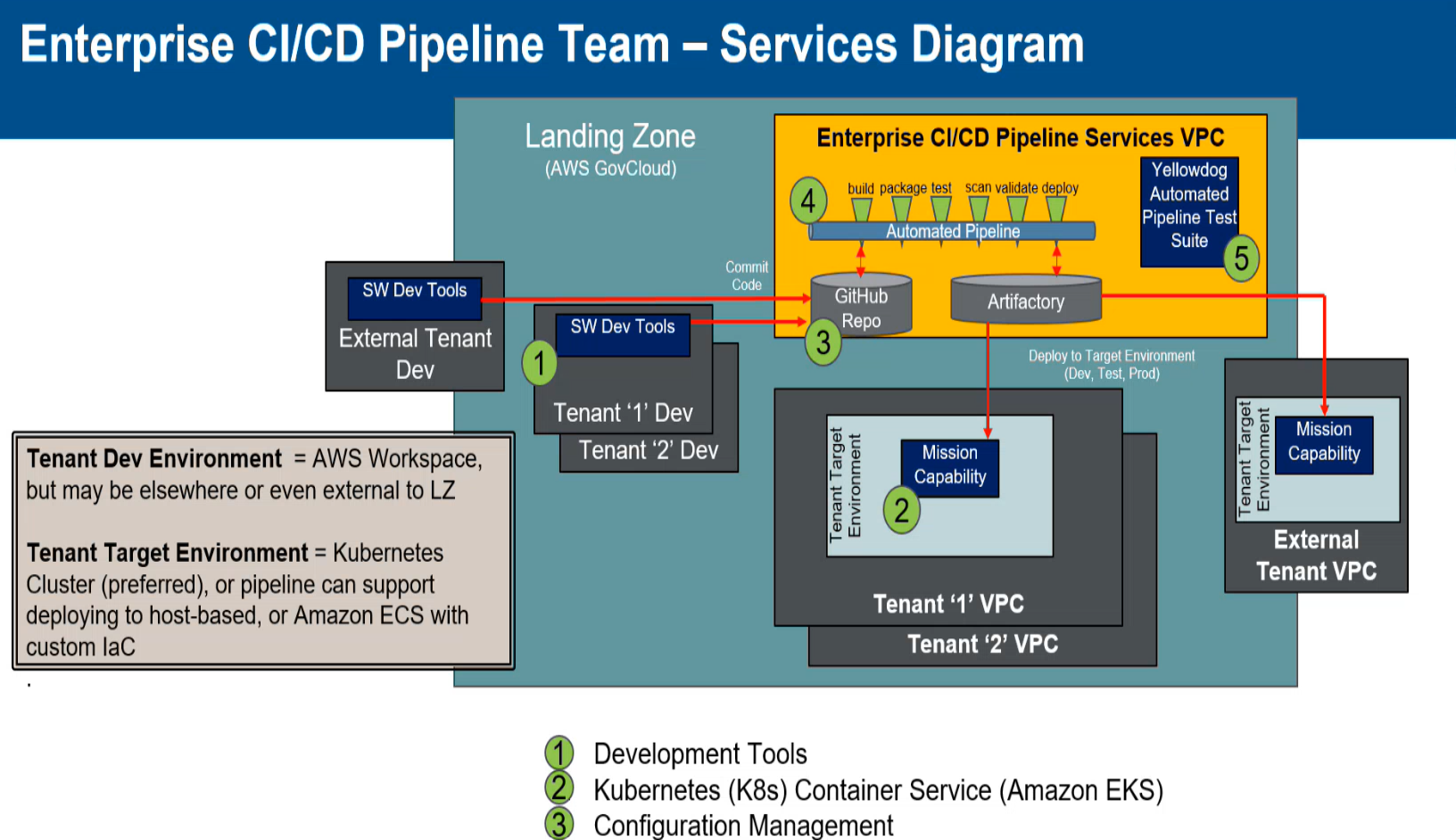
**Jenkins Pipelines**

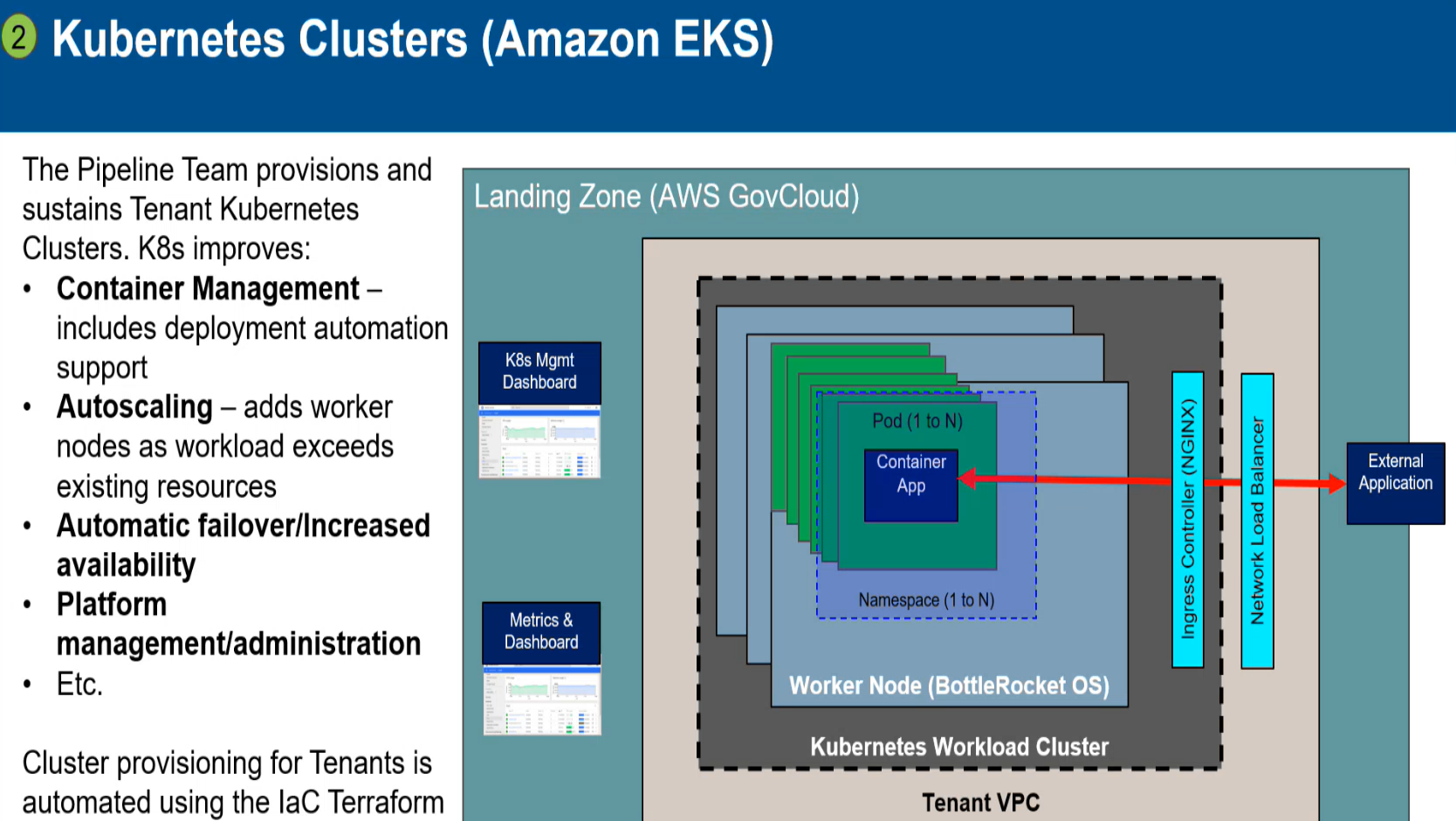
Thursday, November 7, 2024

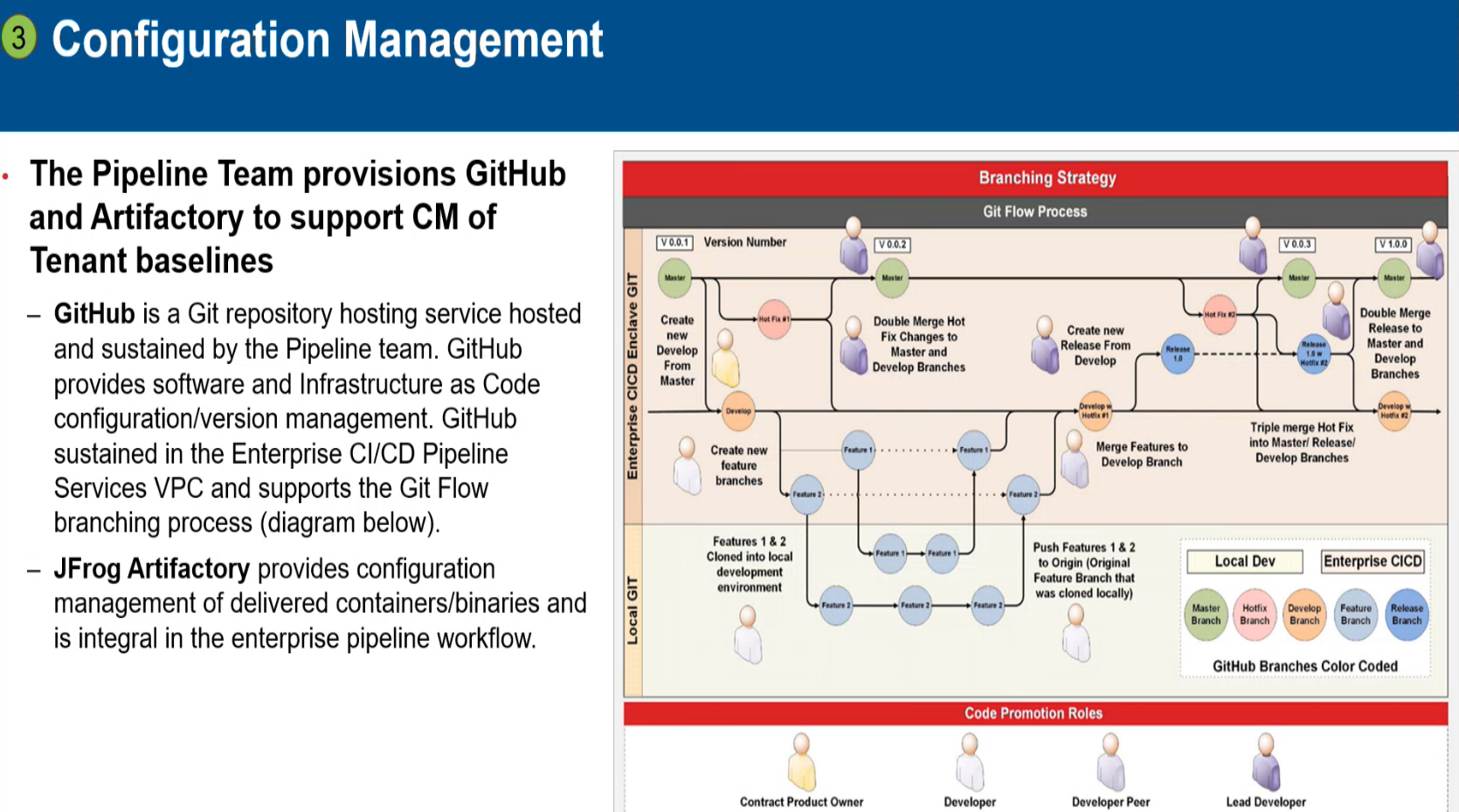
