# **OLAM – Deployment Guide**

# **Technical Runbook Document**

**Version 1.0.0**

**Author – Mohankumar Deivasigamani**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S.No | Revision | Date | Reviewed By | Type | Review Comments |
| 1 | 1.0.0 | 27-Aug-2019 |  | Internal | Application deployment on AKS (Azure-Kubernetes) deployment. Initial version |

Revision History

Table of Contents

[**OLAM – Deployment Guide** 1](#_Toc17811873)

[**Technical Runbook Document** 1](#_Toc17811874)

[**1** **Introduction** 5](#_Toc17811875)

[1.1 Overview 5](#_Toc17811876)

[1.2 Scope 5](#_Toc17811877)

[1.3 Service deployment process 5](#_Toc17811878)

[2.1 High-Level Infrastructure diagram 6](#_Toc17811879)

[**3** **Azure AKS** 7](#_Toc17811880)

[3.1 Cluster Configuration 7](#_Toc17811881)

[3.2 Setup and Select Azure Subscription 7](#_Toc17811882)

[3.3 AKS Cluster Setup 9](#_Toc17811883)

[3.4 Granting privileges 9](#_Toc17811884)

[3.4.1 RBAC Authorization 10](#_Toc17811885)

[3.4.2 Permissive RBAC 10](#_Toc17811886)

[3.5 Configure Azure File Share 10](#_Toc17811887)

[3.5.1 Persistent Volume 11](#_Toc17811888)

[3.5.2 Persistent Volume Claim 12](#_Toc17811889)

[3.6 Configure Docker registry 12](#_Toc17811890)

[**4** **Consul Server in Kubernetes** 13](#_Toc17811891)

[4.1 Helm Installation 13](#_Toc17811892)

[4.1.1 Pre requires 13](#_Toc17811893)

[4.1.2 Configuration and Start Consul Server 13](#_Toc17811894)

[4.1.3 Installing Helm 14](#_Toc17811895)

[4.1.4 Create a Service Account (Tiller) 14](#_Toc17811896)

[4.1.5 Installing Consul 15](#_Toc17811897)

[4.1.6 Share Consul IP with Development Team 15](#_Toc17811898)

[**5** **Ingress Configuration** 15](#_Toc17811899)

[5.1.1 Overview 16](#_Toc17811900)

[5.1.2 Pre requires 16](#_Toc17811901)

[5.1.3 Install Ingress 16](#_Toc17811902)

[5.1.4 Share Ingress IP with Network Team 17](#_Toc17811903)

[5.1.5 Update Ingress Config 17](#_Toc17811904)

[5.1.6 Share host address with Development Team 18](#_Toc17811905)

[**6** **Continuous Integration & Deployment** 18](#_Toc17811906)

[6.1 Overview (Jenkins) 18](#_Toc17811907)

[6.2 Configuration CI/CD Pipeline 19](#_Toc17811908)

[6.2.1 Creating new Pipeline Job 20](#_Toc17811909)

[**7** **Annexure** 23](#_Toc17811910)

# **Introduction**

## Overview

**OLAM Microservice deployment:** In OLAM we have chosen the AKS (Azure Kubernetes cluster) platform to deploy all our microservice components irrespective of all the OLAM Digital projects.

The purpose of this document is to provide the birds-eye view on the deployment workflow setup. This document covers installation and deployment guide of MAS service.

## Scope

This document composed of installation guide of each section AKS/ Jenkins/ Helm/ and other services. It also covers step by step commands and instructions.

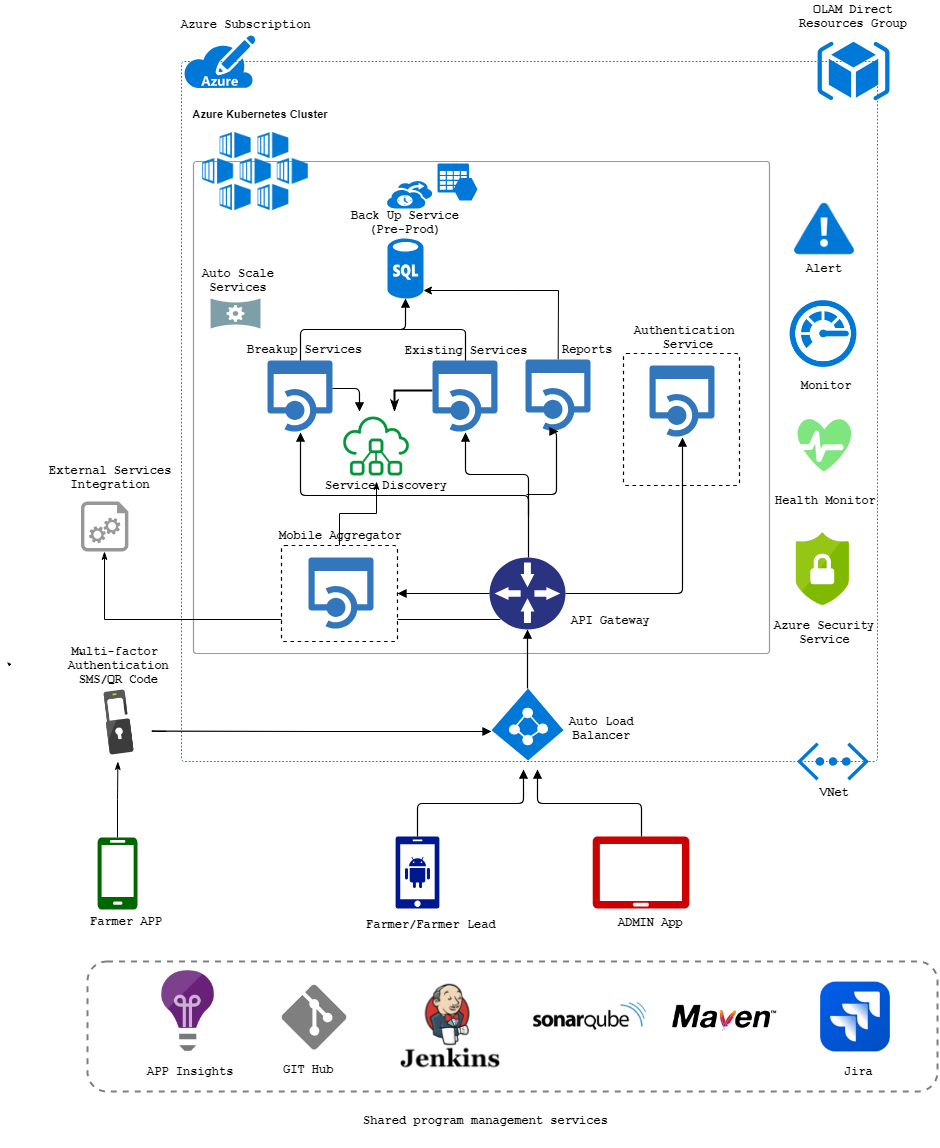
## Service deployment process

In this document, we are going to cover detailing of Dev Ops team activities.

