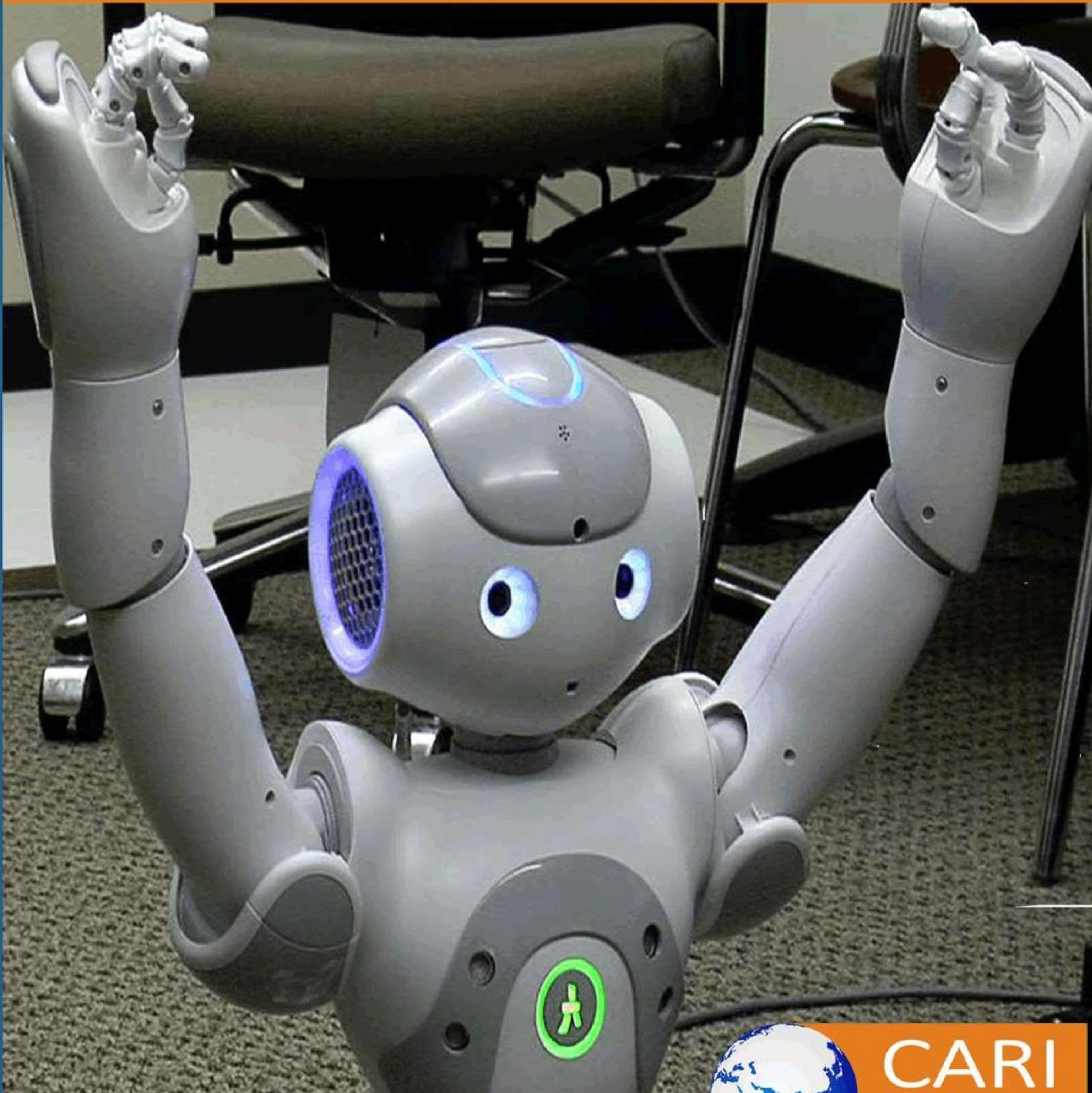
International Journal of
Computing and Engineering
(IJCE)

Build Your Continuous Integration and Continuous Delivery Pipeline for Salesforce Platform



International Journal of Computing and Engineering
ISSN 2958-7425 (online)
Vol. 1, Issue No. 2, pp. 1 - 10, 2020 www.carijournals.org

Build Your Continuous Integration and Continuous Delivery Pipeline for Salesforce Platform