4:58 PM

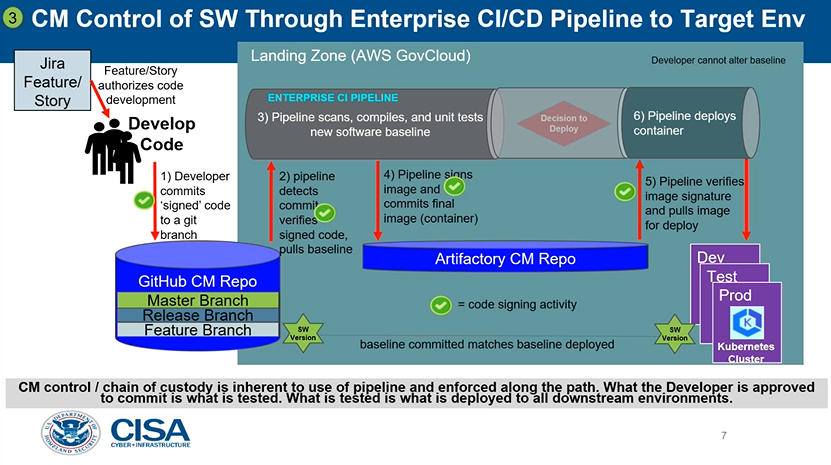
**Jenkins Pipelines Explanation:**

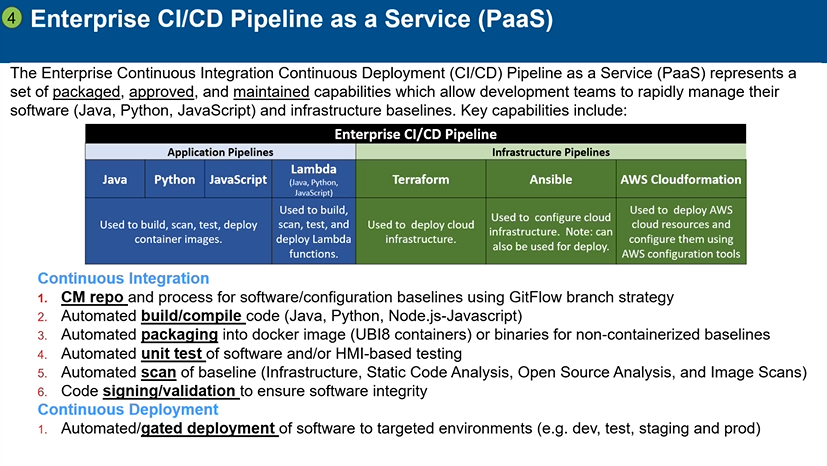
**-------------------------------------------**

