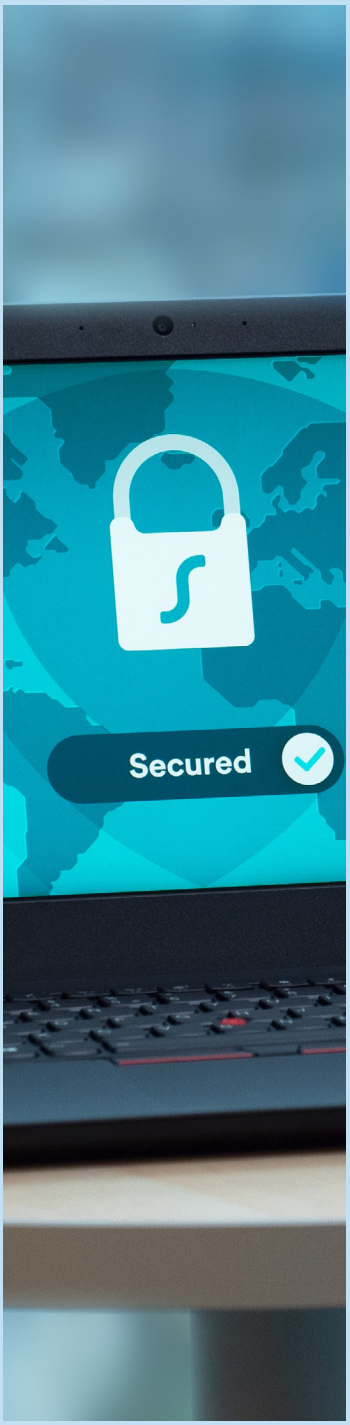
VIEW POINT



SECURE END-TO-END DEPLOYMENT WITH INTEGRATION OF ON PREMISE AND AWS JENKINS

Abstract

For one of the European banking customer projects, Infosys is implementing a SaaS Solution, deployed on Amazon Web Services (AWS)– a public cloud. Due to various component dependencies and customer compliance requirements, codebase needs to reside in customer network (i.e., on premise), whereas application must be deployed on public cloud (i.e., AWS) With on premise code setup, deployment of the code automatically is a challenge, particularly because the current client infrastructure does not allow outbound calls.

Similar other engagements had opted the option of manual deployment that involves getting the build artifacts manually (on Premise) and then pushing it to AWS infrastructure followed by manual deployment in applicable AWS environments (Dev, SIT, UAT et al). This process is error prone, unsecure, person dependent and can be tinkered with to push undesired workloads to environments (including production).

The paper describes the implemented solution to provide end-to-end, secure, automated deployment of UI and Java microservices components to S3 and Kubernetes. The implemented pattern could be implemented across cloud projects for use case mentioned above, we feel that this pattern can be useful for other Infosys projects.

Technologies involved are Jenkins, Jenkins Remote Trigger, SFTP Server, EFS Mount, Lambda, API Gateway, Cloudwatch, IAM

Infosys®
Navigate your next

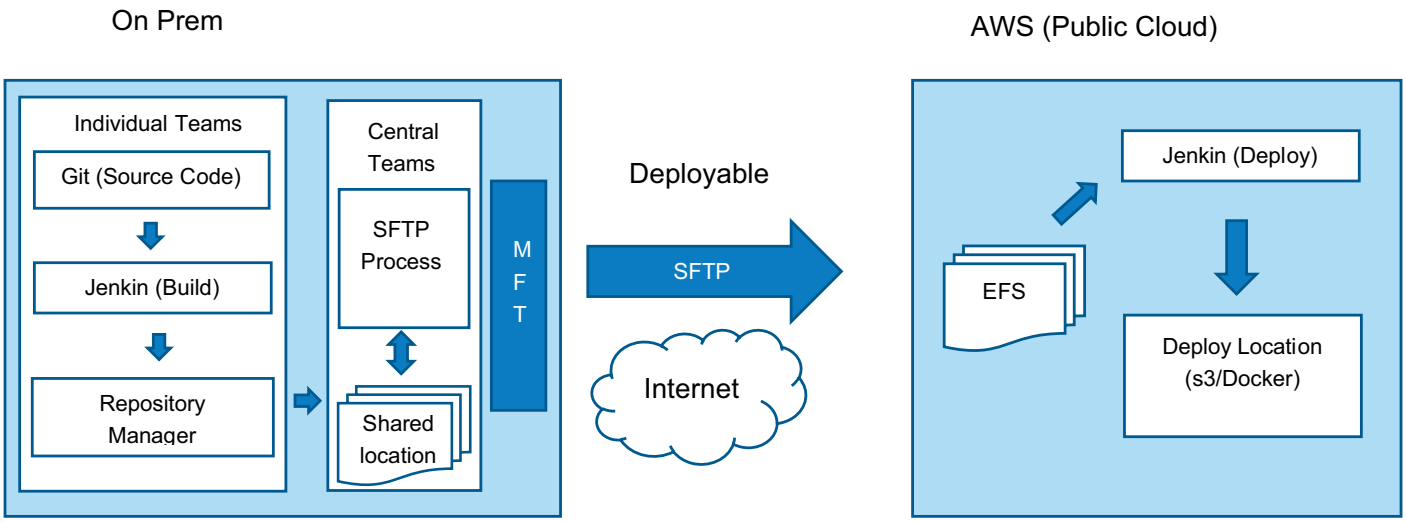
Introduction

Continuous integration and Continuous deployment (CI/CD) are a critical stage of software development life cycle. CICD process needs to be automated with minimal manual intervention. CI/CD process requires additional considerations where Continuation Integration (CI) and Continuous Deployment (CD) processes exist in different system boundaries.

