DevOps Run book

For TIbco Mattel Consumer Master Project

CI/CD for Microservices

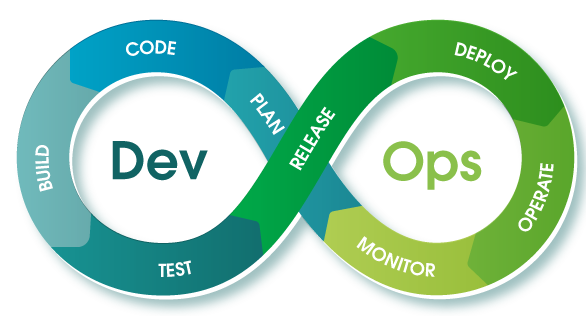


TABLE OF CONTENTS

**1. INTRODUCTION …………………………………………………………………………………………………………………………. 3**

**2. PRE REQUISITIES…………………………………………………………………………………………………………………………. 3**

2.1. ACCESS REQUESTS**…………………………………………………………………………………………………………………. 3**

2.2. GCP CONFIGURATION **……………………………………………………………………………………………………………. 3**

2.3. SOFTWARE REQUIREMENTS **…………………………………………………………………………………………………… 5**

2.4. Best Practices Incorporated **……………………………………………………………………………………………………. 5**

2.5. GITHUB Configuration **……………………………………………………………………………………………………………. 5**

2.6. Jenkins Configuration **…………………………………………………………………………………………………………….. 6**

*2.6.1. Plugins to Be Installed* **……………………………………………………………………………………………………. 7**

*2.6.2. Credentials* **……………………………………………………………………………………………………………………. 7**

*2.6.3 Pipeline Specific Access to team members* ***……………………………………………………………………… 7***

**3. IMPLEMENTATION ………………………………………………………………………………………………………………………. 8**

3.1. Pipeline for DEV Environment **………………………………………………………………………………………………... 8**

*3.1.1. FILES TO BE CHECKED* **…………………………………………………………………………………………………….. 8**

*3.1.2. PIPELINE CONFIGURATION* **…………………………………………………………………………………………….. 9**

*3.1.3. CONSTRAINTS TO CREATE A TAG* **……………………………………………………………………………………..9**

*3.1.4. Constraints that are to be taken care when a new service is been added* **…………………………..9**

*3.1.5. Steps to trigger a build* **…………………………………………………………………………………………………….9**

*3.1.6. Workflow* **………………………………………………………………………………………………………………………10**

*3.1.7. Rollback Mechanism* **………………………………………………………………………………………………………10**

3.2. Pipeline for QA Environment **…………………………………………………………………………………………………12**

*3.2.1. FILES TO BE CHECKED* **…………………………………………………………………………………………………….12**

*3.2.2. PIPELINE CONFIGURATION* **…………………………………………………………………………………………....12**

*3.2.3. Steps to trigger a build* **…………………………………………………………………………………………………..13**

*3.2.4. Workflow* **………………………………………………………………………………………………………………………14**

*3.2.5**Rollback Mechanism* **………………………………………………………………………………………………………14**

3.3. Pipeline for PRE\_PROD Environment **…………………………………………………………………………………….14**

*3.3.1. FILES TO BE CHECKED* **…………………………………………………………………………………………………….14**

*3.3.2. PIPELINE CONFIGURATION* **…………………………………………………………………………………………….15**

*3.3.3. Steps to trigger a build* **…………………………………………………………………………………………………..15**

*3.3.4. Workflow* **………………………………………………………………………………………………………………………16**

*3.3.5**Rollback Mechanism* **……………………………………………………………………………………………………..16**

3.4. Pipeline for PROD Environment **…………………………………………………………………………………………….16**

*3.4.1. FILES TO BE CHECKED* **…………………………………………………………………………………………………….17**

*3.4.2. PIPELINE CONFIGURATION* **…………………………………………………………………………………………….17**

*3.4.3. Steps to trigger a build* **…………………………………………………………………………………………………..17**

*3.4.4. Workflow* **………………………………………………………………………………………………………………………18**

*3.3.5**Rollback Mechanism* **……………………………………………………………………………………………………..16**

3.5**.** CHANGES NEEDED IN FUTURE FOR ADDING NEW APIS IN THE SYSTEM ……………………………………**20**

3.6 TROUBLESHOOTING IN CASE OF ANY ISSUE ………………………………………………………………………………**21**

# **INTRODUCTION**

This document is intended to serve as the basis for creating run book for Jenkins pipeline for TIBCO microservices for Mattel.

The document explains how one can build and deploy to cloud microservices via Jenkins CI/CD pipeline. The documents covers all the pre-requisites needed for creating CI/CD pipelines, on a GCP compute instance.

# **PRE REQUISITIES**

**2.1. ACCESS REQUESTS**

The below access are required to view, trigger and monitor the deployments using the DevOps process.