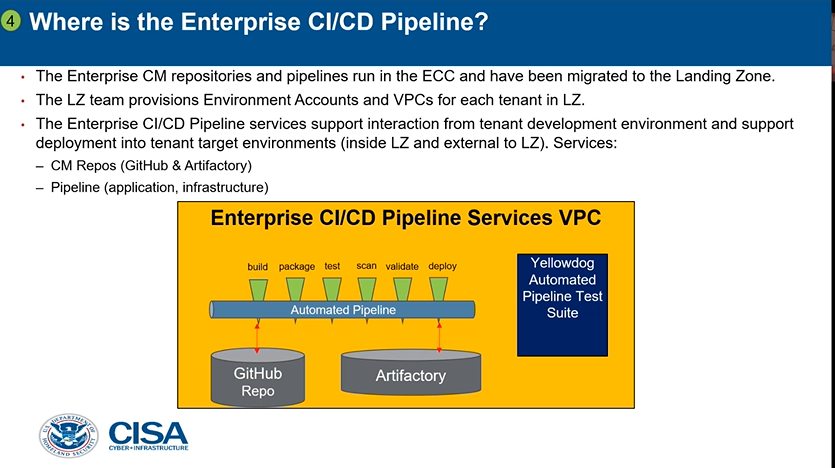


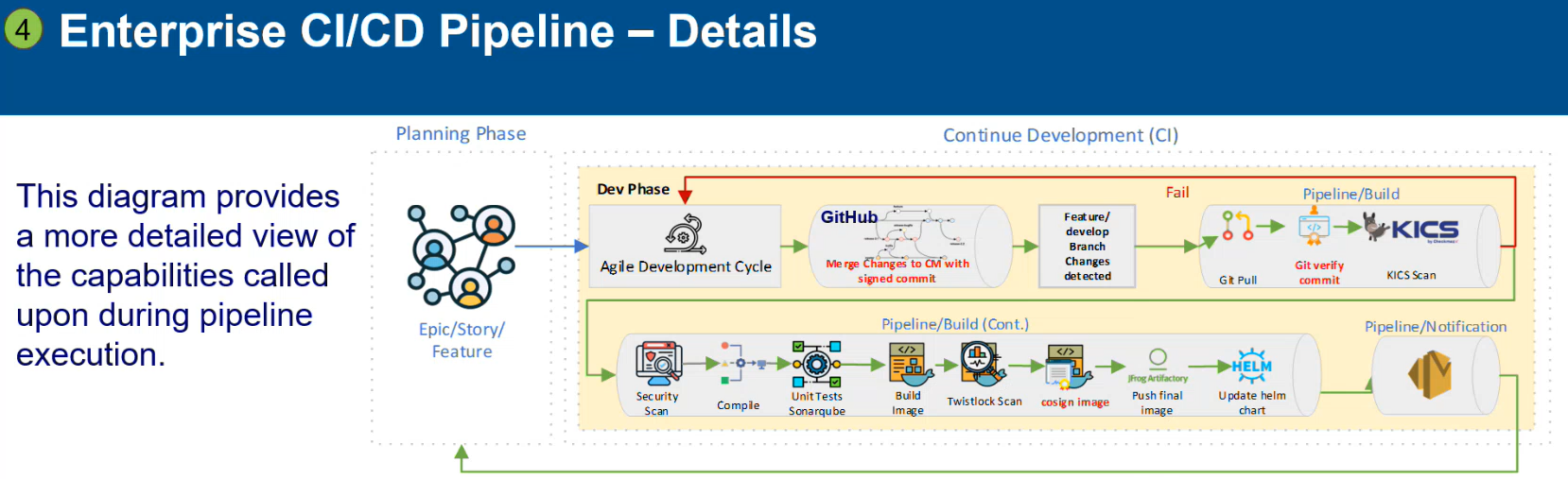


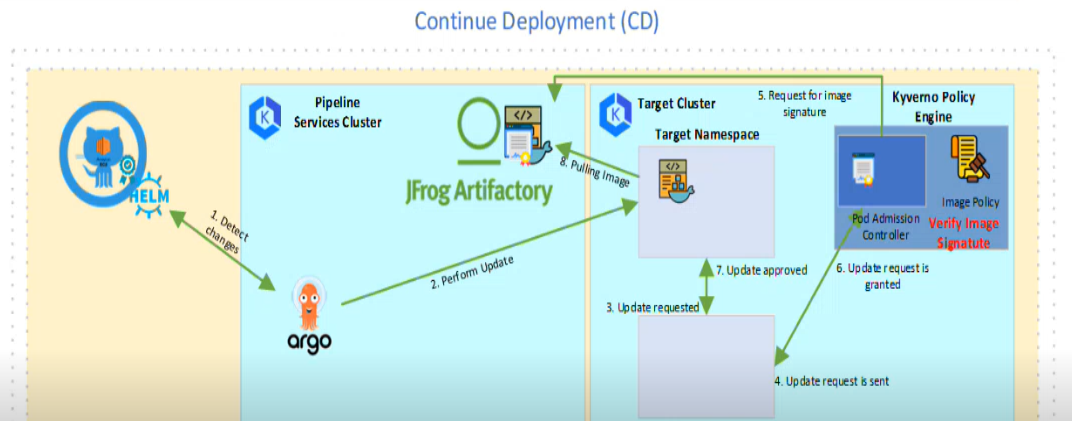


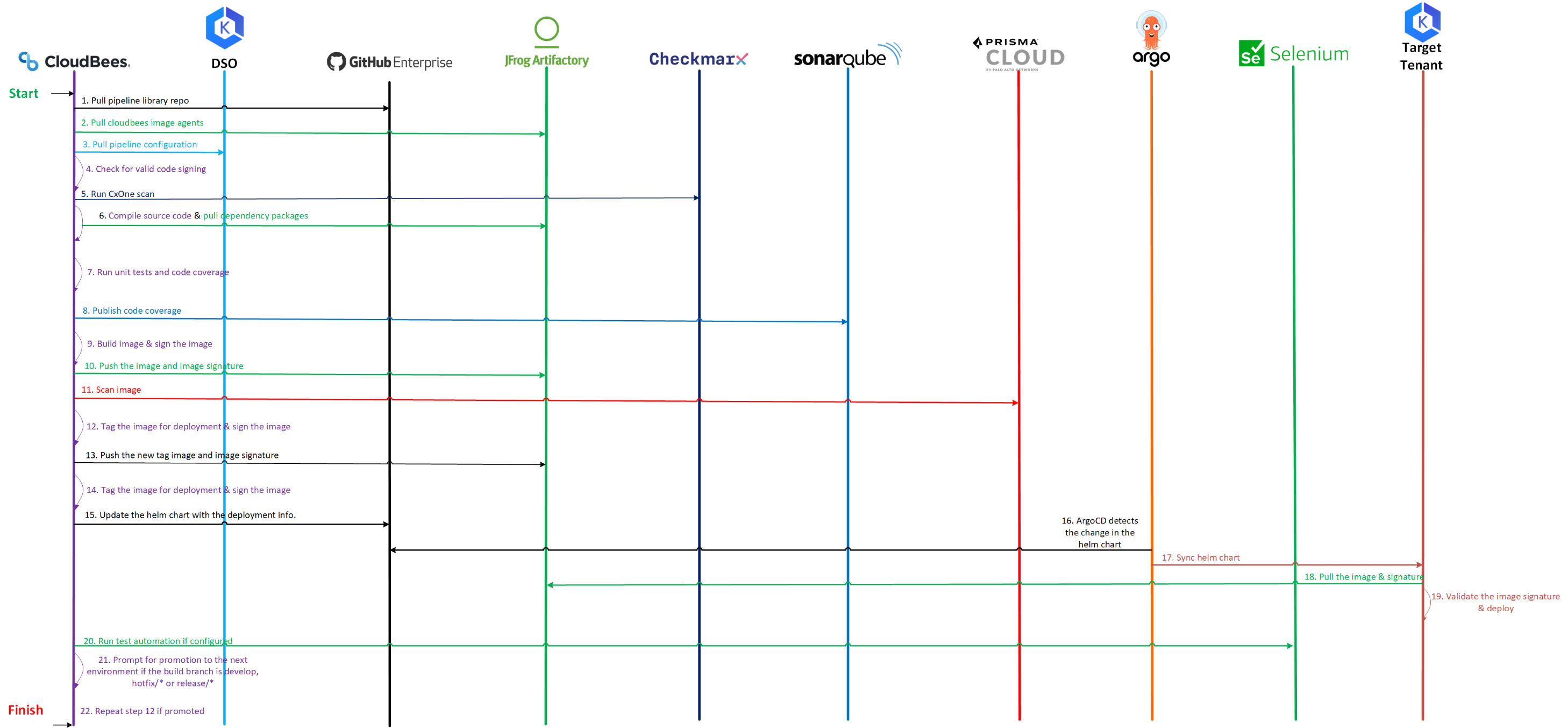












**The pipeline starts with the Software Development Cycle:**

**---------------------------------------------------------------------------------**

* + **The developers working on new features and fixing vulnerabilities.**
  + **They start by creating a feature branch and this branch would to go to github, and when they are ready to deploy and the can check in into github and that's when the feature branch is kicked off.**
  + **Then, the pipeline will pull the code into the new pod and first it will have to verify the latest commit has been signed by an individual with valid credentials.**
  + **After that, It is going to run Infrastructure as Code Scan (KICS Scan), which checks if there anything going on with the docker or any infrastructure as code in the repository. The KICS scan is only for reporting purposes, and it doesn't stop the pipeline if there is high or critical vulnerabilities but in the near future after integrating with CxOne, it will be active.**
  + **After that, we run into the security scan, which include the SAST which is static code analysis security test with CheckMarx & the dependency scan, where we consolidate the result of the scan into CodeDX. (Checkmarx & Dependency Track) = CodeDX**
  + **Once we pass the security scan and the code doesn't has any high or critical vulnerabilities, it will start to build the application and run the Sonarqube code scanning to report for code coverage. Sonarqube doesn't stop the pipeline if it has any code coverage issues like 80% or higher. (Also, we use cfn\_lint, cfn\_nag for CloudFormation Scan).**
  + **Then, we start building our docker image, and Twistlock will report if it has any high or critical vulnerabilities, and it will go to the next stage which is Code signing the image if the image doesn't has any high or critical vulnerabilities, and in this stage we sign the image with a special key which is used by the pipeline ONLY.**
  + **Then, we push that image and its signature key into Artifactory, and we update the Helm charts which conclude the Continuous Integration (CI) part.**