Rishitha Kokku

Salesforce Technical Consultant, Optum Services INC

https://orcid.org/0009-0007-7810-2292

Accepted: 8th Sep, 2020, Received in Revised Form: 19th Sep, 2020, Published: 18th Oct, 2020

ABSTRACT

Salesforce is a CRM platform offering its customers point and click options to build their applications. This work for small scale industries and organizations whose operations are limited. Salesforce is very different to other technology stacks and hence will be its DevOps implementation. Companies when trying to expand their business cannot achieve their requirements with configurations and often need customizations and code development. Deployment of this code is achieved by implementing a CICD process which is different from DevOps practices of other technologies. Salesforce offers a built-in deployment option called Changesets to migrate the changes between environments. This guide talks about what is Changesets and what are the other ways to implement Salesforce DevOps.

Keywords: *Jenkins, SFDX and CLI, Apache ANT, Copado, AutoRABIT*

1. INTRODUCTION

Understanding Salesforce metadata is the first step to achieving successful deployments. Salesforce metadata is complex and is often tightly coupled and interlinked with each other. There are custom and standard objects relationships in Salesforce defining the hierarchy of the metadata. Deployments into Salesforce environments should be carefully tackled due to these relationships and the references in the metadata. but there are various limitations to this approach. Changesets have various limitations and are not suitable for large organizations that develop complex code and applications that majorly rely on customizations for their businesses [1]. This guide discusses these limitations and how to implement DevOps process to regulate the deployment into Salesforce environments and improve the efficiency.

2. WHAT IS A DEPLOYMENT IN SALESFORCE

Salesforce developers are often given their own developer sandbox unlike other tech stacks where everybody works on a single dev instance. The changes that are developed in these developer sandboxes should be deployed to an integrated sandbox to check the stability prior to passing it to testers. Often there are configuration changes from the developers that are to be replicated in the integrated sandbox and other environments up until Production [2]. Deployments of these changes done by each developer are done using Changesets when there are no DevOps practices followed and some of the configurations are not supported with Changesets deployment.

3. LIMITATIONS OF CHANGESETS

3.1. Version History

Changesets offer no history on the modified files making it difficult to identify the contributors to a specific feature or a file. It is also impossible to get a list of contributors to the metadata over a period of time.

3.2. Unsupported Metadata

Salesforce does not support all metadata types through Changesets deployment though it's an inbuilt feature for their deployments [3,4]. Complex applications make use of various metadata types to build their applications and when such metadata is not supported for deployment the migration becomes complicated. Figure 1 lists some of the supported and unsupported metadata types by Changesets.

Metadata Type	Supported by Changesets?
Custom objects and Custom Fields	✓
Standard picklist values	✗
Apex Classes and Triggers	✓
Reports containing certain features	✗
Lightning and VF pages	✓
Sales Processes	✗
Platform Events	✗
Profiles and Permission sets	✓

Figure 1. Supported and Unsupported metadata types in Changesets

3.3. Manual Process

Unlike a structured CICD pipeline, in Changesets each file that is to be deployed must be selected individually. When there are various metadata types involved, this process can be daunting and make room for error. Apart from selecting the files other manual steps like uploading, logging into multiple environments, validating and deploying are involved which is time consuming.

3.4. No control on including dependencies

When selecting the files to add to Changesets there is no way to look at the list of dependent files to be included to make it a successful deployment. This can lead to adding unwanted and unrelated files making it difficult to review the changesets. Once uploaded into the destination environment, Changesets cannot be modified [4]. If you realize there are missing files a new Changeset has to be created and uploaded.

3.5. Misuse of Admin Privileges

Developers are given Administrator permissions to develop and deploy the changes. There is no restriction on what and when to deploy using Changesets for Admins. This level of access can let the developers deploy untested code to Production and introduce bugs [5].

3.6. No data migration

Changesets support only certain metadata deployments and not the data that's stored inside the metadata. Custom metadata and Custom settings in Salesforce are most commonly used to store reference data for easy availability. This data cannot be deployed using Changesets and should be manually created in every environment or use another data loading tool to migrate the data into Production and Sandboxes [4]. This data has to be maintained, imported and loaded into the environments along with the code deployment which can be tedious to the development and operations teams.

4. WAYS TO IMPLEMENT CICD IN SALESFORCE APPLICATIONS

4.1. Force.com migration tool using Apache ANT

Salesforce has designed the Force.com migration tool to deploy the changes using the jar file from Apache ANT. It's a command-line interface for executing the commands by user after the initial setup is done. This process has been in use since Salesforce's inception and is widely used across organizations. There are certain prerequisites to be met to build this process and the user's machine should have Java and Apache ANT downloaded and installed.