|  |  |  |
| --- | --- | --- |
| 1. **Step** | **Activity** | **Team** |
| *1* | *Request for cloud infrastructure resource* | *Dev Team* |
| *2* | *Send infra resource request email to cloud infra team* | *Infra Manager* |
| *3* | *Procurement for resources and share details to infra manager* | *Infra Team* |
| *4* | *Sharing new cloud resource details to Dev Ops* | *Infra Manager* |
| *5* | *Sharing deployment service details (with all prerequisite) to Dev Ops* | *Dev Team* |
| *6* | *Base level configuration in AKS cluster and create deployment workflow* | *Dev Ops* |
| *7* | *Request Peer connection between cloud resources to infra team* | *Dev Ops* |
| *8* | *Share gateway service URI with Dev team* | *Dev Ops* |
| *9* | *Configure gateway URI in other services* | *Dev Team* |
| *10* | *Share gateway IP address to infra team for a domain, SSL and WAF config* | *Dev Ops* |
| *11* | *Mapping Public IP/ SSL & WAF configuration* | *Infra Team* |
| *12* | *Configure public IP with domain in Godaddy* | *Dev Ops* |
| *13* | *Once all request has been completed as requested by Dev Ops, the team deploys all other services.* | *Dev Ops* |
| *14* | *Share required URI to Dev team* | *Dev Ops* |
| *15* | *Jenkins URI and permission has shared with Dev team to deploy Dev / SIT* | *Dev Ops* |

*table 3: deployment process activity*

## High-Level Infrastructure diagram



# **Azure AKS**

Any stages of the application development lifecycle (DEV/ SIT/ UAT/ PROD), development team requires infrastructure to deploy. The infrastructure resource requirement has sent to infra manager. Dev Ops team will receive resources details from infra manager over email and service details from development team over email (refer attached service details template in annexure).

Cloud resource details

* Azure subscription name
* Cluster name
* Resource group name
* Storage account name
* Storage account key
* Relevant access from infra team

Service details

* Microservice name
* Git project URL
* Branch name
* Docker file name
* Configuration profile name
* Application port name
* Java version
* Other parameters

## Cluster Configuration

To configure the AKS cluster from your local machine, you required to install KUBECTI executable. refer: - <https://kubernetes.io/docs/tasks/tools/install-kubectl/>

On successful installation, open command line terminal and proceed with the below step-by-step command to configure.

## Setup and Select Azure Subscription

**Step 1:** Then run the following CLI command to connect with your Azure account, and you can see the below-highlighted message with login details to the web portal

C:\user\devopes>az login

To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter code HNSDXXXX8 to authenticate

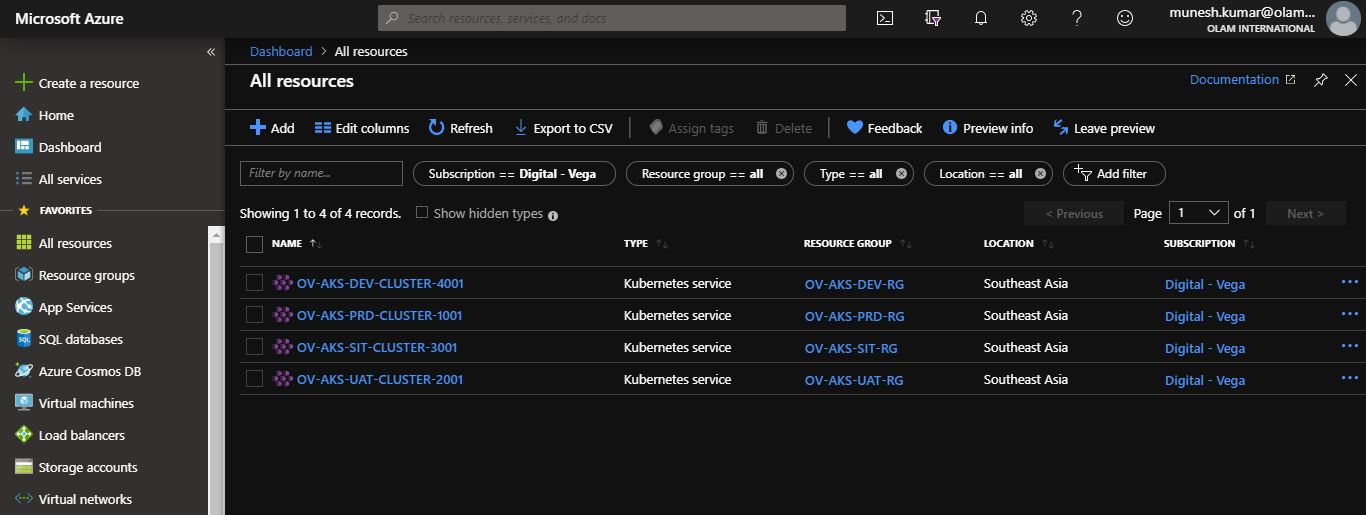
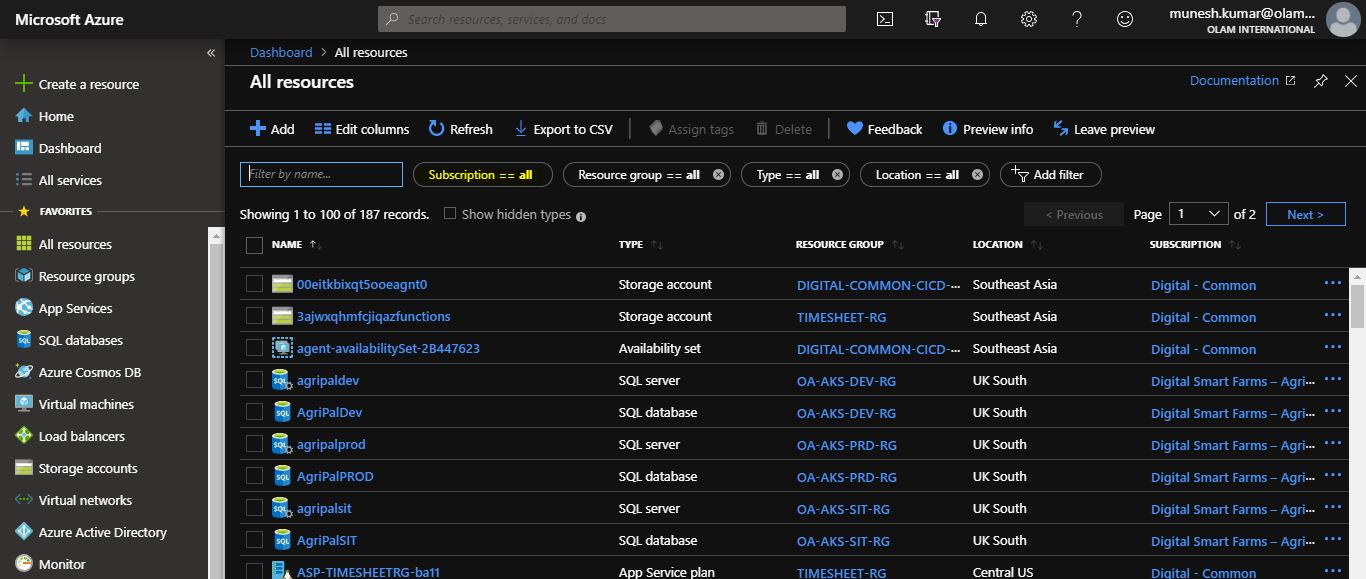
**Step 2:** Activate the correct subscription. Azure uses the concept of subscriptions to manage to spend. You can get a list of subscriptions your account has access to by running