  + **Then, we start the Continuous Deployment (CD) Part, which is done using ArgoCD.**
  + **The ArgoCD will detect that there is a change in the helm chart which was updated in the CI section, because it scan the github repository every 5 mins.**
  + **Then, ArgoCD will redeploy the image to the target environment. But It will initially check if the image had been signed by the correct private key before doing the deployment. It verifies the image signature through the Kyverno policy engine in Kubernetes, which enforces the integrity of the pipeline.**
  + **Then, It will pull that image from Artifactory and deploy it into the target environment in the target namespace in Kubernetes.**
  + **This is the Entire CI/CD Pipeline.**
  + **Then, we have an optional automated Testing framework suite that would allow testing after the deployment has occurred in any environment. For example after deploying into a specific environment (Ex: Test), you can run a specific automated test for this environment, and you do the same for the Dev & Staging & Production as shown below. Where you can hook for a particular environment to run automated test**
  + **This automation test can be hooked into the pipeline, and run as a validated state for the next step. For example, before deploying into the test environment you can run the automation test on the dev environment and check whether or not all the functionalities have been checked and all the stories have been validated through the automation test. Then, after that the individual can promote the code to the next stage of the Test enviornment if the automation test has been successful in the dev environment.**

**Code Scanning Tools:**

**------------------------------**

* + **KICS - IAC Scan** ---> Scan the base directory
    - Scans Infrastructure code

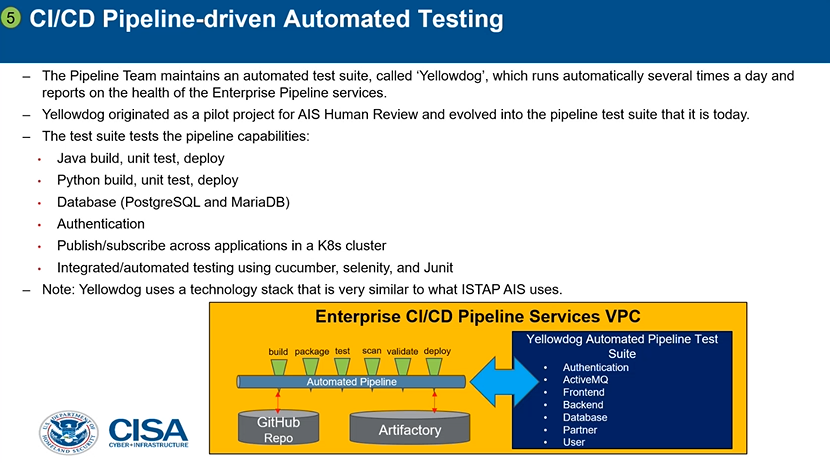
* + **tflint** ---> Scan the base directory

* + **SAS** - Static Analysis
  + SCA Scan
    - Dependencies and open source libraries.

* + **OSHA** - Open Source Scan

* + **APISEC - API Security Scan** ---> Scan the

* + **Inspec** --> Scans the InspecProfile Folder



**Lynn Explanation for NPM Pipeline:**

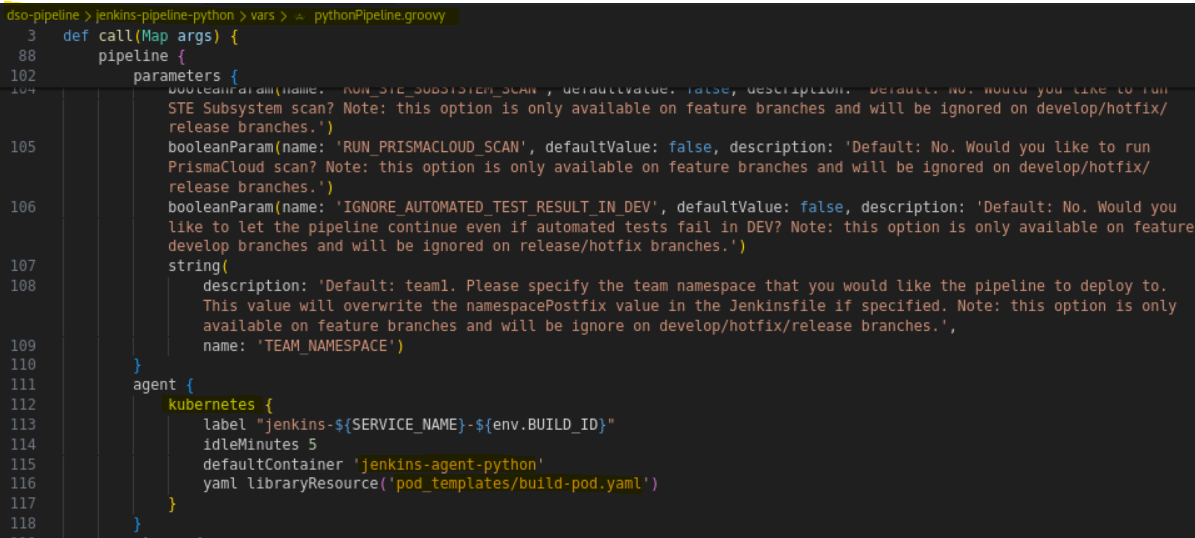
**-------------------------------------------------**