1. Google Cloud Platform (GCP) access - <https://console.cloud.google.com> (access to be provided to your GCP account created by Mattel email id)

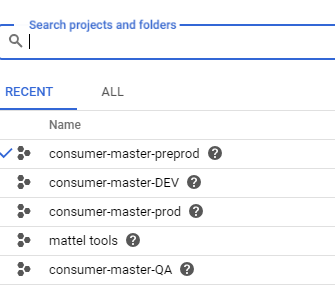
* Projects Access:
  + - consumer-master-preprod
    - consumer-master-prod
    - consumer-master-QA
    - consumer-master-DEV
    - mattel tools

1. GitHub access - <https://github.com/mattel-dig/ConsumerMaster--GSL-> (Microservice source code repository)
2. Jenkins console access - <http://35.224.202.169:8080/>(Jenkins to be installed in build server)

**2.2. GCP CONFIGURATION**

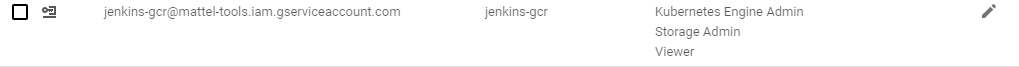
* Account in GCP –Login to GCP via the Mattel email ids
* Get the Access Permissions to the Projects in GCP

(If a new project to be created check with the Contact specified below)



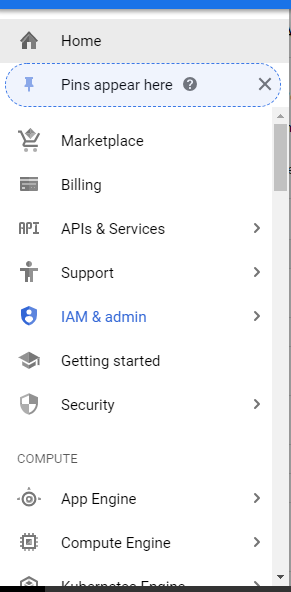
* Service accounts for GCR of each Project (environment) should be created

1. Service account to access Mattel Tools project is configured in Jenkins.



1. Service account to access GitHub to checkout the code is configured as well.

* Kubernates Admin Access, Storage Viewer ,Storage Admin - “IAM Role and Access” should be provided to each project (environment) in regards to the generic project(Mattel tools)

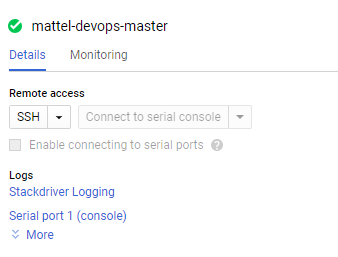


[Contact [Prasad.Gavali@cognizant.com](mailto:Prasad.Gavali@cognizant.com) ]

**2.3. SOFTWARE REQUIREMENTS**

**Mattel tools (Generic Project) - Build Server**

The entire DevOps project is present in [mattel-devops-master](https://console.cloud.google.com/compute/instancesDetail/zones/us-central1-a/instances/mattel-devops-master?project=mattel-tools) instance, which is present in Mattel Tools project, in GCP.



Below are the software installations done in the mattel-devops-master instance.

|  |  |  |
| --- | --- | --- |
| No | Software | Versions |
| 1 | Jenkins | 2.164.1 |
| 2 | Java | 1.8.0\_212 |
| 3 | Maven | 3.6.0 |
| 4 | Docker | 18.09.6 |
| 5 | Tibco-maven-plugin | bw6-maven-plugin 6.5 |
| 6 | BART | 1.0 |

**2.4. Best Practices Incorporated**

* Build Once Multiple Deploy
* Tag Based Deployment
* Quality Gates based on the code quality
* Pipeline as code based framework
* Generic pipeline for all micro services/APIs
* Requirement traceability based tagging

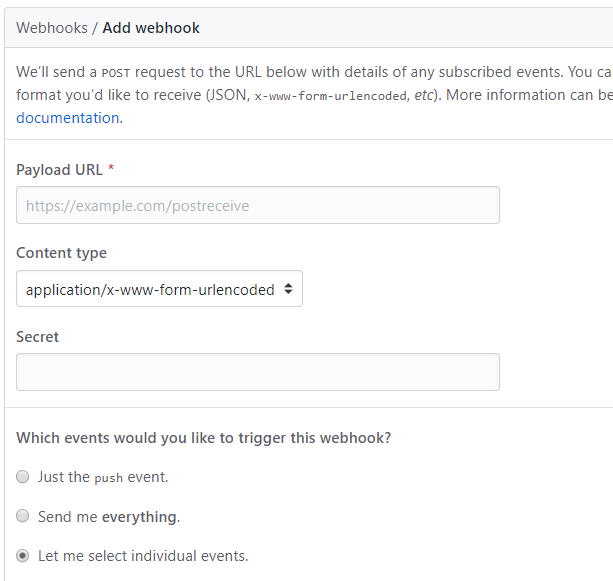
**2.5. GitHub Configuration**

* GitHub repository: <https://github.com/mattel-dig/ConsumerMaster--GSL-.git>
* Branches :master,release,dev
* “GitHub-hook trigger” in above-mentioned repository for GITScm polling is configured in GitHub.

