1. **Start of the Jenkins Pipeline:** This marks the beginning of the Jenkins Pipeline. It sets the context for the entire pipeline and defines the stages and steps to be executed.
2. **Set up the Jenkins agent to run on any available node:** The `agent any` directive specifies that the pipeline can run on any available Jenkins agent/node. This allows Jenkins to automatically allocate an appropriate agent for the pipeline based on availability and labels.
3. **Define the tools used in the pipeline:** a. **Install and configure Maven with the label "M3":** The `tools` directive is used to define and configure the tools needed in the pipeline. In this case, it installs and configures Maven with the label "M3." The label can be used later in the pipeline to specify that a certain stage should run on a node with this label, where Maven is available.

   b. **Install and configure Java 1 with the label "java1":** Similarly, the `tools` directive installs and configures Java 1 with the label "java1." This allows later stages in the pipeline to run on a node where Java 1 is available.
4. **Define the stages in the pipeline:** a. **Checkout from GitHub:** The first stage in the pipeline is to checkout the source code from the specified GitHub repository. This is achieved using the `git` command, which clones the "awtraining1/sl" repository from the "main" branch.

   b. **Maven Build:** The second stage involves building the Spring Boot application using Maven. It navigates to the "PHASE5/my/spring-bootdemo" directory using the `dir` directive and then runs the command `mvn clean package` to build the Spring Boot application. The `-Dmaven.test.skip=true` flag is used to skip running tests during the build, which can speed up the build process.

   c. **Docker Image Creation:** The third stage is dedicated to creating a Docker image for the Spring Boot application. It navigates to the "PHASE5/my/spring-bootdemo" directory using the `dir` directive and then builds the Docker image using the Dockerfile. The Dockerfile is configured to use an Alpine-based JDK 17 image, copy the packaged JAR file ("app.jar") into the image, and set the entry point to run the JAR.

   d. **Push Docker Image:** The fourth stage involves tagging the Docker image created in the previous stage with the name "my-morning-spring-app" using the `docker tag` command. After tagging, the pipeline pushes the tagged Docker image to the Docker Hub repository "anithaneel/my-morning-spring-app" using the `docker push` command.

5.End of the Jenkins Pipeline: This marks the end of the Jenkins Pipeline. At this point, all the defined stages and steps have been executed successfully (assuming there were no errors). The pipeline can now be considered complete, and any further post-processing or cleanup tasks can be performed if necessary.