**Pipeline will set some Required fields & Optional parameters**

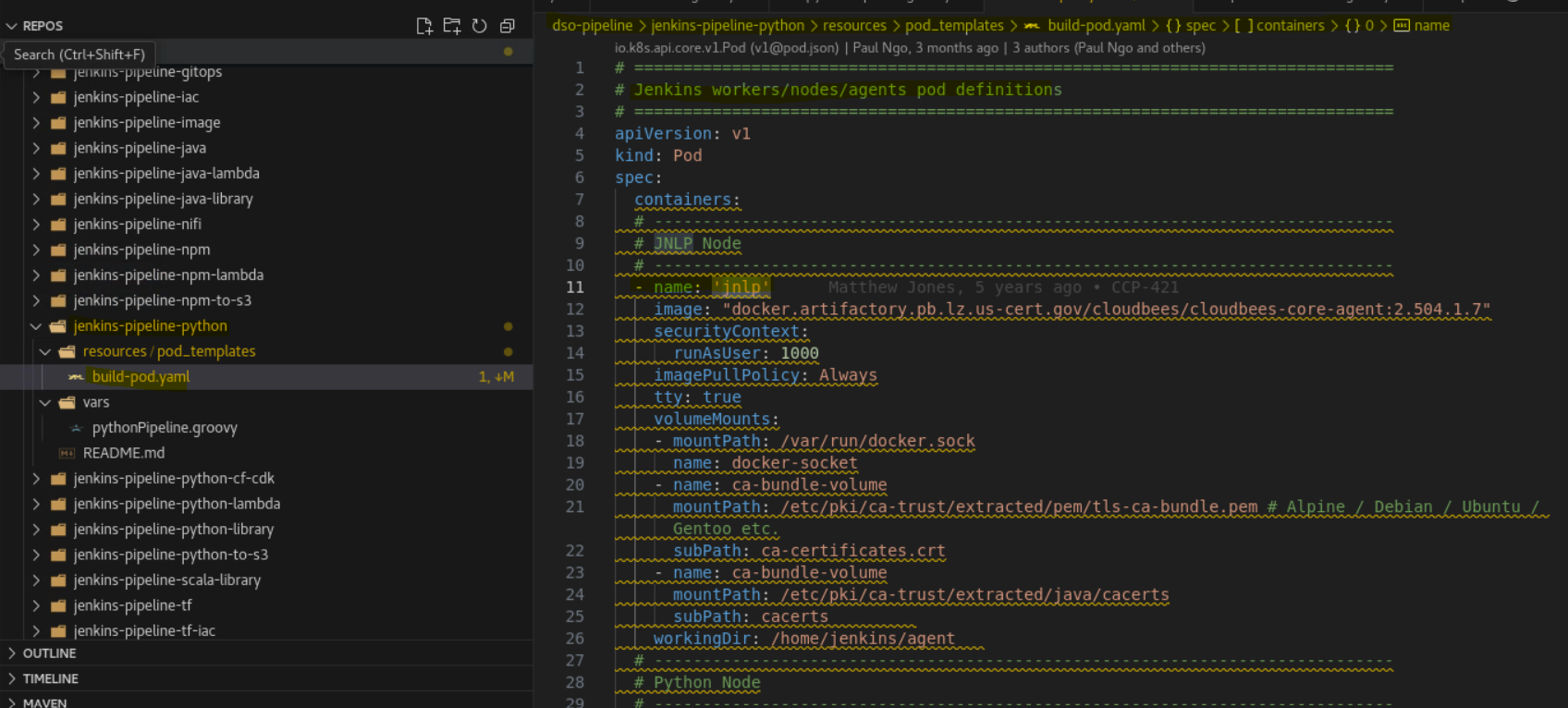
* + **Set some boolean Parameters for security scans**
  + **Default container** is **NPM** --> **jenkins-agent-npm**

**All of the following containers** are defined in the **Pod templates** which are called in **every pipeline. Like Python pipeline, Java Pipeline, NPM Pipeline, etc..**

* + Calling the **Pod Templates** inside **Python Pipeline** on **Kubernetes agent Container**.



* + **Pod Templates Configurations showing the list of all containers used** inside the **Python Pipeline.**



* + **All Containers:**

* + **jnlp agent container** --> it is **running** all the **plugins** and it is needed by **all pipelines** to run the plugins that we **install** on **every managed controller** in **Cloudbees**. Like **Twistlock plugin** that is used to scan all docker images using the **prismaCloudScanImage** function which is used by the plugin.
  + **Image mgmt agent container**  --> to **handle** the **interactions** with **Artifactory** like **Creating docker image and pulling image from Artifactory** and creating task for image tagging.
  + **kubernetes agent container**  --> to **handle** the **interactions** with **Kubernetes** which contains the **global-config** and secrets that we pull from pipeline-config namespace like **global-config ConfigMap** & **global-config Secrets**, which is used to pull all the required variables for all EKS clusters and Artifactory Credentials, Twistlock Credentials and EKS Cluster information for each tenant. Also, this agent is used for deployments to Kubernetes clusters.
  + **IAC agent container**  --> It is used for **Scanning tools** and installing tools like **terraform**.
  + **Mvn agent container** --> It is used to **Insert repos** for **Artifactory**
  + **Python container -->** is used for **Python code build** and **linting**.

* + **The pipeline must start with these branch names (release, develop, hotfix, feature, PR, pipeline)**
  + **These branch names must be is used to retrieve all the configmap of all the variables that are required to run the pipeline for each project**

- I have to show the **Kubernetes pipeline-config** namespace in **Kubernetes dashboard**.

* + Then, we go to the **BUILD STAGE**, which checks for the branch names

* + **In develop branch**, it will **always run** the **security scans**

* + **But** the **Feature branch** will **ONLY** run the **security scans** that I **select** while **running** the **pipeline**.
    - **IAC\_ENABLED: true**, this will **always run** in the **pipeline**, because it is set to "**true**" inside the **pipeline-config** namespace in **configmap**
    - in the **Security Scan**, it will **download** the **Checkmarkx docker image**

* + After it run the **Sonarqube scan**, it **publishes** a **report**.
  + Then, we go to the **Build NPM STAGE**,
    - Then, it has the **option** to **run** the **Unit Testing** under **SonarQube STAGE**, using "**npm run test**" **command**
    - Then, It will **build** the **docker image** using the **function** "**buildAndTagWithPodman**"
  + Then, it will Scan code using **twistlock**

* + **deployTarget Method** will do the **following**

* + It will load the configurations for testing and load the configuration for application that we are deploying
  + namespace to be deployed to.
  + Retag the image for based on the target environment (dev, test, staging, or production)
  + Then it deploy to the target environment
  + Also, It will have the option to run the automated tests.
  + Once it finish deployment, it will delete **some old images** to save **storage space in Artifactory.**
  + Then, It will build a custom report and post deploy notification email for users in the MOE.
  + All of this is controlled in Kubernetes pipeline-config namespace, which include a list of users that are allowed to receive email notifications.