C:\user\devopes>az account list

Pick the subscription, to set a subscription to be the current active subscription

C:\user\devopes>az account set --subscription "<<subscription name>>"

**Step 3:** Now you can log in into Azure web page and can see resources which are mapped to you for deployment



You can see view details cluster on selecting of cluster name in the cluster list

## AKS Cluster Setup

**Step 4:** Get access credentials for a managed Kubernetes cluster

C:\user\devopes>az aks get-credentials \

--resource-group OV-AKS-XXX-XX \

--name ov-aks-YYY-cluster-XXXX

* --name is the cluster name you received from infra manager over email
* --resource-group name you received from infra manager over email

This automatically updates your Kubernetes client configuration file.

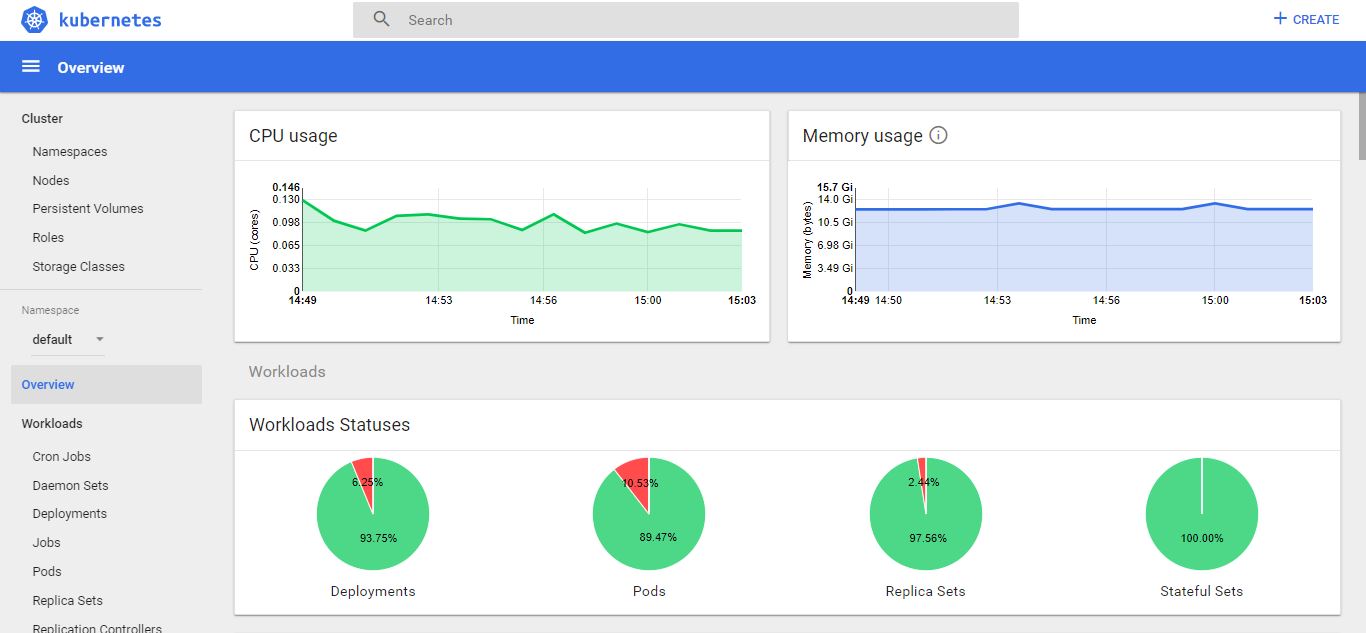
**Step 5:** To see the dashboard for a Kubernetes cluster on the web page

C:\user\devopes>az aks browse \

--resource-group OV-AKS-XXX-XX \

--name ov-aks-YYY-cluster-XXXX

* --name is the cluster name you received from infra manager over email
* --resource-group name you received from infra manager over email



## Granting privileges

Authorization Mode: Kubernetes API server may authorize a request using one of the authorization modes:

* Node
* ABAC (Attribute-based access control)
* RBAC (Role-base access control)
* Webhook (HTTP callback)

We are following RBAC access control for providing privilege

## RBAC Authorization

**Step 6:** Granting admin privileges to Dashboard’s Service Account create a ClusterRoleBinding which gives the role cluster-admin (= full admin privilege) to the ServiceAccount Kubernetes-dashboard

C:\user\devopes>kubectl create clusterrolebinding kubernetes-dashboard \

-n kube-system \

--clusterrole=cluster-admin \

--serviceaccount=kube-system:kubernetes-dashboard

## Permissive RBAC

**Step 7**: Permissive RBAC permissions. You can replicate a permissive policy using RBAC role bindings.

C:\user\devopes>kubectl create clusterrolebinding permissive-binding \

--clusterrole=cluster-admin \

--user=admin \

--user=kubelet \

--group=system:serviceaccounts

Any application running in a container receives service account credentials automatically and could perform any action against the API, including viewing secrets and modifying permissions.