1. Under Settings 🡪Webhooks
2. The **payload url** : <http://35.224.202.169:8080/github-webhook/> is provided

**Content Type**: “application/json” is selected

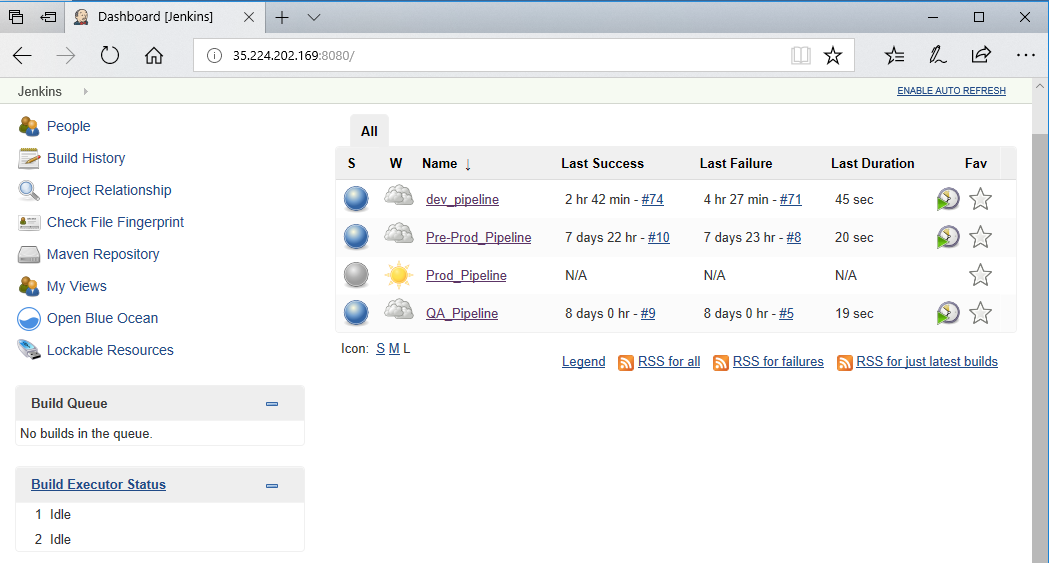
1. **Which events would you like to trigger this webhook? Is selected**



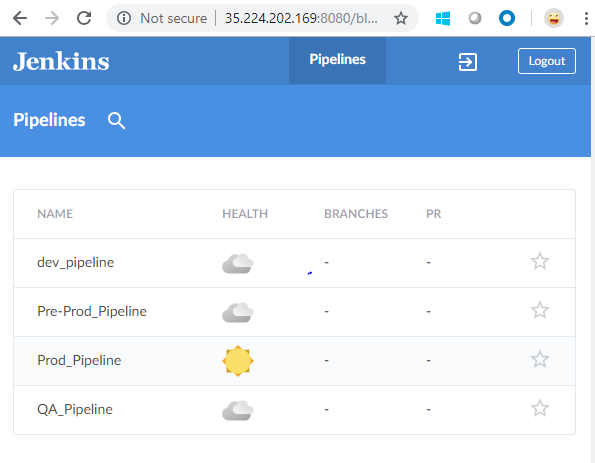
**2.6. Jenkins Configuration**

Login via Jenkins credentials [Already provided] (<http://35.224.202.169:8080>)

Once you had logged in to the Jenkins console using your credentials, you will be able to see the below page. Totally, there are four pipeline jobs present in the Jenkins console.



In order to view the various stages of the pipeline and it progress we need to select the option “Open Blue Ocean” in the left pane.



**2.6.1. Plugins To be installed**

**(Manage Jenkins>>Manage Plugins)**

* Google OAuth Credentials Plugin
* Google Container Registry Auth Plugin
* Build Pipeline Plugin
* Maven Integration plugins
* Pipeline maven integration

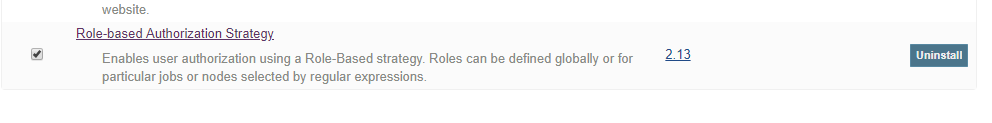
**2.6.2. Credentials**

* The GitHub credentials in Jenkins ( Credentials>>Add New) is configured
* The mattel-tools project service account credentials is configured

**2.6.3. Pipeline Specific Access to Team Members**

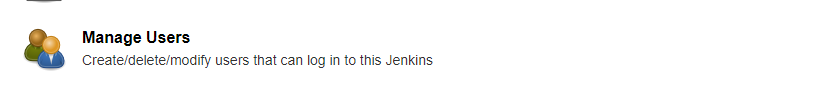
Role Based Strategy is used to give pipeline specific access to Team Members

Hence, Role based Strategy plugin in installed under “Manage Plugins”

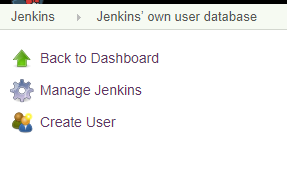


**For new user,**

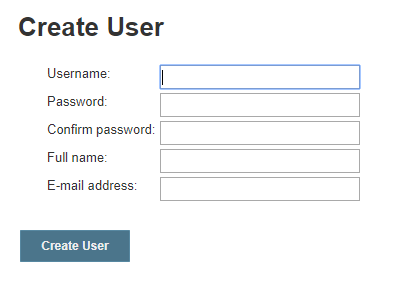
* Create user accounts in Jenkins by clicking “**Manage Users**“ under “**Manage Jenkins**” Tab



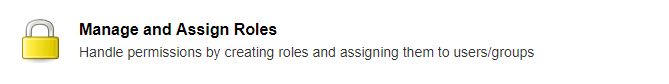
* Click “Create User “ to add the new user



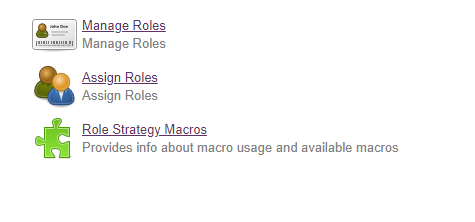
* Populate the Create User Form with necessary details



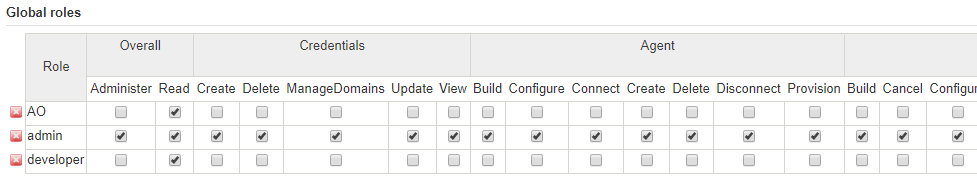
* Click “ Manage and Assign Roles “ under “Manage Jenkins” (appears only after the plugin is installed)