* + Then, I need to show how the **deployEnv** works through a build, to show the **difference** between it and **DepoyTarget** function.

* + **CICD\_NAMESPACE** is used for **building** the **docker image** in the **dev namespace**
    - for **nifi** service, it will be **CICD namespace**

* + In **deploy target**, we **don't** actually **deploy**, instead we **update** the **helm charts**
  + **Our pipeline** do the **CI part**, and then the **deployment means update** the **HELM Chart** so that **ArgoCD** will **deploy** to the **target environment**
  + **ArgoCD** will **monitor** the **repository** for **yellowdog** every **5 mins**, and it will **deploy** if the **image version changed**.
  + Then, it makes a **commit** to the **helm chart repository**.
  + Check the **deployment ID** if it **matches,** it will **run** the **test**, **otherwise** it will **do** the **deployment** to the **intended namespace because** the **image version** has **changed**.
  + The **image** will be **retagged** from one **environment** to **another**, but the **image** will **always** be the **SAME one** that we **build** in the **Development environment**.
  + **Any variable** starts with "**input**." means its **value** will be **defined** in the **pipeline**.
  + Note that the **upper management** wants to see that the **image** used for **deploying** to **production** is the one **tagged** for "**PROD**" **environment**.

**========================================================================================================================================================================================================================================================================================**

**Helm Chart Deployment:**

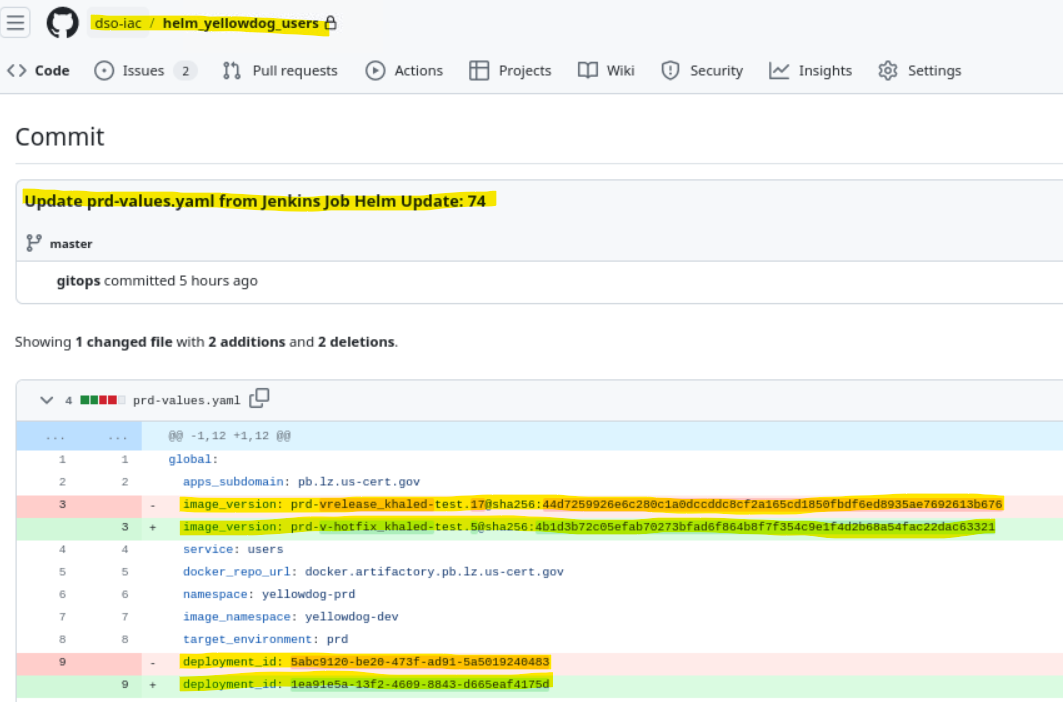
**------------------------------------**

**Deployments happen through committing a message to the infrastructure repository of the application that is being deployed**

**The file "prd-values.yaml" is updated with a new docker image version, and a new deployment\_id**

**ArgoCD monitors this repo every 5 mins, and it starts the deployment process to the intended namespace for this application (yellowdog-users) whenever it sees any change in the repository files. (Ex: prd-values.yaml)**

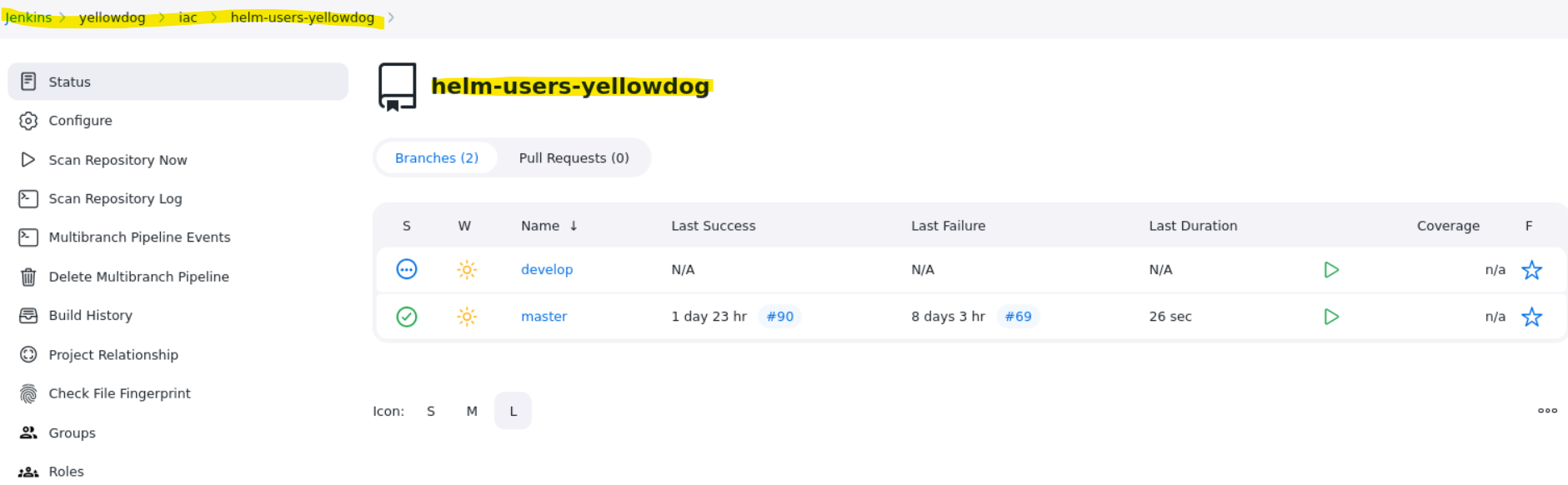
* + **Application Name: yellowdog-users**
  + **Repo Name: helm\_yellowdog\_users**
  + **Repo Link:** <https://git.pb.lz.us-cert.gov/dso-iac/helm_yellowdog_users/commit/6b01ebb82b35dba7bc4f073590e367f1e8909413>
  + **Updated File: prd-values.yaml**