While organizations transition to clouds, due to compliance requirement, and existing infrastructure constraint, scenarios need to be addressed where – Application development and build is done on Local network (i.e., On Prem) whereas application needs to be deployed on public cloud (Example AWS).

While technology stack mentioned in the paper includes Jenkin for CICD, AWS services (EFS, API Gateway, Lambda, CloudWatch etc.). The concepts could be applied for any of the technologies.

Problem Statement



The setup in above diagram depicts the use case where because of various dependencies (including compliance requirement, existing infrastructure) codebase needs to reside on premises, the application needs to be deployed on AWS. The on-premises infrastructure does not allow direct network outbound calls, resulting target application servers (i.e., on AWS) being in-accessible from on-prem. This requires separate deployment pipeline on cloud. This setup requires following components–

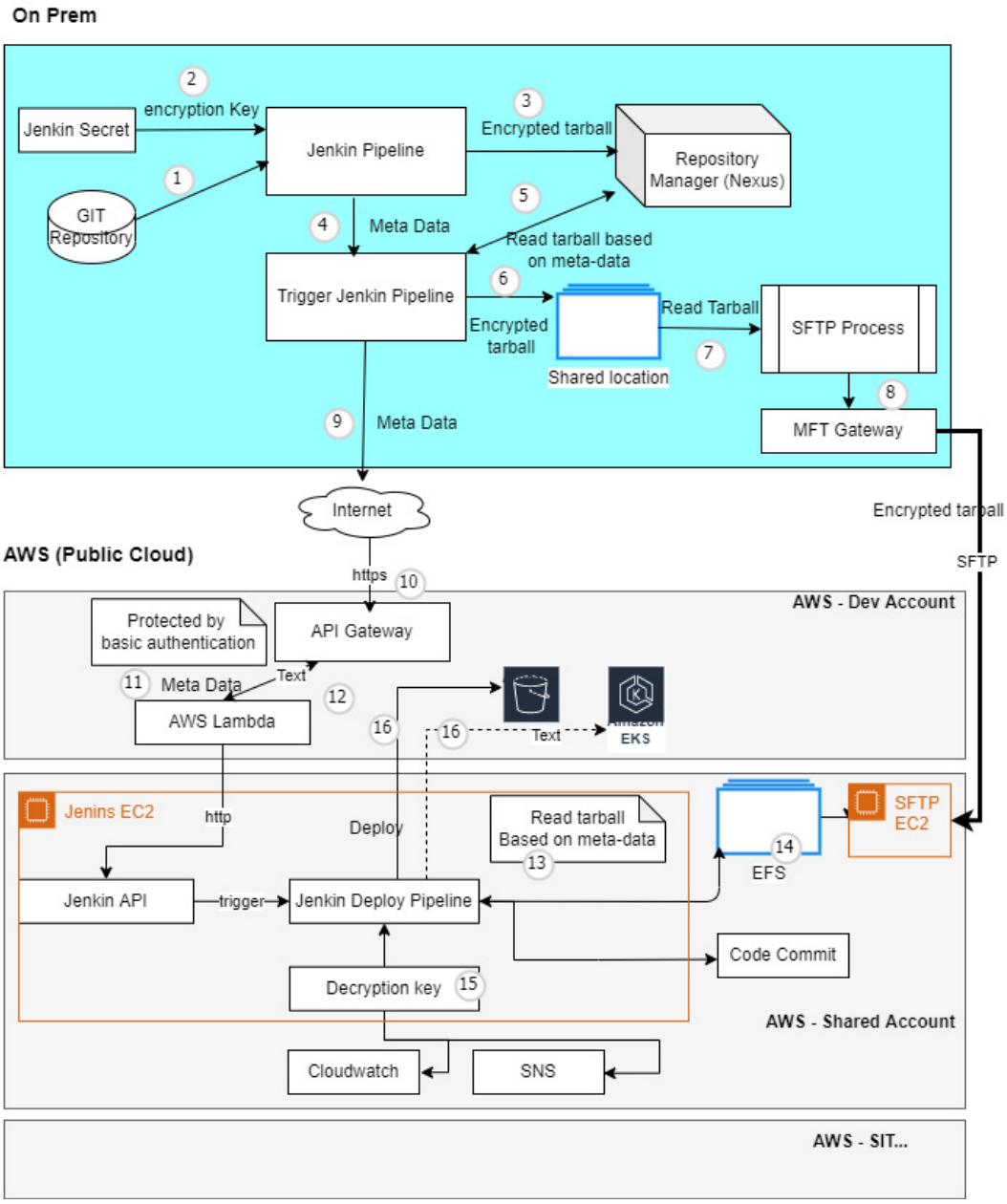
- **On Premise Jenkin Pipeline** – The pipeline is developed and maintained by application team. The pipeline is to access the source code repository, creates build artifact, stores (optional) on repository manager (ex. Nexus).
- **SFTP Process** – The responsibility of the SFTP process is transfer build artifact to predefined AWS Elastic file system (EFS) using MFT gateway. As this process is required across teams, is managed by central team.
- **Jenkin Pipeline on AWS** – The pipeline is developed and maintained by application team. The pipeline reads the deployable from AWS EFS and deploys it to deploy location (ex. S3, Kubernetes cluster etc.)