* Click Manage Roles to assign Global roles and Project Roles

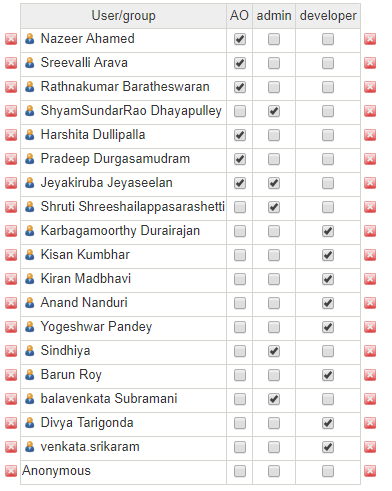


* Under Global Roles :Admin , Developer and AO roles are specified

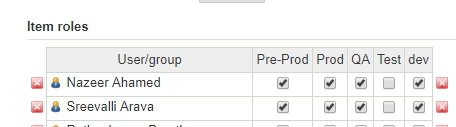


* Click Assign Roles to provide global roles to users added

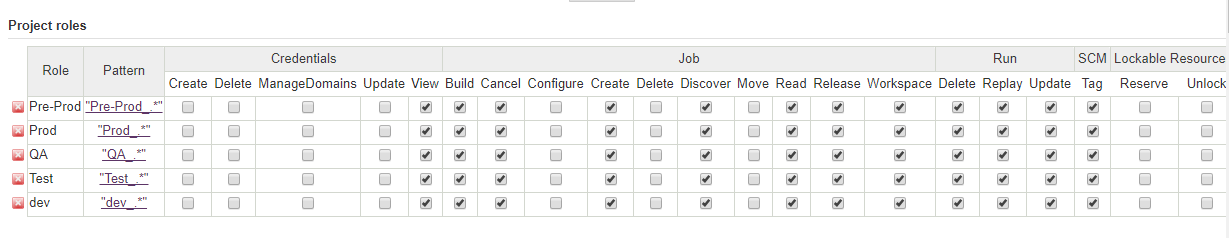
Check the team under which the new user should be added.



* Under item roles, check the items to which the access should be provided for that particular user.



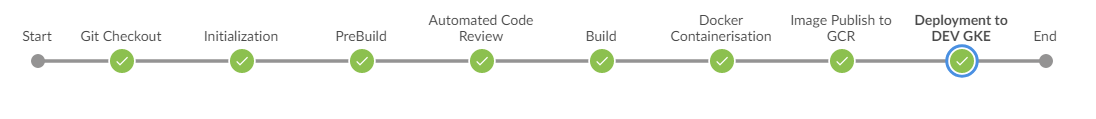
* If a new project or pipeline needed to be added
* Click Manage Roles .Under Project roles add the patterns to access the jobs in Jenkins eg: dev\_.\*



# **IMPLEMENTATION**

**3.1 Pipeline for DEV Environment:**

The dev pipeline has a tag based GitHub webhook trigger setup in it. Whenever a tag is created in the GitHub, automatically the pipeline is triggered the Jenkins. Only this pipeline will fetch the code from GitHub and undergoes a Bart code review process and code is build using Maven.

**Open Blue Ocean to view the entire pipeline:**

**3.1.1. FILES TO BE CHECKED:**

* Dockerfile for all microservices & API are placed under deploy\_rearch/dockerfiles [Eg:/API/consumeraddress/Dockerfile,/Microservice/consumeraddress-ms/Dockerfile]

Provide the ear file name with version 1.0.0 always

* Deployment.yml files for all microservices & APIs are to be placed under deploy\_rearch/manifest.yml/DEV

[eg: consumeraddress\_dev.yml,consumeraddress-ms\_dev.yml]

Provide the image name as gcr.io/mattel-tools/<microservice\_deployment\_name>:latest

* Code for microservices & APIs are placed under code-rearch/ (Note: Code to be pushed with the pom.xml files)

[Note: code follows a folder structure which includes Microservice ,API , SharedModules folders having their respective code]

* Propertiesfile is placed under /propertiesfile/dev.properties

[Note: This file contains the variables zone, name, region of Dev environment and the names of the microservice’s/API’S

Eg: consumeraddress-api=Mattel.CM.ConsumerAddress.API.application

consumeraddress-ms= Mattel.CM.ConsumerAddress.MicroService.application]

On Developing, a new microservice /API just add the name of the respective service in the propertiesfile and follow the other naming constraints.

**3.1.2. PIPELINE CONFIGURATION:**

* + Under Build Trigger, “GitHub hook trigger for GITScm polling” is checked.
  + Since the pipeline groovy is placed in the GitHub ,Pipeline Script from SCM under pipeline Definitions is selected
  + The GitHub URL is provided and its respective credentials configured under Credentials is selected.
  + Under Advanced : “+refs/tags/\*:refs/remotes/origin/tags/\*” is added to RefSpecs
  + Under Branches to Build : refs/tags/\*\_dev\_\* is provided for Branch Specifier (Note: Only the Commits with tags matching the above pattern triggers the build)
  + Finally, the Path to the Jenkinsfile (groovy script) is provided.