Set the environment variables for Java and ANT once the softwares has been installed as below.

```
ANT_HOME = C:\path_to_ant_directory
JAVA_HOME = C:\path_to_java_lib_directory
```

Add the %ANT_HOME%\bin and %ANT_HOME%\bin to the "PATH" environment variable. To verify the setup open the command prompt and check the version of Java and ANT. The result should print the versions without giving any error

Download the Force.com Migration Tool zip file from Salesforce and save the directory in a place that's easy to access. It is recommended to rename the folder to "Salesforce ANT" to differentiate its contents from Apache ANT. Copy the "ant-salesforce-jar" file from ANT folder and paste it in the "lib" folder inside the Salesforce ANT folder. Use the *build.xml* file and *build.properties* file to set the deployment target, deployment root, testelevel and credentials to the destination org. Place the metadata contents inside the *sample* folder. Check for the *package.xml* file and make sure that all the metadata types inside the sample folder are listed in the xml file.

Open the command line interface from the Salesforce ANT directory where the build files are located and execute the ant commands to validate, deploy and destruct the metadata. It is important to note that all the metadata sitting in the sample folder will be deployed to the destination org once the command is executed [5]. Cherry-picking and deploying delta files is not possible with this process as there is no version control. Another caveat to this process is that all the apex classes listed in the sample folder should have test classes and the overall coverage should be above 75% when deploying to Production. If there is not enough code coverage it results in a failed deployment.

4.2. MDAPI deployment with Jenkins Freestyle job and GitHub

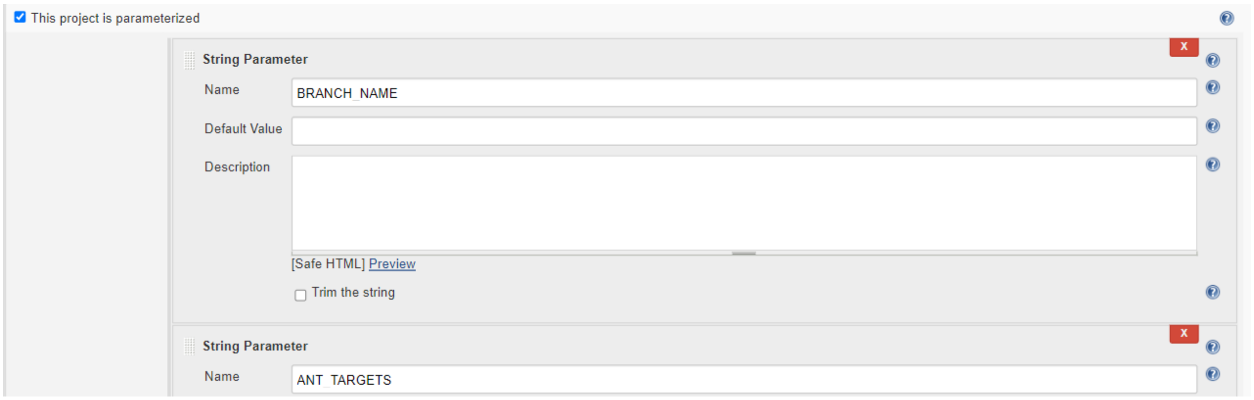
It is necessary to have Jenkins installed and GitHub setup with a repository to streamline the CICD pipeline. This is a more simplified and enhanced version of the above mentioned approach addressing its limitations. Jenkins should be having ANT and GitHub plugins installed and configured under "Manage Jenkins" to build a successful pipeline. Folders can be uploaded into the GitHub repo in the same structure as mentioned in 4.1.

Create a new job in Jenkins and give an appropriate name to it. It can either be a free style or multi-branch pipeline based on the branching strategy. Once the job is created, navigate to the configure part and provide the following details.

1. Description: Briefly describe what this job will be used for to differentiate it from other jobs.

2. GitHub: Check the box next to "GitHub Project" and provide the URL to the GitHub repository.

3. Parameters: To make the job easier to build, check the "This project is parameterized" and create parameters of your choice. Suggested parameters would be the Targets from the build.xml, Branch name, Username and Password to the destination environment, Validate and Deploy dropdown, Start_Revision to pick the delta files, TestLevel for each deployment etc [6].