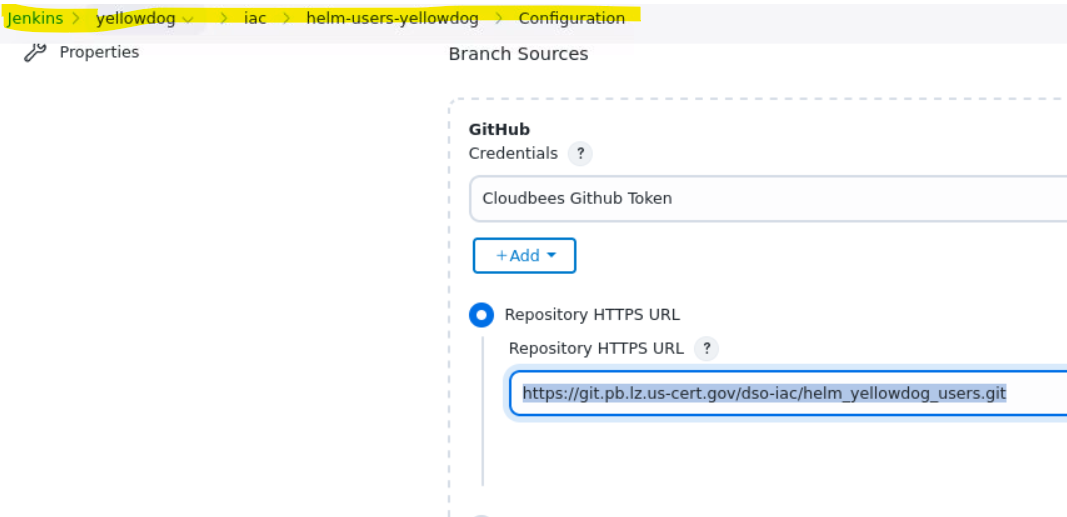


**Cloudbees Deployments:**

**-----------------------------------**

* + **Yellowdog --> iac --> helm-users-yeloowdog**
  + **This is mapped to the following git repository**
    - <https://git.pb.lz.us-cert.gov/dso-iac/helm_yellowdog_users.git>





**============================================================================================================================================**