## Configure Azure File Share

Share mounted Azure file storage across multiple containers in your deployment in hosted Kubernetes (AKS).

**Step 8:** Mount storage space (Azure File Storage) to AKS cluster, to map created storage

C:\user\devopes>kubectl create secret generic azure-storage-secret \

--from-literal=azurestorageaccountname=<<account-name>> \

--fromliteral=azurestorageaccountkey=<<account-key>>

## Persistent Volume

A hostPath PersistentVolume uses a file or directory on the Node to emulate network-attached storage. Instead, a cluster administrator would provide a network resource like Azure file storage.

Here is the configuration file for the hostPath PersistentVolume

|  |
| --- |
| apiVersion: v1  kind: PersistentVolume  metadata:  name: log-storage-new  labels:  usage: log-storage-new  spec:  capacity:  storage: 10Gi  accessModes:  - ReadWriteMany  persistentVolumeReclaimPolicy: Retain  azureFile:  secretNamespace: default  secretName: azure-storage-secret  shareName: spyderuatlogsss  readOnly: false |

* metadata: name: - The name of the volume. This is how it is identified via PV claims or from pods
* spec: capacity: storage: - The amount of storage allocated to this volume
* azureFile: - This defines the volume type being used: azureFile plug-in
* azureFile: secretName: - The name of the secret used.
* azureFile: shareName: - The name of the file share.
* azureFile: readOnly: - Defaults to false (read/write).

**Step 9:** Command to create the persistent volume

kubectl apply -f <<persistent yaml file>>

View information about the PersistentVolume

kubectl get pv <<metadata: name>>

## Persistent Volume Claim

The next step is to create a PersistentVolumeClaim. Pods use PersistentVolumeClaims to request physical storage, you create a PersistentVolumeClaim that requests a volume of at least three gibibytes that can provide read-write access for at least one Node.

Here is the configuration file for the PersistentVolumeClaim

|  |
| --- |
| apiVersion: v1  kind: PersistentVolumeClaim  metadata:  name: log-storage-claim-new  # Set this annotation to NOT let Kubernetes automatically create  # a persistent volume for this volume claim.  annotations:  volume.beta.kubernetes.io/storage-class: ""  spec:  accessModes:  - ReadWriteMany  resources:  requests:  storage: 10Gi  selector:  # To make sure we match the claim with the exact volume, match the label  matchLabels:  usage: log-storage-new |

**Step 10:** Command to create a persistent volume claim

kubectl apply -f <<persistent volume claim yaml file>>

View information about the PersistentVolume

kubectl get pvc <<metadata: name>>

## Configure Docker registry

A Kubernetes cluster uses the Secret of docker-registry type to authenticate with a container registry to pull a private image.

**Step 11:** Create a Secret by providing credentials on the command line

C:\user\devopes>kubectl create secret docker-registry digitalcontainerregistry \

--docker-server=<<your-registry-server>> \

--docker-username=<<username>> \

--docker-password=<<password>> \

--docker-email=<<email-address>> \

--namespace default

* --docker-server is your registry URI
* --docker-username is your registry username
* --docker-password is your registry password
* --docker-email email address of your registry

# **Consul Server in Kubernetes**

Consul can run directly on Kubernetes, both server and client mode. The recommended way to run Consul on Kubernetes is via Helm chart. This will install and configure all necessary components to run consul. The configuration enables you to run only server consul, using helm chart.

## Helm Installation

## Pre requires

* A Kubernetes 1.8+ cluster with role-based access control (RBAC) enabled.
* The kubectl command-line tool installed on your local machine configured to connect to your cluster.

If in case, multiple clusters configured, be sure to verify that you are selected correct cluster context

**Step 1:** To get the list of clusters configured in selected cluster

kubectl config get-contexts

**Step 2:**  You have connected to correct cluster context

kubectl config use-context <<context-name>>

## Configuration and Start Consul Server

**Step 3:** Activate the correct subscription. Azure uses the concept of subscriptions to manage to spend. You can get a list of subscriptions your account has access to by running

C:\user\devopes>az account list

Pick the subscription, to set a subscription to be the current active subscription

C:\user\devopes>az account set --subscription "<<subscription name>>"

**Step 4:** Get access credentials for a managed Kubernetes cluster

C:\user\devopes>az aks get-credentials \

--resource-group OV-AKS-XXX-XX \

--name ov-aks-YYY-cluster-XXXX

* --name is the cluster name you received from infra manager over email
* --resource-group name you received from infra manager over email

## Installing Helm

First, we need to install the helm command-line utility on our local machine. Helm provides a script that handles the installation process.

**Step 5:** Use the below command to download the script from Helm’s GitHub repository

curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > install-helm.sh

**Step 6:** On this step, you may prompt for your password

./install-helm.sh

## Create a Service Account (Tiller)

Tiller is a companion to the helm command that runs on your cluster, receiving commands from the helm and communicating directly with the Kubernetes API to do the actual work of creating and deleting resources. Before you can deploy Helm in an RBAC-enabled AKS cluster, you need a service account and role binding for the Tiller service.

**Step 7:** Create a file named helm-rbac.yaml and copy in the following YAML

|  |
| --- |
| apiVersion: v1  kind: ServiceAccount  metadata:  name: tiller  namespace: kube-system  ---  apiVersion: rbac.authorization.k8s.io/v1  kind: ClusterRoleBinding  metadata:  name: tiller  roleRef:  apiGroup: rbac.authorization.k8s.io  kind: ClusterRole  name: cluster-admin  subjects:  - kind: ServiceAccount  name: tiller  namespace: kube-system |

**Step 8:** Create the tiller service account

kubectl apply -f helm-rbac.yaml

**Step 9:** Now we can run helm init, which installs Tiller on our cluster

helm init --service-account tiller --node-selectors "beta.kubernetes.io/os"="linux"

**Step 10:** Upgrade the helm or tiller if required,

helm init --upgrade

## Installing Consul

To install Consul, clone the consul-helm repository,

**Step 11:** Check out the latest release

git clone <https://github.com/hashicorp/consul-helm.git>

**Step 12:** Install consul service

cd consul-helm

helm install --f .\Consul

**Step 13:** Verify the consul installation, and to get service URI

kubectl get services

## Share Consul IP with Development Team

**Step 14:**

Note: - Get this service IP and share it with the application development team.

# **Ingress Configuration**

An Ingress is an API object that allows and manage access to Kubernetes cluster from outside network.