☑ This project is parameterized

String Parameter
Name: BRANCH_NAME
Default Value:
Description:
[Safe HTML] Preview
☐ Trim the string

String Parameter
Name: ANT_TARGETS

Figure 2. Examples of Parameters in a Parameterized Jenkins job

4. Source Code Management: Select Git as the SCM and provide the repository URL. For credentials field, a new credential must be created for the service user that has access to both Git and Jenkins and this credential should be selected.

5. Build: In this section select the "Invoke Ant" from the dropdown and provide the ANT version configured in Manage Jenkins mentioned at the beginning of this section and the target would be the one configured above as shown in Figure 2.

Once the configuration is saved click on "Build with Parameters" and select the appropriate value from the parameters. Provide the Start revision from GitHub commits so that the deployment package is smaller and only deploys the needed files [7]. Open the console output to view the results of the deployment. If the configuration is done right the deployment should be successful [8].

4.3. MDAPI deployment with Jenkinsfile and GitHub

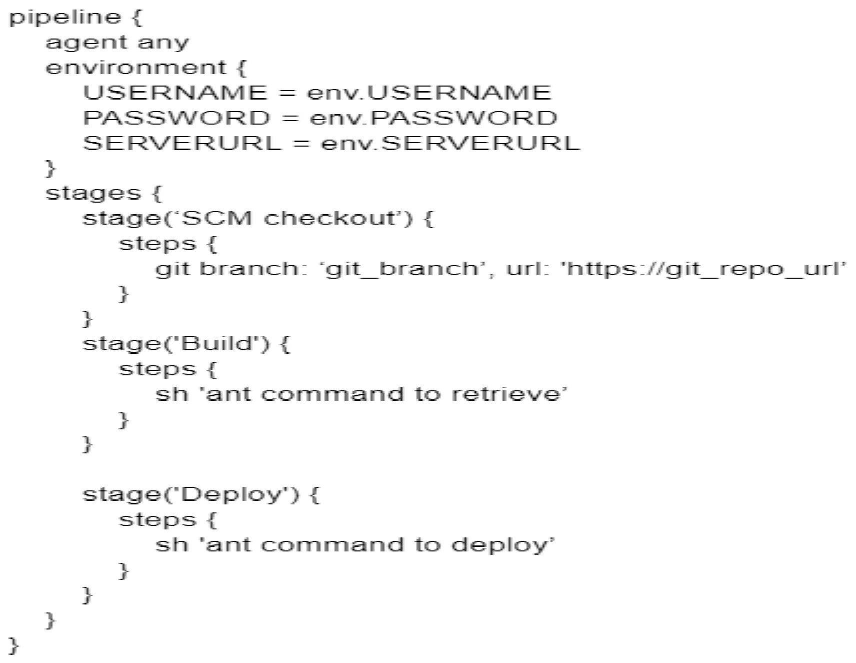
This approach is similar to the above approach except the configuration is controlled by the Jenkinsfile. All the prerequisites remain the same and configure the Jenkins pipeline to use a Jenkinsfile. This Jenkinsfile is stored inside the GitHub repository in the root directory. Jenkinsfile can have multiple stages and can be customized as needed. Some of the Stages in the Jenkinsfile are

1. **Environment Variables:** Store the Username, Server URL, MaxPoll as environment variables in Jenkins and refer to their labels inside the Jenkinsfile. Passwords can be stored as credentials in Jenkins under Credentials interface [9]. This is a more secure way of storing the password and limiting the exposure.

2. **Source checkout:** The first stage inside the Jenkinsfile is to checkout the SCM from the GitHub repository. The steps inside this stage have the Git URL and the branch to use. Once the authorization is successful it moves to the next stage.

3. **Build:** This stage refers to the build.xml target to retrieve the metadata from the deployroot mentioned in the target and builds the package.

4. **Deploy:** This stage is used to deploy the retrieved components into the destination environment mentioned in the build.xml target. It pulls the credentials from the environment variables defined at the beginning of the pipeline and executes the deployment. Tests are also run based on the TestLevel mentioned in the target. Figure 3 is a sample Jenkinsfile with stages defined, "env." is used to pull the values from the environment variables mentioned in Jenkins.



```
pipeline {
   agent any
   environment {
      USERNAME = env.USERNAME
      PASSWORD = env.PASSWORD
      SERVERURL = env.SERVERURL
   }
   stages {
      stage('SCM checkout') {
         steps {
            git branch: 'git_branch', url: 'https://git_repo_url'
         }
      }
      stage('Build') {
         steps {
            sh 'ant command to retrieve'
         }
      }

      stage('Deploy') {
         steps {
            sh 'ant command to deploy'
         }
      }
   }
}
```