**3.1.3. CONSTRAINTS TO CREATE A TAG:**

* Tag has to be created, only when there is a need to trigger the build.
* After committing the changes to the GITHUB, one need create the tag with the service name followed by “\_dev\_” and then the comment, which they would like to add. One needs to follow the below naming convention:

<microservice/api name>\_dev\_<your additional comment>

Eg: consumeraddress-ms\_dev\_Defect1, consumerchild\_dev\_Defect1

* The microservice/API name should always be in small letters, as the service name for the deployment is fetched only from the tag.
* On tagging, the build is triggered in Jenkins based on GitHub hook trigger for GITScm polling.
* The build gets triggered only if the tag pushed matches with the pattern “refs/tags/\*\_dev\_\*” [e.g:consumeraddress\_dev\_Defect1]
* Only the tag to the new commit triggers a build
* Build does not trigger for commits without tags

**3.1.4.CONSTRAINTS THAT ARE TO BE TAKEN CARE WHEN A NEW SERVICE IS BEEN ADDED**

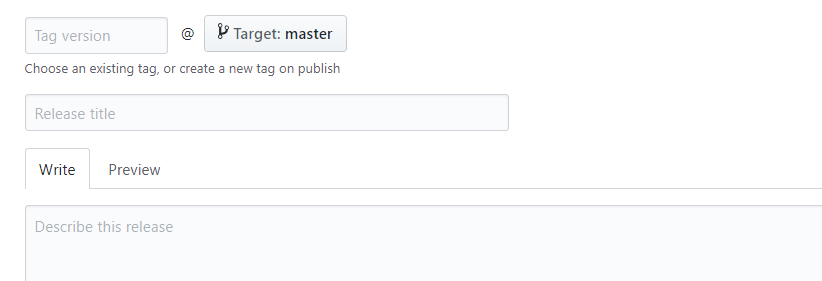
* Add the name of the respective service in the properties file by following the above-mentioned naming conventions.
* Add its Dockerfile, deployment yml files under above mentioned respective folder

**3.1.5. STEPS TO TRIGGER BUILD**

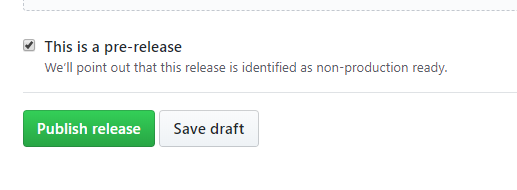
1. Commit the code to GitHub repository
2. Login to GitHub
3. Select dev under branches
4. Click the releases tab and under releases click “Draft a new release”

Now provide the tag and select the branch dev. Following the naming conventions on tagging is mandatory : i.e. <microservice/api name>\_dev\_<your additional comment>

Eg: consumeraddress-ms\_dev\_Defect1, consumerchild\_dev\_Defect1



1. Check “This is a pre-release” and click Publish release



1. The build triggers in Jenkins ( dev\_pipeline job )
2. Login to Jenkins and click dev\_pipeline job to check the triggered build
3. Click on the build number to view the logs (On Failure ,Check the output logs for the error statement.)

**3.1.6. WORKFLOW:**

* 1. **Code pull (GITHUB)**
  2. Tibco Micro service Developers commits the application code to the dev branch of ConsumerMaster—GSL GitHub repository.
  3. On tagging, the build is triggered in Jenkins based on GitHub hook trigger for GITScm polling.
  4. The build triggers only if the tag pushed matches with the pattern “refs/tags/\*\_dev\_\*” [e.g:consumeraddress\_dev\_Defect1]

(Note: Build does not trigger for commits without tags)

* 1. Jenkins pulls the code into its workspace from the Specified GitHub Repository.
  2. The microservice/API name is fetched from the tag pushed under “SERVICE\_NAME”
  3. **Pre Build Steps**

1. Micro services are mapped to its respective dependent Shared Modules
2. ReadMe.txt of a particular microservice is read and the dependent Shared modules mentioned in the ReadMe.txt are copied into a temporary folder. The pom.xml of these shared modules are modified by replacing the parent tag to point to the microservice parent folder.
   1. **Code Analysis (BART)**
      1. Code Analysis is Done through Bart plugin
      2. ERROR and WARNING threshold limits are set in Jenkins
      3. The html report is generated into the Jenkins workspace on the code analysis
      4. On threshold check :
         1. When error and warning count is below the threshold limit , pipeline proceeds to maven build
         2. On Failure in meeting the threshold , the build breaks

v. The html report build during BART analysis is pushed to GitHub repository ([ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/[deploy](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/deploy)/**Bart\_report**/)

* 1. **Build (MAVEN)**

1. Maven Builds the TIBCO micro service using the pom.xml file present in the
2. Parent folder of microservice.
3. Ear file is generated in the target folder of the micro service.
4. The Ear file built is pushed to GitHub repository ([ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/[deploy](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/deploy)/**Earfiles**/)
5. The BART file built is pushed to GitHub repository ([ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/[deploy](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/deploy)/**Bart\_report**/)
   1. **Containerization (DOCKER)**
      1. **Image Build via Docker File**
         1. The Dockerfile is copied into ear file generated location and the image is built.
         2. The image is tagged with the GitHub commit id of the respective repository
      2. **Image push to GCR**
         1. The Docker image built is been pushed to the GCR in Mattel-tools (Build server) project.
   2. **Deployment Orchestration to Dev GKE**
   3. The Dev environment is logged in via the Mattel-tools (Build server) service account and its credentials.
   4. Gcloud config is configured to DEV project to access its respective kubernates.
   5. Deployment.yml file is used to deploy to the GKE cluster in DEV(Deployment.yml are environment specific)
   6. Before deployment, the “latest” tag inside the respective deployment.yml file is replaced with the “IMAGE\_TAG”
   7. The existence of deployment is checked. If deployment is not present create a new deployment else overwrite the existing deployment.
   8. This file pulls the image tagged with the commit id from Mattel-tools (Build server) GCR and deploys into the DEV GKE cluster specified.