## Overview

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined in the Ingress resource.

## Pre requires

* A Kubernetes 1.8+ cluster with role-based access control (RBAC) enabled.
* The kubectl command-line tool installed on your local machine configured to connect to your cluster.
* Helm package manager for Kubernetes

Note: - If in case, multiple clusters configured, be sure to verify that you are selected correct cluster context and pick and set a subscription to be the current active subscription

## Install Ingress

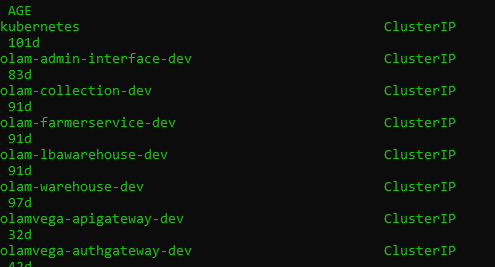
Refer below command to install ingress in selected Kubernetes cluster

**Step 1:** This command will install both the ingress controller and default backend service in the cluster

helm install stable/nginx-ingress

**Step 2:** To verify the service creation, run following command

kubectl get services



Now you can see a couple of service running and listed, initially ingress controller will expose public IP. Since we required private IP for internal load balancer which needs to be shared with the network team for public IP address mapping.

**Step 3:** Step needs to be followed to enable load balancer in the ingress

kubectl edit service <<ingress controller service name>>

Edit service configuration yaml file and add below parameters & save

|  |
| --- |
| metadata:  annotations:  service.beta.kubernetes.io/azure-load-balancer-internal: "true" |

After the configuration file update, the private IP address will be mapped with the ingress controller.

## Share Ingress IP with Network Team

**Step 4:**

Note: - Get this ingress IP and share it with the network team for public IP mapping

## Update Ingress Config

To modify ingress property, we required config file which needs to create and store in local machine <<ingress.yaml>> to apply. Generally, ingress needs three main fields in property file apiversion, kind, and metadata.

The Ingress property has all the information needed to configure a load balancer or proxy server. Most importantly, it contains a list of rules matched against all incoming requests. Ingress resource only supports rules for directing HTTP traffic.

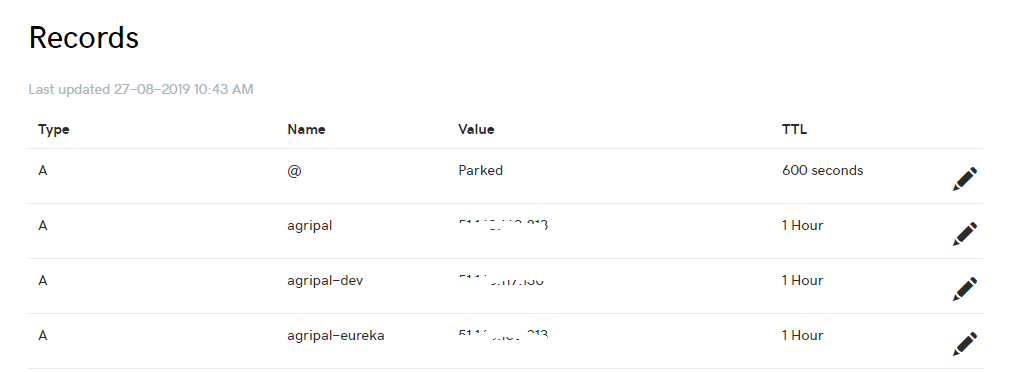
Here the ingress property file

|  |
| --- |
| apiVersion: extensions/v1beta1  kind: Ingress  metadata:  name: ssl  annotations:  kubernetes.io/ingress.class: "nginx"  nginx.ingress.kubernetes.io/rewrite-target: /  spec:  rules:  - host: bakewell-gateway-dev.olamdigital.com <<host address>>  http:  paths:  - backend:  serviceName: bakewell-scgw-dev <<gateway service or endpoint service>>  servicePort: 80 |

**Step 5:** To apply, above property with ingress

kubectl apply -f <<ingress.yaml>>

**Step 6:** Add A record in Godaddy for the hostname configured with ingress and public IP address received from the network team.



## Share host address with Development Team

**Step 4:**

Note: - Host address will be shared with development team, in need basic dev-ops team will share more details.

# **Continuous Integration & Deployment**

## Overview (Jenkins)

Jenkins helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery. It supports version control tools, includes GIT and can execute Apache Maven-based projects. Jenkins will support more version control tools and source execute tools, but as of now, we configured the mentioned tools for deployment.

We have in our Olam, both Jenkins and GIT is on-premises deployment inside our network.

The reference URI of,

Jenkins: <https://cyborg.olamdigital.com/>

GIT: <https://digitalgit.olamnet.com/>

## Configuration CI/CD Pipeline

For continues deployment, the team followed below steps to create a new pipeline job in Jenkins tool.

Assumption (pre-required configurations):

* Jenkins tool already configured and running in current cluster pod
  + Maven configured in the same pod
* GIT auth is configured in current cluster pod
* SonarQube is configured in current cluster pod

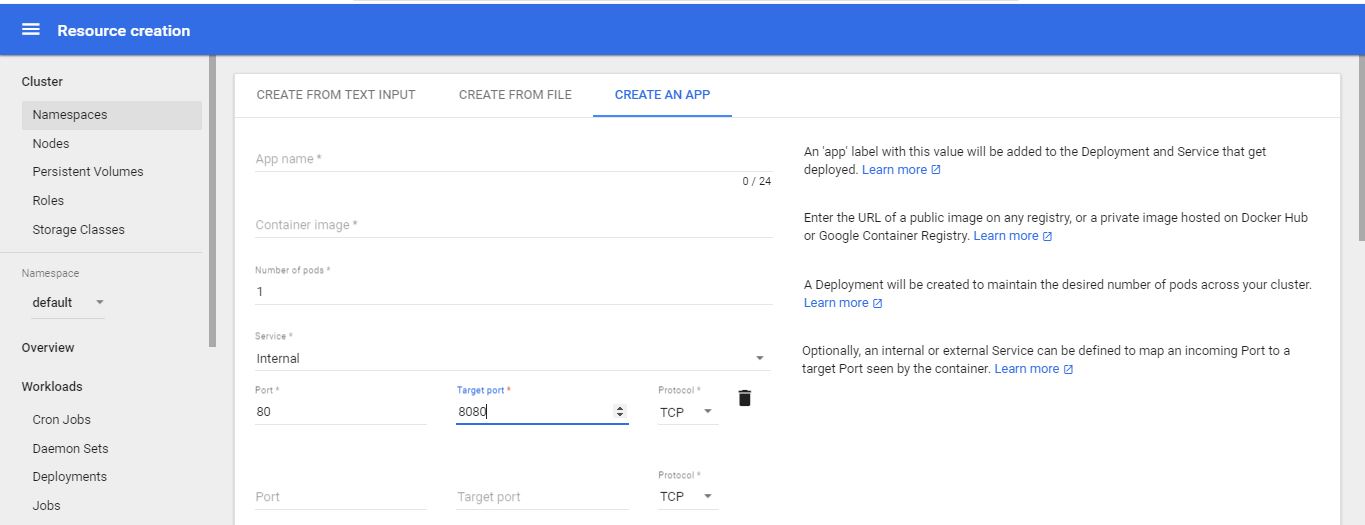
The team is creating two different new items (Pipeline jobs) with different stages of pipeline script for the same service deployment.