Figure 3. Sample Jenkinsfile

4.4. SFDX deployment with Jenkinsfile and GitHub

SFDX is a combination of tools designed to improve the development and deployment cycle. Salesforce has introduced the concept of scratch orgs for a source driven development approach. It has proved to have a better team collaboration, version control and better quality of output. CICD approach offers a next level advantage for SFDX model development. The implementation steps are similar to MDAPI but the commands and plugins used will be different. Taking into consideration the Jenkinsfile stored in the root directory of the repository, below are the steps to follow for building a working pipeline

1. Server files: The JWT authorization into the org requires *server.key*, server.csr and *server.crt* files. Make sure to have OpenSSL installed and open it from the directory you want to save these files in. Execute the below commands.

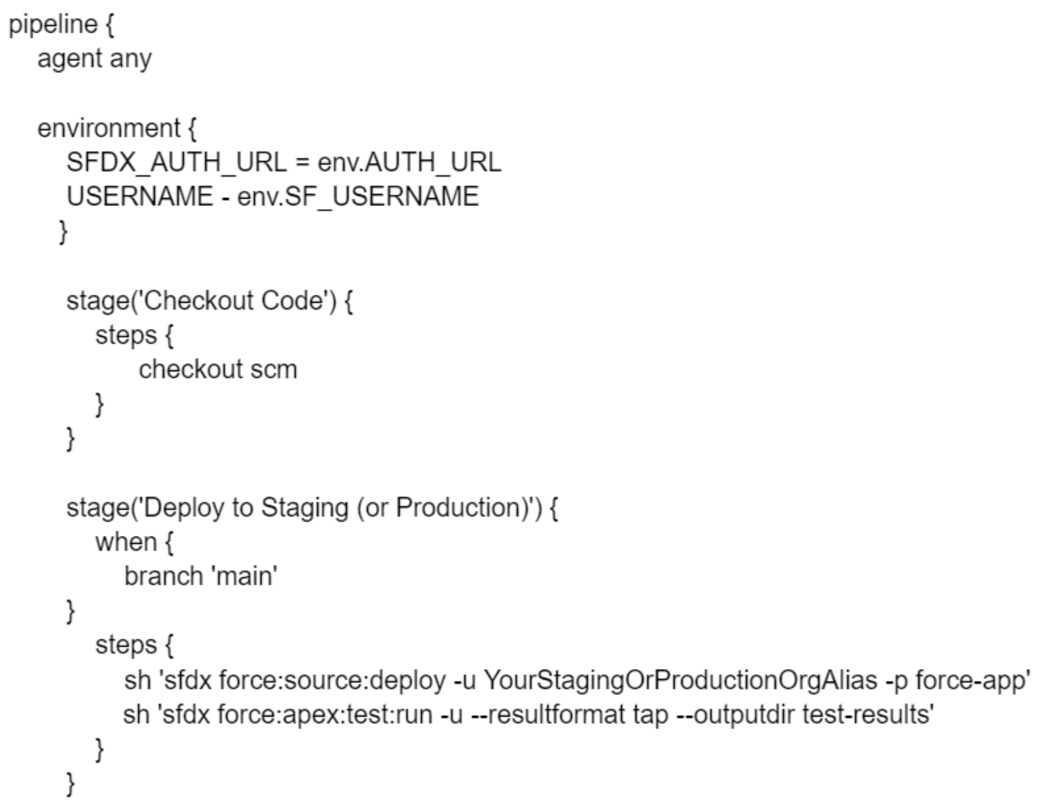
```
openssl genrsa -des3 -passout pass:Password -out server.pass.key 2048
openssl rsa -passin pass:Password -in server.pass.key -out server.key
openssl req -new -key server.key -out server.csr
openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt
```

2. Connected App: SFDX deployment authorization in Jenkins happens through a connected app created in the Salesforce environment. Create a connected app and upload the server.crt created in step 1 above [10]. Each environment in the pipeline should be having a connected app of its own and the System Administrator profile should be granted access to it to deploy the changes. The call back URL will be the URL to the Jenkins instance