**3.2.5.ROLLBACK MECHANISM:**

Trigger the build by passing the previous successfully deployed image. The image is obtained

from dev\_images folder of “Mattel-tools” GCR.

* 1. **Pipeline for QA Environment:**

Unlike dev pipeline, QA pipeline is a parameterized pipeline. One need to pass the required parameters before triggering the job. This pipeline is a manually triggered pipeline.

**3.2.1. FILES TO BE CHECKED:**

* Deployment.yml files for all micro services & APIs are placed under deploy\_rearch/manifest.yml/QA

Provide the image name as gcr.io/qa\_images/mattel-tools/<microservice\_deployment\_name>:latest

* Jenkinsfile for QA is placed under /deploy\_rearch/groovy\_scripts/QA/qa\_pipeline.groovy
* propertiesfile is placed under /propertiesfile/gcloud.properties

[Note: This file contains the variables zone, name, region of QA environment and the names of the microservice’s/API’S

**3.2.2. PIPELINE CONFIGURATION:**

* + “This project is parameterized” is selected . IMAGE\_TAG and SERVICE\_NAME are passed as parameters.
  + The GitHub URL is provided and its respective credentials configured under Credentials is selected.
  + Under Branches to Build : /\*dev is provided for Branch Specifier
  + Finally, the Path to the Jenkinsfile (groovy script) is provided.

**3.2.3. STEPS TO TRIGGER BUILD**

1. Login to Jenkins via credentials
2. Click on QA\_pipeline job
3. Click “Build with Parameters” and pass the parameters respective to the microservice to be deployed
4. The parameters to be passed:

**IMAGE\_TAG**: Tag of the microservice image deployed successfully in dev environment

* Get the latest image tag from GCR in GCP under respective microservice

Select “mattel-tools” Project under projects tab🡪Click Container Registry🡪Click the respective microservice repository(eg:cm-consumeraddress)🡪get the latest image tag

OR

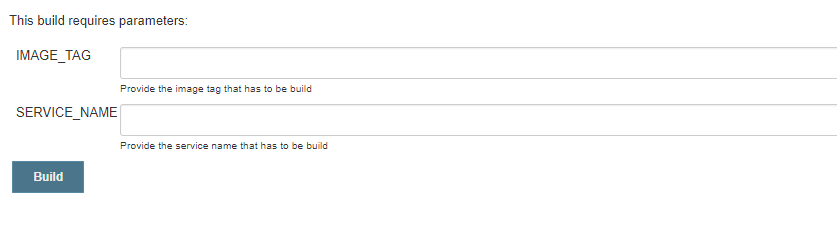
* Get the image tag from the GKE in GCP under the respective microservice deployment

Select “consumer-master-DEV” Project under projects tab🡪Click Kubernates Engine 🡪Under Workloads Click the respective microservice deployment(eg:cm-consumeraddress)🡪under active revisions get the image tag(highlited)



**SERVICE\_NAME**: Respective Microservice/API Name to be deployed in QA ( Provide in lowercase e.g. consumeraddress)

1. Click “Build” Button and the job is started.



**3.2.4. WORKFLOW:**

**a. Image Push to QA folder in GCR**

1. The dev image with the “IMAGE\_TAG” as its tag and “SERVICE\_NAME” as its Image name is now pushed into qa\_images folder of Mattel-tools GCR

**b. Deployment Orchestration to QA GKE**

1. The environment is logged in via the Mattel-tools (Build server) service account and its credentials.
2. Gcloud config command is used to switch to QA project in order to access its respective kubernates.
3. Deployment.yml file is used to deploy to the GKE cluster in QA(Deployment.yml are environment specific)
4. Before deployment, the “latest” tag inside the respective deployment.yml file is replaced with the “IMAGE\_TAG”
5. The existence of deployment is checked. If deployment is not present create a new deployment else overwrite the existing deployment
6. This file pulls the image with tag as “IMAGE\_TAG” from its respective microservice/API under qa\_images folder from Mattel-tools (Build server) GCR and deploys into the QA GKE cluster specified.

**3.2.5.ROLLBACK MECHANISM:**

Trigger the build by passing the previous successfully deployed image. The image is obtained

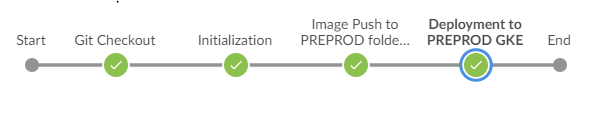
from qa\_images folder of “Mattel-tools” GCR.

Follow the steps mentioned in **3.2.3**

* 1. **Pipeline for PRE\_PROD Environment:**

PRE\_PROD pipeline is a build with parameters job. One need to pass the required parameters and then trigger the build. This pipeline is a manually triggered pipeline.

**Open Blue Ocean to view the entire pipeline:**



**3.3.1. FILES TO BE CHECKED:**

* Deployment.yml files for all microservices & APIs are placed under deploy\_rearch/manifest.yml/PREP

Provide the image name as gcr.io/preprod\_images/mattel-tools/<microservice\_deployment\_name>:latest

* Jenkinsfile for PRE\_PROFD is placed under /deploy\_rearch/groovy\_scripts/Pre-Prod/pre-prod\_pipeline.groovy
* propertiesfile is placed under /propertiesfile/gcloud.properties

**3.3.2. PIPELINE CONFIGURATION:**

* + “This project is parameterized” is selected . IMAGE\_TAG and SERVICE\_NAME are passed as parameters.
  + The GitHub URL is provided and its respective credentials configured under Credentials is selected.
  + Under Branches to Build : /\*dev is provided for Branch Specifier
  + Finally, the Path to the Jenkinsfile (groovy script) is provided.