Different stages in CI/CD pipeline script:

1. Check out
   1. Service source code
   2. Service configuration
2. Execute/Trigger SonarQube code analysis
3. Build stage
4. Create a docker image and check-in to Azure container repo
5. Deployment stage

For the initial(first) deployment, the team will cover steps from 1 to 4 in pipeline script the fifth step is carried out by Dev Ops team manually.

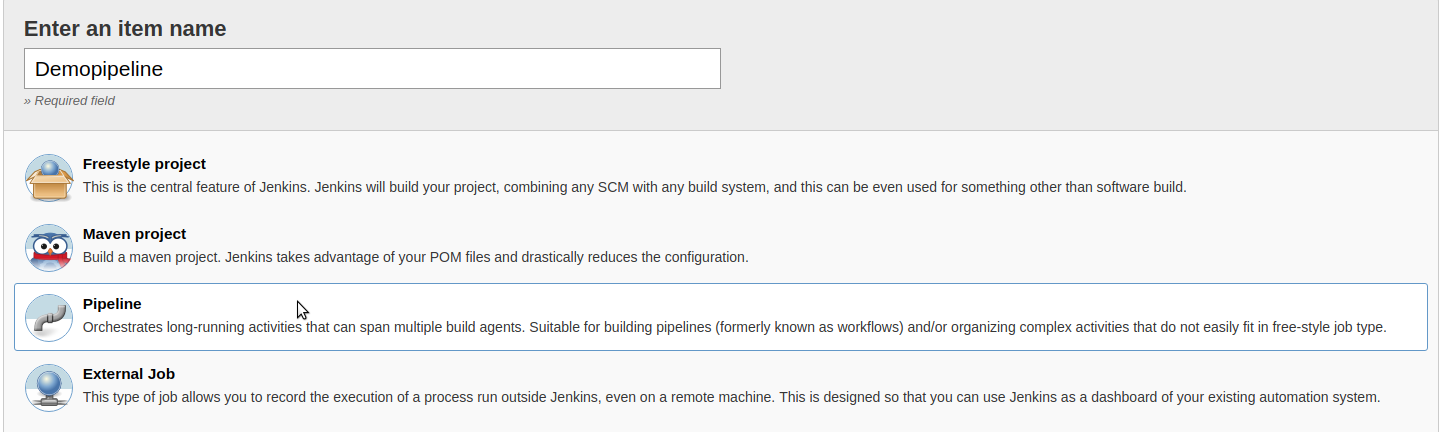
New service resource will be created in AKS cluster in the Azure portal



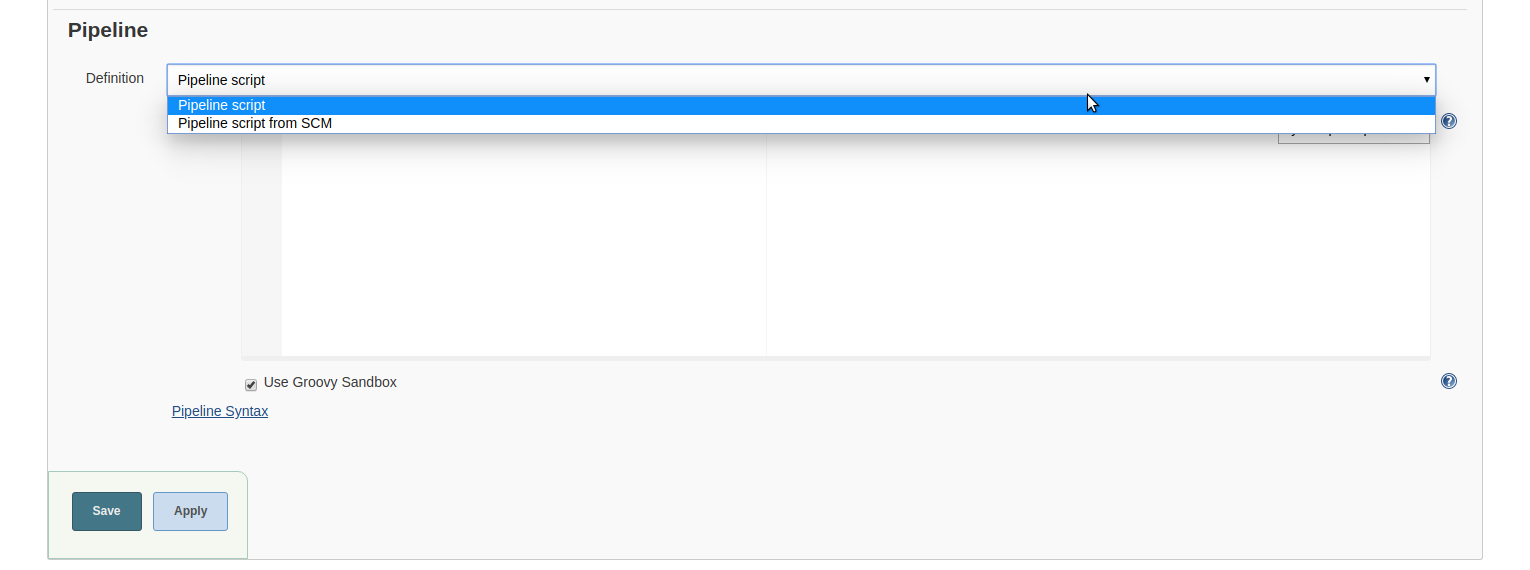
Another pipeline stage will be used for service update/upgrade which will help for auto-deployment when deployment job triggered.