Challenges

The design described above poses few of the challenges which needs to be addressed –

- Manual intervention required
  - On prem SFTP process requires build artifact to be copied manually to shared location in order it to be FTPed to AWS.
  - On AWS artifact needs to be deployed by manually (including in production) triggering Jenkin pipeline.
- On Prem & AWS Jenkin are not integrated
  - On Prem and AWS Jenkin pipeline work independently. It requires due human diligence to ensure correct build artifact generated by on prem pipeline is deployed by AWS pipeline.
- Security concerns
  - Manual deployment of artifact is person dependent, error prone and does not ensure integrity of artifact being deployed. Artifacts could be tampered with during SFTP process or manual deployment process. Compromised SFTP might result in deployment of un-trusted build artifact.

Proposed Solution



Overall Flow

1. On premise Jenkin pipeline reads the source code from on premise bitbucket's git repository.
2. The pipeline encrypts the build process generated artifact (tar-ball) using secret key by reading it from Jenkin Credential store. Symmetric/ Asymmetric key cryptography could be used to encrypt the tar-ball.
3. The pipeline publishes the encrypted tar-ball to Repository manager (example Nexus)
4. The pipelines Invoke another trigger Jenkin pipeline with build meta-data. Meta data has properties related to currently published tar-ball (example tar-ball name). the meta-data is required during automated deployment

   Metadata is represented in JSON format for ease of processing. Following attributes are part of the meta data.

   { "fileName" : "tarball name",

   "jobName" : "name of Jenkins job"

   "environment" : "dev",

   "dryRun" : "true/false" // used to execute the job without or with actual deployment }

   Trigger Jenkins pipelines is scheduled to execute on the virtual machines that reside in a separate subnet that is configured to connect to internet along with intranet.

5. Trigger Jenkin pipelines process downloads encrypted tar-ball from nexus based on input meta-data values.
6. Copies the encrypted tar-ball to shared location on premise.
7. As soon as tar-ball is copied to shared location, a polling enabled automated process reads the encrypted tar-ball.
8. Poller process SFTP the encrypted tar-ball to AWS's SFTP server. This file is stored on predefined AWS EFS.
9. As soon as SFTP is completed, trigger Jenkin process invokes AWS API gateway end point, passing the meta-data as input parameter. Meta data is explained in #4.
10. The REST API initiates the deployment on AWS.
11. API gateway internally calls AWS Lambda function.
12. Lambda function invokes the Jenkin API. Jenkin API is required to trigger pipeline execution remotely.
13. Jenkin API triggers the deployment pipeline. Deployment pipeline is implemented as Jenkinsfile using groovy language
14. Based on input metadata, Jenkin Pipeline reads the encrypted tar-ball from EFS. Symmetric encryption key is known only by DevOps team on both ends and cannot be read once set. This key is rotated every month to prevent any leaks.
15. Reads the decryption key from Jenkin secrets, decrypts the tar-ball. Alternatively, as a best practice decryption key could be stored in AWS Secret manager. Usage of asymmetric key enables to use secret manager auto rotation of keys.
16. Deploys the deployable to deploy location. UI components i.e., angular based SPA application is deployed in S3 bucket while spring boot microservices are deployed on Elastic Kubernetes Cluster. The pipeline pushes the job logs to cloudwatch and published message to SNS topic for successful and unsuccessful deployments. This SNS topic is configured to send emails to a distribution list that development team is part of.

Advantages

The proposed solution helps to achieve following –

End to end automation

The taken approach results in end-to-end automation without any manual intervention.

Integration of on prem and AWS Jenkin Pipeline

Though OnPrem & AWS Jenkin pipeline has different system boundary, On prem Jenkin pipeline triggers AWS Jenkin pipeline during CI lifecycle. As this interaction is automated, the process is more robust, less error prone, secure and repeatable.

Security

Security is the key consideration in the taken approach. End to end automation makes process more secured. Encryption/decryption of deployable using industry standard algorithm makes sure, tar-ball could not be tampered with. On premise interacts with AWS via SFTP & password protected secured REST API, making end to end system more secured and reliable.

About the Authors

Nachiket Deshpande
Senior Technology architect

Manish Pandey
Senior Technology architect

About the Mentor

Chidambaram GS
Senior Technologist

For more information, contact askus@infosys.com

Infosys®
Navigate your next