**Project Name: Jenkins Pipeline with Docker Integration on AWS**

**Step 1: Validate Jenkins application access, runtime installations, and Git repository:**

As part of this project, I validated the accessibility of the Jenkins application and ensured that it was properly installed and configured. I also verified the availability of the required runtime installations such as Java and other dependencies. Additionally, I validated the connectivity to the Git repository where the source code is stored.

**Step 2: Prepare Jenkins pipeline script with basic stages:**

In this step, I created a Jenkins pipeline script to automate the build and deployment process. The pipeline script consisted of several stages, starting with a code checkout stage to fetch the source code from the Git repository. Then, I incorporated Maven build automation to build the backend code and NodeJS build for the frontend code. These stages ensured that the code was compiled, tested, and packaged correctly.

**Step 3: Integrate Docker image build stage:**

To further enhance the pipeline, I integrated a Docker image build stage. This stage allowed the application to be packaged into a Docker image, providing consistency and portability. I used Dockerfile and relevant configurations to build the image, ensuring that all the necessary dependencies and libraries were included.

**Step 4: Create Docker Hub Credentials and prerequisites:**

To publish the Docker image, I created Docker Hub credentials. This involved setting up an account on Docker Hub and generating access tokens or API keys. These credentials were securely stored in Jenkins so that the pipeline could access them during the build process. I also ensured that any necessary prerequisites, such as Docker and Docker Hub configurations, were in place.

**Step 5: Integrate Docker Image publishing within the pipeline:**

In this step, I integrated the Docker image publishing stage within the Jenkins pipeline. After building the Docker image in the previous stage, I utilized the Docker Hub credentials to push the image to a designated repository on Docker Hub. This ensured that the Docker image was readily available for deployment or distribution.

**Step 6: Setting up Docker remote host on AWS and configuring deploy stage:**

To facilitate the deployment process, I set up a Docker remote host on Amazon Web Services (AWS). This involved provisioning an EC2 instance or any other suitable AWS resource to act as the Docker host. I configured the necessary security groups, access credentials, and networking settings to ensure secure and reliable communication with the Docker host. Finally, I added a deploy stage to the Jenkins pipeline, enabling the deployment of the Docker image to the AWS Docker host.

Throughout the project, I followed best practices for Jenkins pipeline development, such as using declarative syntax, incorporating error handling and notifications, and maintaining version control for the pipeline script. By completing this project, I gained expertise in Jenkins, Docker, and AWS, along with proficiency in building and deploying applications using a CI/CD pipeline.