## Creating new Pipeline Job

**Step 1:** Create a new deployment project (new item) in Jenkins



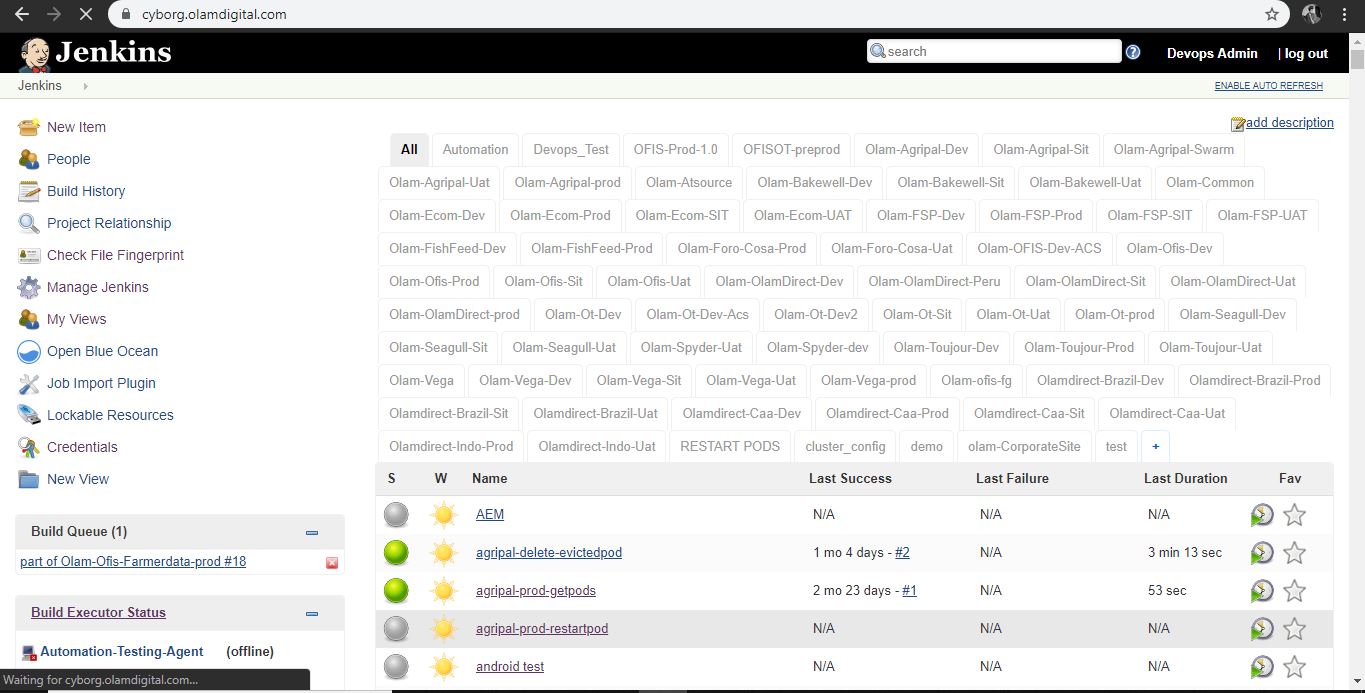
**Step 2:** After created pipeline, the user will able to see this page in the job detail page.



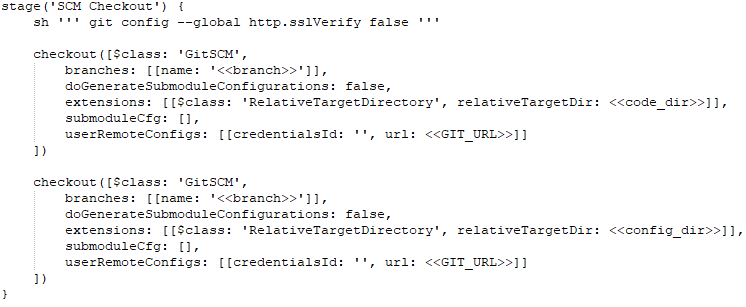
There are two options here:

* Pipeline scripts
  + We can write a pipeline script on Jenkins
* Pipeline scripts from SCM
  + We can write Jenkinsfile and upload into GIT(SCM) and connect it using pipeline script from SCM option

In Olam, we are using pipeline script, below brief explanation about stages in pipeline script (*The full pipeline script file attached in the annexure part*)

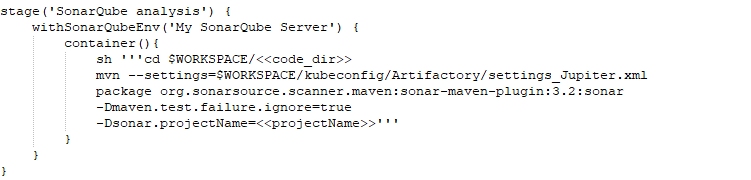


**Stage 1:** Check out from SCM, both application service source code and configuration into build machine



|  |  |
| --- | --- |
| **Parameters** | **Description** |
| branch | *SCM branch name* |
| code\_dir | *Application service code download path* |
| condig\_dir | *Application configuration download path* |
| git\_url | *SCM repository URI* |

**Stage 2:** To execute SonarQube code analysis

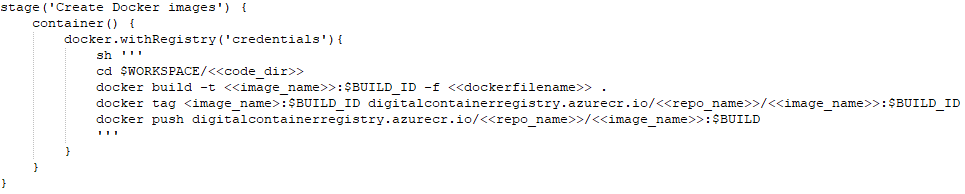


|  |  |
| --- | --- |
| **Parameters** | **Description** |
| projectName | *Name of the application* |
| code\_dir | *Application service code downloaded path* |

**Stage 3:** To build service code for deployment

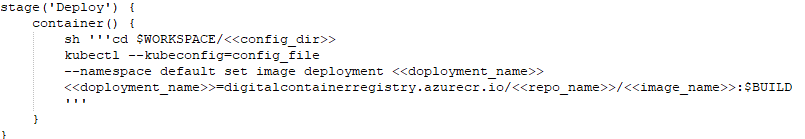


|  |  |
| --- | --- |
| **Parameters** | **Description** |
| code\_dir | *Application service code downloaded path* |

