Directory



Deployed Containers