**Trigger build Function:**

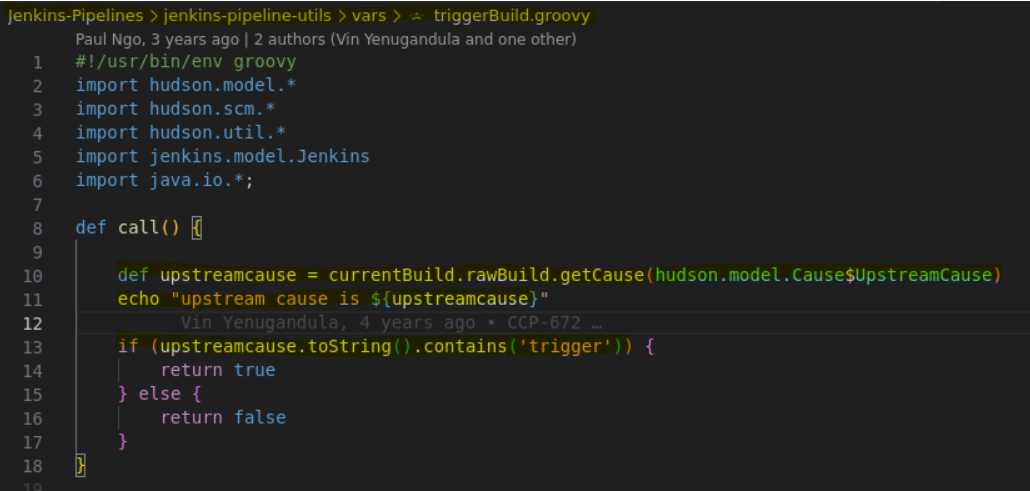
**---------------------------------**

**Function Name: triggerBuild**

**Location: Jenkins-utils --> triggerBuild.groovy**

**This function is looking for "trigger" keyword in the upstream cause in order to start the build.**

**If the "trigger" keyword is found, it will be "true" and the build will start accordingly.**

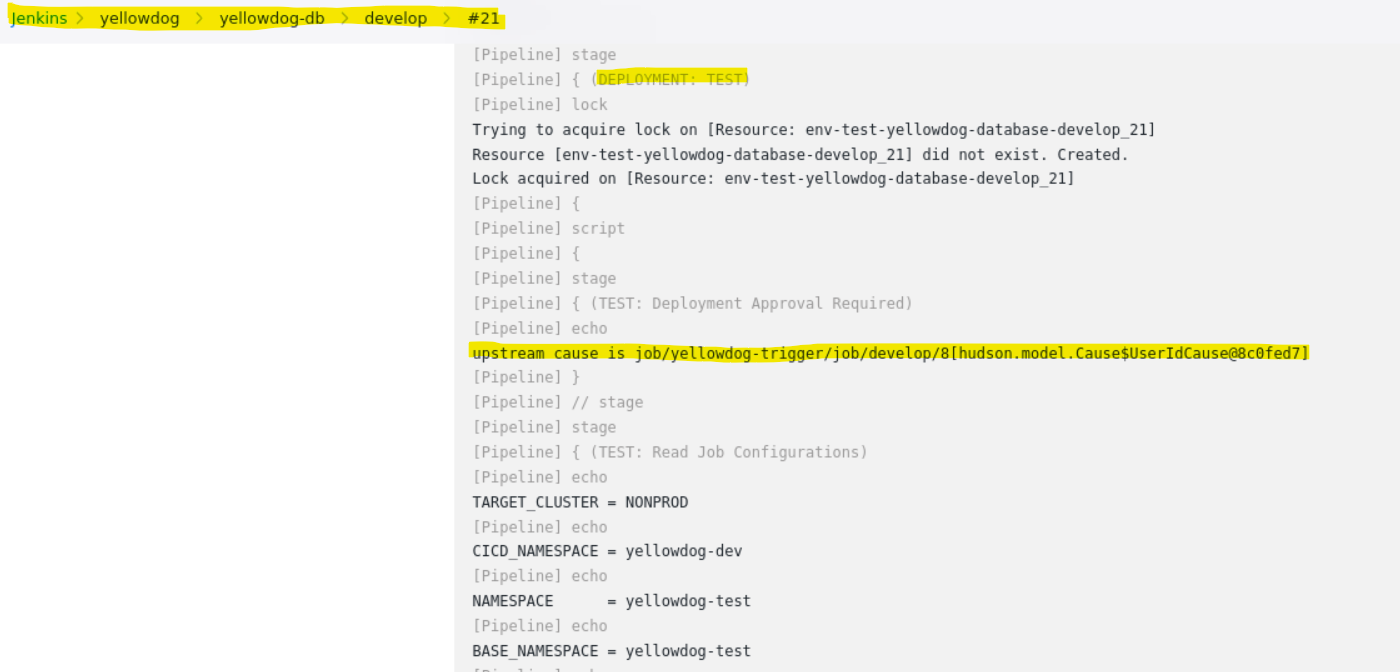


**Trigger for yellowdog Services:**

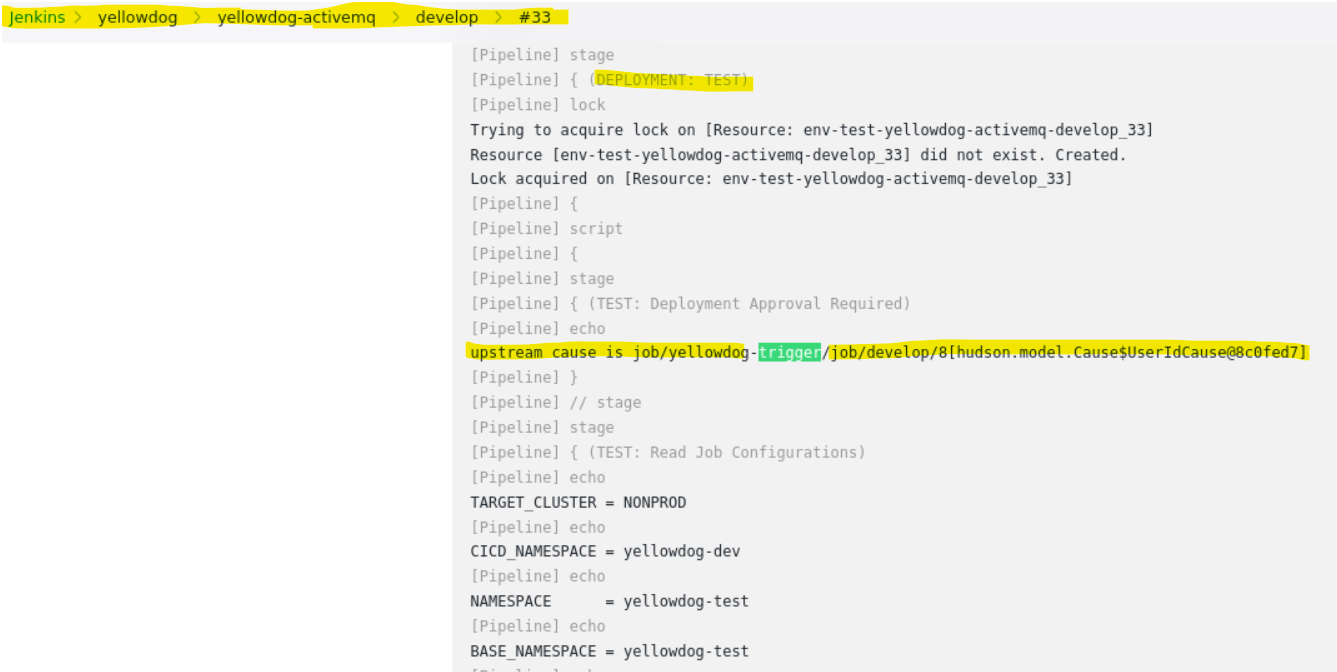
* + **Yellowdog-db**
  + **Yellowdog-activemq**



* + **Deployment of yellowdog-db in TEST environment.**



* + **Deployment of yellowdog-activemq in TEST environment.**



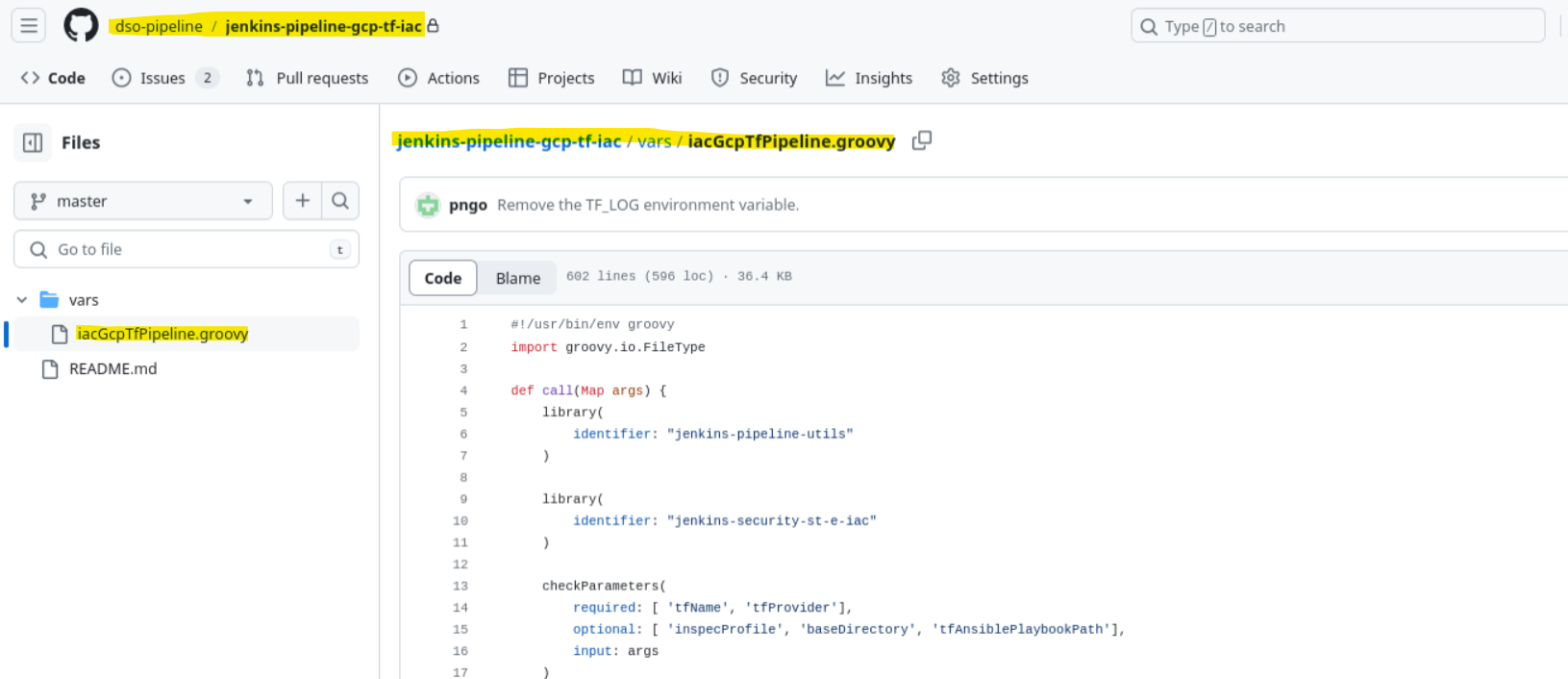
**============================================================================================================================================**

**Google Cloud Terraform Pipeline:**

**----------------------------------------------**

**Repo: dso-pipeline**

**Location:** <https://git.pb.lz.us-cert.gov/dso-pipeline/jenkins-pipeline-gcp-tf-iac/blob/master/vars/iacGcpTfPipeline.groovy>



**============================================================================================================================================**

**Azure Cloud Terraform Pipeline:**

**----------------------------------------------**

**Repo: dso-pipeline**

**Location:** <https://git.pb.lz.us-cert.gov/dso-pipeline/jenkins-pipeline-azure-tf-iac/blob/master/vars/iacAzureTfPipeline.groovy>