3. Environment Variables: The environment variables for SFDX approach would be different from MDAPI. The consumer key from the connected app created in step 2 is stored in Jenkins as an environment variable. Store the server.key file in Jenkins as a credential and create another environment variable using the credential ID as its value. Store the Jenkins username as the next environment variable. To verify the authorization using the JWT flow, execute the below command. If the below command executes without any error, it's time to create a Jenkins job to build the pipeline

```
sfdx force:auth:jwt:grant --clientid "Your_clientid_here" --jwtkeyfile "location_to_your_server.key" --username "username" --instanceurl https://login.salesforce.com --setdefaultdevhubusername
```

4. Jenkins: Create a multi-branch pipeline in Jenkins and configure the job by providing the respective values. The Jenkinsfile will be having all the Stages for authorization, validation and deployment.



```
pipeline {
  agent any

  environment {
    SFDX_AUTH_URL = env.AUTH_URL
    USERNAME - env.SF_USERNAME
   }

    stage('Checkout Code') {
      steps {
          checkout scm
      }
    }

    stage('Deploy to Staging (or Production)') {
      when {
        branch 'main'
    }
      steps {
        sh 'sfdx force:source:deploy -u YourStagingOrProductionOrgAlias -p force-app'
        sh 'sfdx force:apex:test:run -u --resultformat tap --outputdir test-results'
      }
    }
```

Figure 4. Sample Jenkinsfile for SFDX

5. RECOMMENDATIONS

5.1 Version Control: Use a version control system (VCS), such as Git, to store all Salesforce metadata, including code and declarative changes.

5.2 Automate SCA: Implement automated static code analysis scans within the pipeline to make sure quality code is delivered and vulnerabilities are caught in early stages of SDLC.

5.3 Automate Deployment: Automate the deployment and unit-testing within the pipeline for every code commit. This will eliminate human intervention and CI/CD.

5.4 Implement Monitoring and Roll Back Mechanism: Constant monitoring and feedback loops help reduce downtime of the application and deliver quality to the business.

6. CONCLUSION

The main goal of CICD implementation in Salesforce projects is to eliminate the manual deployments, improve collaboration, reduce defects in Production, maintain proper versioning of files and reduce the deployment time and human errors. The above mentioned processes aligns with DevOps practices and helps organizations to faster deliver the project and improve their productivity.

REFERENCES

1. J. Smart, J. Humble, and R. Deane, *Lean Enterprise: How High Performance Organizations Innovate at Scale*, O'Reilly Media, 2015.
2. Dive, P., Gornalli, N, *DevOps for Salesforce,* Packt Publishing Ltd, 2018.
3. Salesforce.com, *Salesforce Metadata API Developer Guide*. [Online]. Available: https://developer.salesforce.com/docs/atlas.en-us.226.0.api_meta.meta/api_meta/meta_intro.htm. Accessed: August 12, 2020.
4. Salesforce.com, *Metadata Components and Types*. [Online]. Available: https://developer.salesforce.com/docs/atlas.en-us.226.0.api_meta.meta/api_meta/meta_objects_intro.htm. Accessed: August 16, 2020.
5. Salesforce.com, *Modify Metadata Through Metadata API Functions Permission*. [Online]. Available: https://developer.salesforce.com/docs/atlas.en-us.226.0.api_meta.meta/api_meta/meta_modify_metadata_perm.htm. Accessed: August 16, 2020.
6. Salesforce.com, Deploying and Retrieving Metadata with the Zip File. [Online]. Available: https://developer.salesforce.com/docs/atlas.en-us.226.0.api_meta.meta/api_meta/file_based_zip_file.htm. Accessed: August 16, 2020.
7. Salesforce.com, *Sample package.xml Manifest Files*. [Online]. Available: https://developer.salesforce.com/docs/atlas.en-us.226.0.api_meta.meta/api_meta/manifest_samples.htm. Accessed: August 20, 2020.
8. Salesforce.com, *Check the Status of Your Deployment Using REST Resources*. [Online]. Available: https://developer.salesforce.com/docs/atlas.en-us.226.0.api_meta.meta/api_meta/meta_rest_deploy_checkstatus.htm. Accessed: August 20, 2020.

9. Jenkins.io, *Jenkins User Documentation*. [Online]. Available: https://www.jenkins.io/doc/. Accessed: August 22, 2020.
10. Salesforce.com, Create a Private Key and Self-Signed Digital Certificate. [Online]. Available: https://developer.salesforce.com/docs/atlas.en-us.240.0.sfdx_dev.meta/sfdx_dev/sfdx_dev_auth_key_and_cert.htm. Accessed: September 16, 2020.