Overview of Jobs in Jenkins

What is Jenkins?
From the Jenkins home page:

> "Jenkins is an award-winning, cross-platform, **continuous integration and continuous delivery** application that increases your productivity. Use Jenkins to **build and test your software projects continuously** making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to **continuously deliver** your software by providing powerful ways to define your build pipelines and integrating with a large number of testing and deployment technologies."

Currently we are using Jenkins to automate the integration of the kuali-research application.
Jenkins is hosted on a dedicated AWS EC2 instance at the following URL:

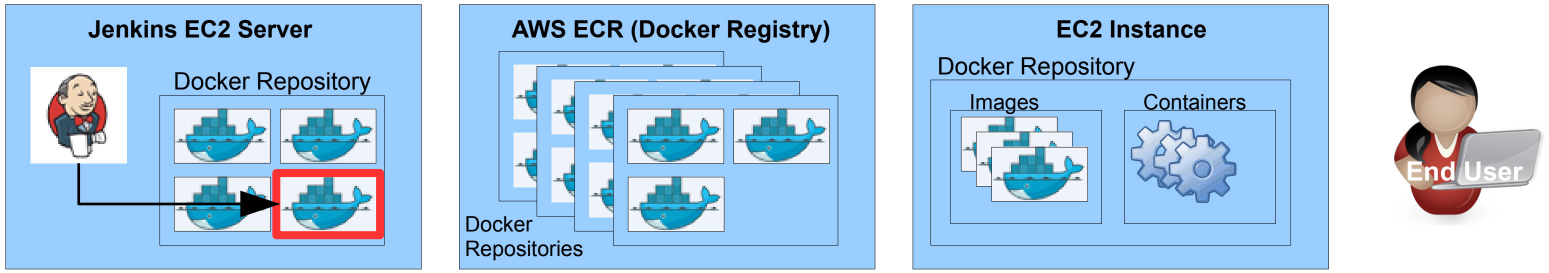
http://10.57.236.6:8080/

- This URL is only accessible from within the BU network (If at home go through the BU VPN)
- Obtain the login credentials from the site administrator.

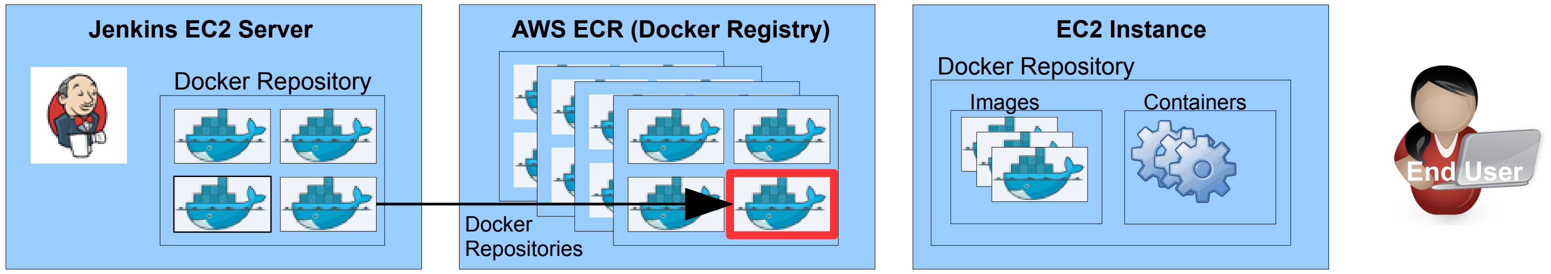
Basic job steps

Up to now, Jenkins is being used to deploy applications that run in docker containers which in turn are running on Amazon web services EC2 instances. Therefore, any Jenkins job will be involved as part of an overall deployment and each deployment will basically follow this pattern:

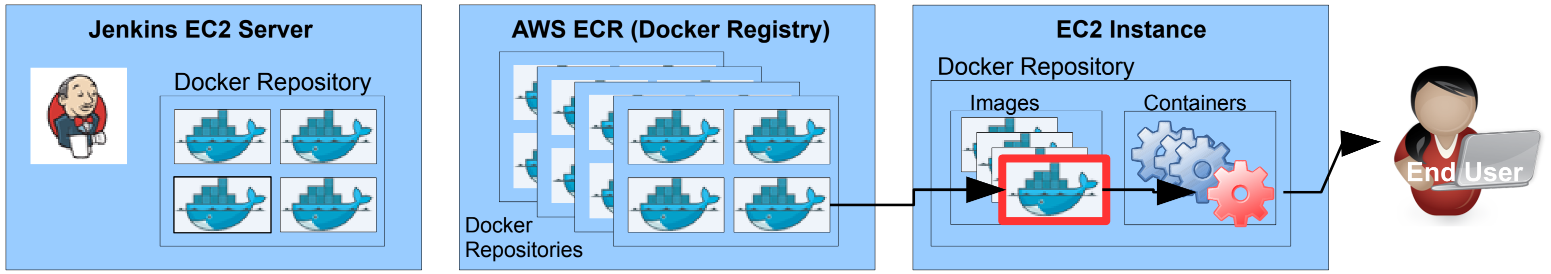
1) A Jenkins job will create a docker image locally in its own local docker repository. This image incorporates the application to run for end users.



2) A Jenkins job will load the new docker image to a more central location within the AWS cloud called the elastic container registry (ECR). This registry is basically a collection of docker repositories exposed publically for access by any and all docker repositories that can authenticate to it and gain access to its images for download.



3) A Jenkins job will shell into the target EC2 instance where the application is to run and call docker to run a new container. Docker is pointed to the network location for the image in the registry and is instructed to base the container on this remote image. To do this, it download the image to its own repository and runs the container from it.



Jobs By Tab

"Kuali-Research"

There are 5 jobs in this tab. While each can be run separately, the use case that will apply 99% of the time is that the "kuali-research" job will be triggered, which in turn calls the remaining 4 jobs in order:

1. "kuali-research-1-build-war"

While each of the remaining jobs can be run separately, the use case that will apply 99% of the time is that this job will be triggered, manually or by changes to the git codebase, and it in turn calls the remaining 4 jobs in order.
If you are using this build, your goal is to go through the "Basic Job Steps" as described in the previous section with respect to a particular "landscape" (sandbox, ci, qa, staging, production). That is, the EC2 instances that will be updated with new docker containers are those that comprise a particular landscape. Currently each landscape is comprised of 2 load balanced ec2 instances, except qa which runs the kuali research application on only one ec2 instance.
This job takes in 2 standard parameters:

a) The url for the git repo where the source code for building the kuali research war file resides.

b) A pick list for selecting what landscape we are building to.

There is also an advanced set of parameters titled "CUSTOM GIT REFERENCE SELECTION".
Normally, building the kuali research application with maven is done so against source code from the HEAD of the associated git branch.
However, you can use these advanced parameters to specify any other commit in the git repository to build against.
Currently, use of these advanced parameter assumes you are deploying some kind of feature branch or old commit point for testing temporarily

2. "kuali-research-1-build-war"
This job uses maven to build the kuali research war file. If you are using this build, your goal is to go through the "Basic Job Steps" as described in the previous section with respect to a particular ec2 instance. There are multiple parameters to consider if running manually:

a) BRANCH
The branch in the github repository to be pulled for the maven build.

b) CHECK_DEPENDENCIES
Build any kuali modules that are out of date (rice, s2sgen, api, schemaspy). This is determined by analyzing the pom file for the build and comparing to what can be found in the local .m2 repository

c) DEPLOY
If you uncheck this parameter, the war file will be built, but nothing will happen beyond that. Leaving this parameter checked will cause this job to continue and build the docker image, load it to the registry, and have each associated ec2 instance rebuild and rerun their respective docker containers accordingly.

d) ECR_REGISTRY_URL
This is the URL that identifies the location of the docker registry where newly build docker images are to be pushed to.

e) EC2_INSTANCE_ID
This is the identifier for the ec2 instance that is being targeted for deployment of the build.

f) GIT_REPO_URL
This is the url for the git repository where the source code for the build resides.

g) GIT_REFSPEC
The build is being executed against changes in source code. This job fetches those changes from the specified git repository, but limits what is fetched to what is specified in this parameter. If you leave this parameter blank, the job will fetch everything from the git repository. NOTE: This does not specify what commit to build from, only what is getting tracked and fetched from the remote repo.

h) GIT_BRANCHES_TO_BUILD
Once content is fetched according to GIT_REFSPEC, this parameter specifies what specific commit within that content will provide the source code that maven will compile to create the war file. This parameter is labelled with the numerous forms its value can take.

3. "kuali-research-2-docker-build-image"
4. "kuali-research-3-docker-push-image"
5. "kuali-research-4-docker-run-container"

"Kuali-Lib"

"apache-shibboleth"

"Patching and updating"

"Github"

"--- TESTING ---"