**3.3.3. STEPS TO TRIGGER BUILD**

1. Login to Jenkins via credentials
2. Click on Pre-Prod\_pipeline job
3. Click Build with Parameters and pass the parameters respective to the microservice to be deployed
4. The parameters to be passed:

IMAGE\_TAG: Tag of the microservice image deployed successfully in qa environment

* Get the latest image tag from GCR in GCP under respective microservice

Select “mattel-tools” Project under projects tab🡪Click Container Registry🡪Click qa\_images🡪Click the respective microservice repository(eg:cm-consumeraddress)🡪Get the image tag



OR

* Get the image tag from the GKE in GCP under the respective microservice deployment

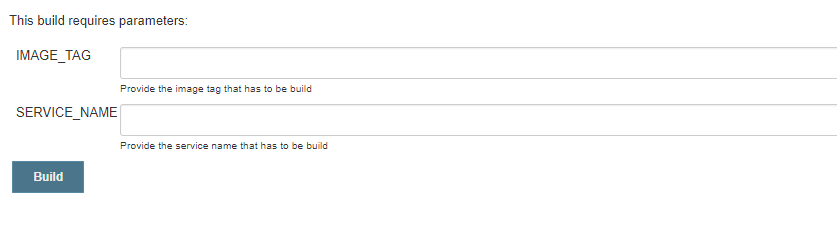
Select “consumer-master-preprod” Project under projects tab🡪Click Kubernates Engine 🡪Under Workloads,Click the respective microservice deployment

(eg:cm-consumeraddress🡪Under Active revisions tab get the image tag(highlighted)



**SERVICE\_NAME**: Respective Microservice/API Name to be deployed in Pre-Prod (Provide in lowercase e.g. consumeraddress)

1. Click “Build” Button and the job is triggered.



**3.3.4. WORKFLOW:**

**a. Image Pushed to PRE\_PROD folder in GCR**

1. The qa image with the “IMAGE\_TAG” as its tag and “SERVICE\_NAME” as its Image name is now pushed into pre-prod\_images folder of Mattel-tools GCR

**b.Deployment Orchestration to PRE\_PROD GKE**

1. The environment is logged in via the Mattel-tools (Build server) service account and its credentials.
2. Gcloud config command is used to switch to Pre-Prod project in order to access its respective kubernates.
3. Deployment.yml file is used to deploy to the GKE cluster in PRE\_PROD (Deployment.yml are environment specific)
4. Before deployment, the “latest” tag inside the respective deployment.yml file is replaced with the “IMAGE\_TAG”
5. The existence of deployment is checked. If deployment is not present create a new deployment else overwrite the existing deployment
6. This file pulls the image with tag as “IMAGE\_TAG” from its respective microservice/API under pre-prod\_images folder from Mattel-tools (Build server) GCR and deploys into the PRE\_PRODGKE cluster specified.

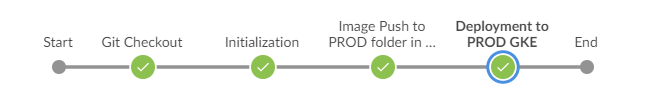
**3.3.5.ROLLBACK MECHANISM:**

Trigger the build by passing the previous successfully deployed image. The image is obtained from pre-prod\_images folder of “Mattel-tools” GCR.

Follow the steps mentioned in **3.3.3**

* 1. **Pipeline for PROD Environment:**

PROD pipeline is a build with parameters job. When the parameters are passed the build is triggered in Jenkins. This pipeline is a manually triggered pipeline.



**3.4.1. FILES TO BE CHECKED:**

* Deployment.yml files for all microservices & APIs are placed in master branch under the location “deploy\_rearch/manifest.yml/PROD”

Always the image name as to be provided in the below format in the Prod yml files. “gcr.io/prod\_images/mattel-tools/<microservice\_deployment\_name>:latest”

* Jenkinsfile for PROD is placed under /deploy\_rearch/groovy\_scripts/prod\_pipeline.groovy in master branch
* propertiesfile is placed under /propertiesfile/gcloud.properties in master branch

**3.4.2. CONFIGURE PIPELINE:**

* + “This project is parameterized” is selected . IMAGE\_TAG and SERVICE\_NAME are passed as parameters.
  + The GitHub URL is provided and its respective credentials configured under Credentials is selected.
  + Under Branches to Build : /\*dev is provided for Branch Specifier
  + Finally, the Path to the Jenkinsfile (groovy script) is provided.