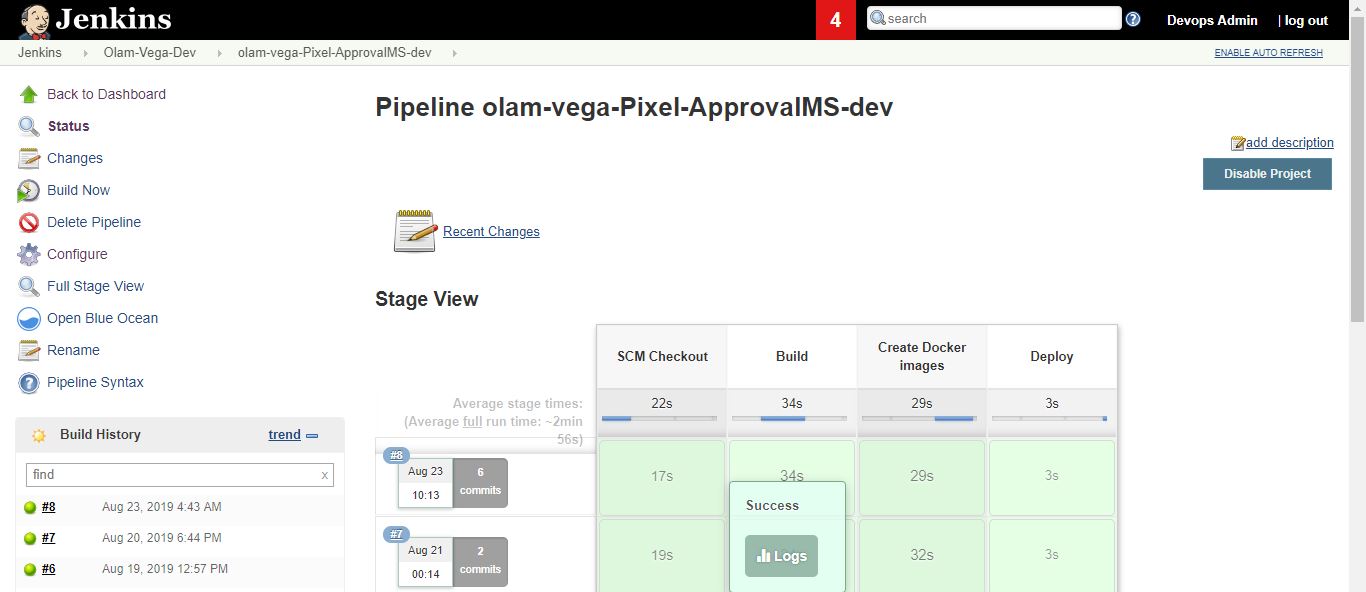
**Stage 4:** Create a new docker image and push to Azure container registry for continues deployment 

|  |  |
| --- | --- |
| **Parameters** | **Description** |
| dockerfilename | *Name of the docker file* |
| code\_dir | *Application service code downloaded path* |
| image\_name | *Docker image name* |
| repo\_name | *Container repository name* |

**Stage 5:** Deploy compiled library in cluster node



|  |  |
| --- | --- |
| **Parameters** | **Description** |
| deployment\_name | *Azure resource name, you will get once you deploy service manually* |
| config\_dir | *Application configuration downloaded path* |
| repo\_name | *Container repository name* |
| image\_name | *Docker image name* |



Above screen shows the pipeline stage view of deployment flow.

On completion of all service deployment, dev-ops team create new login account (is not already created) and provide access to view and execute Jenkins job to deploy services in development and testing (SIT) environment.

# **Annexure**

Pipeline script: -



node {

stage('SCM Checkout') {

sh ''' git config --global http.sslVerify false '''

checkout([$class: 'GitSCM', branches: [[name: '\*/master']], doGenerateSubmoduleConfigurations: false, extensions: [[$class: 'RelativeTargetDirectory', relativeTargetDir: 'ofis-survey-prod']], submoduleCfg: [], userRemoteConfigs: [[credentialsId: '89f78578-e807-4ed7-b999-50ab3544674f', url: 'https://172.29.0.110/OFIS/Olam-ofis-survey-service.git']]])

checkout([$class: 'GitSCM', branches: [[name: '\*/master']], doGenerateSubmoduleConfigurations: false, extensions: [[$class: 'RelativeTargetDirectory', relativeTargetDir: 'kubeconfig']], submoduleCfg: [], userRemoteConfigs: [[credentialsId: '89f78578-e807-4ed7-b999-50ab3544674f', url: 'https://172.29.0.110/DevOps/olam\_digitaldevops.git']]])

}

stage('SonarQube analysis') {

withSonarQubeEnv('My SonarQube Server') {

container('maven11'){

sh '''cd $WORKSPACE/ofis-survey-prod

mvn --settings=$WORKSPACE/kubeconfig/Artifactory/settings\_1003\_password.xml package org.sonarsource.scanner.maven:sonar-maven-plugin:3.2:sonar -Dmaven.test.failure.ignore=true -Dsonar.projectName=ofis-survey-prod -Dsonar.projectKey=ofis-survey-prod'''

}

}

}

stage('Build') {

container('maven11'){

sh '''cd $WORKSPACE/ofis-survey-prod

mvn --settings=$WORKSPACE/kubeconfig/Artifactory/settings\_1003\_password.xml clean install -DskipTests=true'''

}

}

stage('Create Docker images') {

container('docker') {

docker.withRegistry('https://digitalcontainerregistry.azurecr.io', '729303a6-6bcf-45bd-b7fb-1ea2e5c04d4d'){

sh '''

cd $WORKSPACE/ofis-survey-prod

docker build --build-arg profile=ofisprod -t ofis-survey-prod:$BUILD\_ID -f Dockerfile-ofis .

docker tag ofis-survey-prod:$BUILD\_ID digitalcontainerregistry.azurecr.io/olam-ot-prod/ofis-surveys-prod:$BUILD\_ID

docker push digitalcontainerregistry.azurecr.io/olam-ot-prod/ofis-surveys-prod:$BUILD\_ID

'''

}

}

}

stage('Deploy') {

container('kubectl') {

sh '''cd $WORKSPACE/kubeconfig/kubeconfig

kubectl --kubeconfig=config\_OT\_Prod --namespace default set image deployment ofis-survey-prod ofis-survey-prod=digitalcontainerregistry.azurecr.io/olam-ot-prod/ofis-surveys-prod:$BUILD\_ID

'''

}

}

}