**3.4.3. STEPS TO TRIGGER BUILD**

1. Login to Jenkins via credentials
2. Click on Prod\_pipeline job
3. Click Build with Parameters and pass the parameters respective to the microservice to be deployed
4. The parameters to be passed:

**IMAGE\_TAG**: Tag of the microservice image deployed successfully in Pre-Prod environment

* Get the latest image tag from GCR in GCP under respective microservice

Select “mattel-tools” Project under projects tab🡪Click Container Registry🡪Click preprod\_images🡪Click the respective microservice repository(eg:cm-consumeraddress)🡪Get the image tag



OR

* Get the image tag from the GKE in GCP under the respective microservice deployment

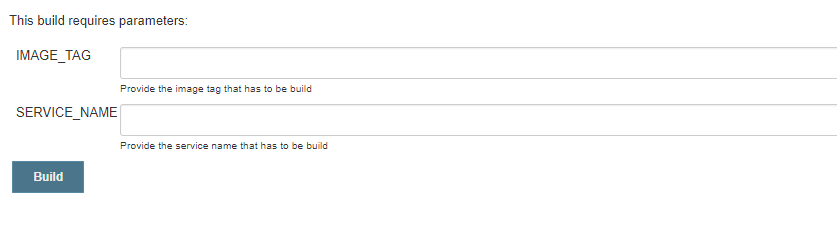
Select “consumer-master-QA” Project under projects tab🡪Click Kubernates Engine 🡪Under Workloads,Click the respective microservice deployment

(eg:cm-consumeraddress)🡪Under Active revisions tab get the image tag(highlighted)



**SERVICE\_NAME**: Respective Microservice/API Name to be deployed in Prod (Provide in lowercase e.g. consumeraddress)

1. Click “Build” Button and the job starts .



**3.4.4. WORKFLOW:**

* 1. **Image Pushed to PROD folder in GCR**
* The dev image with the “IMAGE\_TAG” as its tag and “SERVICE\_NAME” as its Image name is now pushed into prod\_images folder of Mattel-tools GCR
  1. **Deployment Orchestration to PROD GKE**
* The environment is logged in via the Mattel-tools (Build server) service account and its credentials.
* Gcloud config command is used to switch to Prod project in order to access its respective kubernates.
* Deployment.yml file is used to deploy to the GKE cluster in PROD (Deployment.yml are environment specific)
* Before deployment, the “latest” tag inside the respective deployment.yml file is replaced with the “IMAGE\_TAG”
* The existence of deployment is checked. If deployment is not present create a new deployment else overwrite the existing deployment
* This file pulls the image with tag as “IMAGE\_TAG” from its respective microservice/API under prod\_images folder from Mattel-tools (Build server) GCR and deploys into the PRODGKE cluster specified.

**3.4.5.ROLLBACK MECHANISM:**

Trigger the build by passing the previous successfully deployed image.The image is obtained from prod\_images folder of “Mattel-tools” GCR.

Follow the steps mentioned in **3.4.3**

**3.5 CHANGES NEEDED IN FUTURE FOR ADDING NEW APIS IN THE SYSTEM:**

Below are the changes to be incorporated on adding a new API or Microservice,

* **SourceCode** :

The code of the API and Microservice should be placed under [ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/**code\_rearch**/API and [ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/**code\_rearch**/Microservice respectively.

Make sure the code is pushed with pom.xml files

* **Properties file:**

Add the names of the API or Microservice in the dev.properties file

[ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/[propertiesfile](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/propertiesfile)/**dev.properties**

Eg: consumerinfo-api=Mattel.CM.ConsumerInfo.API.application

consumerinfo-ms=Mattel.CM.ConsumerInfo.MicroService.application

[It is preferable to maintain Camel Case “MicroService “]

* **Dockerfile:**

Add the Dockerfile for API and Microservice in the path

[ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/[deploy\_rearch](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/deploy_rearch)/**dockerfiles**/API/<api-name>/Dockerfile

and

[ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/[deploy\_rearch](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/deploy_rearch)/**dockerfiles**/Microservice/<Microservice-name>/Dockerfile respectively.

Eg: [ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/[deploy\_rearch](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/deploy_rearch)/**dockerfiles**/API/consumeraddress-ms/Dockerfile

Please ensure the api-name and Microservice name are in lowercase.

* **Deployment.yml files:**
* Add the Deployment.yml files in the path

[ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/[deploy\_rearch](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/deploy_rearch)/[manifest.yml](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/deploy_rearch/manifest.yml)/**DEV**/<microservice-name or api-name>\_dev.yml

Eg: [ConsumerMaster--GSL-](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev)/[deploy\_rearch](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/deploy_rearch)/[manifest.yml](https://github.com/mattel-dig/ConsumerMaster--GSL-/tree/dev/deploy_rearch/manifest.yml)/**DEV**/[bulkupload\_dev.yml](https://github.com/mattel-dig/ConsumerMaster--GSL-/blob/dev/deploy_rearch/manifest.yml/DEV/bulkupload_dev.yml" \o "bulkupload_dev.yml)

Please ensure that the naming of the yml file should be a microservice or api name in lowercase followed by “\_dev.yml”.

* Please change the name of the image inside the yml file

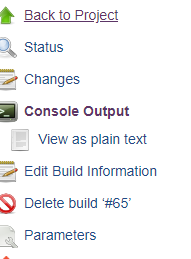
Eg: gcr.io/mattel-tools/cm-<Microservice or api-name>:latest

Please ensure that the naming of the image should be “cm-“ followed by the a microservice or api name in lowercase.

Always the image specified inside yml should be “latest”

**3.6 TROUBLESHOOTING IN CASE OF ANY ISSUE:**

* On failure of the job check its build logs
* The log of Jenkins job is visible under Jenkins🡪Pipeline\_name🡪Build History🡪Failed Build Number🡪ConsoleOutput



* If the Jenkins throws “No space in device” Error

Check the space in the mattel-tools (build server instance).Try cleaning up the images locally created in the instance.

Get the images using command: **docker images**

* If the jenkins throws “File specified not found “ Error

Check whether the naming constraints for all dependent files are been followed.(e.g. Dockerfile, Deployment.yml file)

* If jenkins throws Error under Deployment stage

Check the deployment.yml file Or check for the environment variables been correctly in GCP.