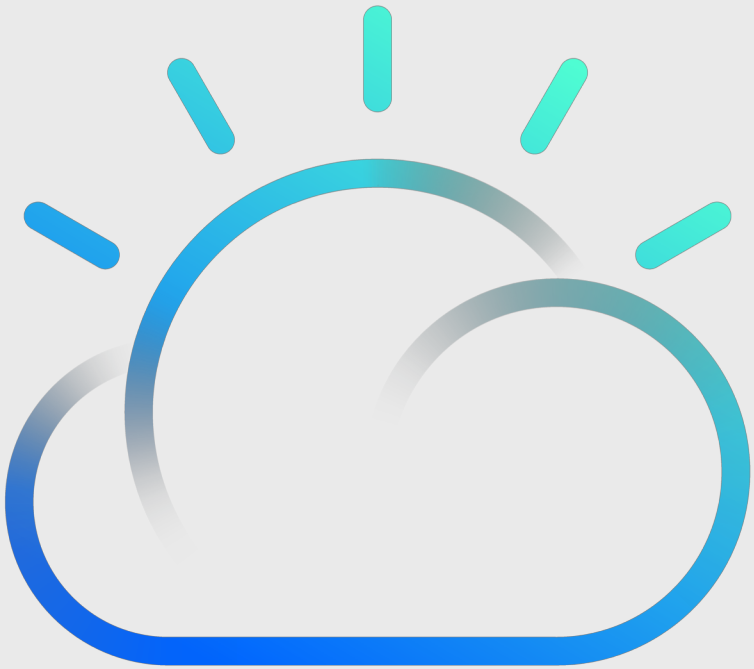
Continuous integration / continuous deployment (CI/CD)



IBM Cloud

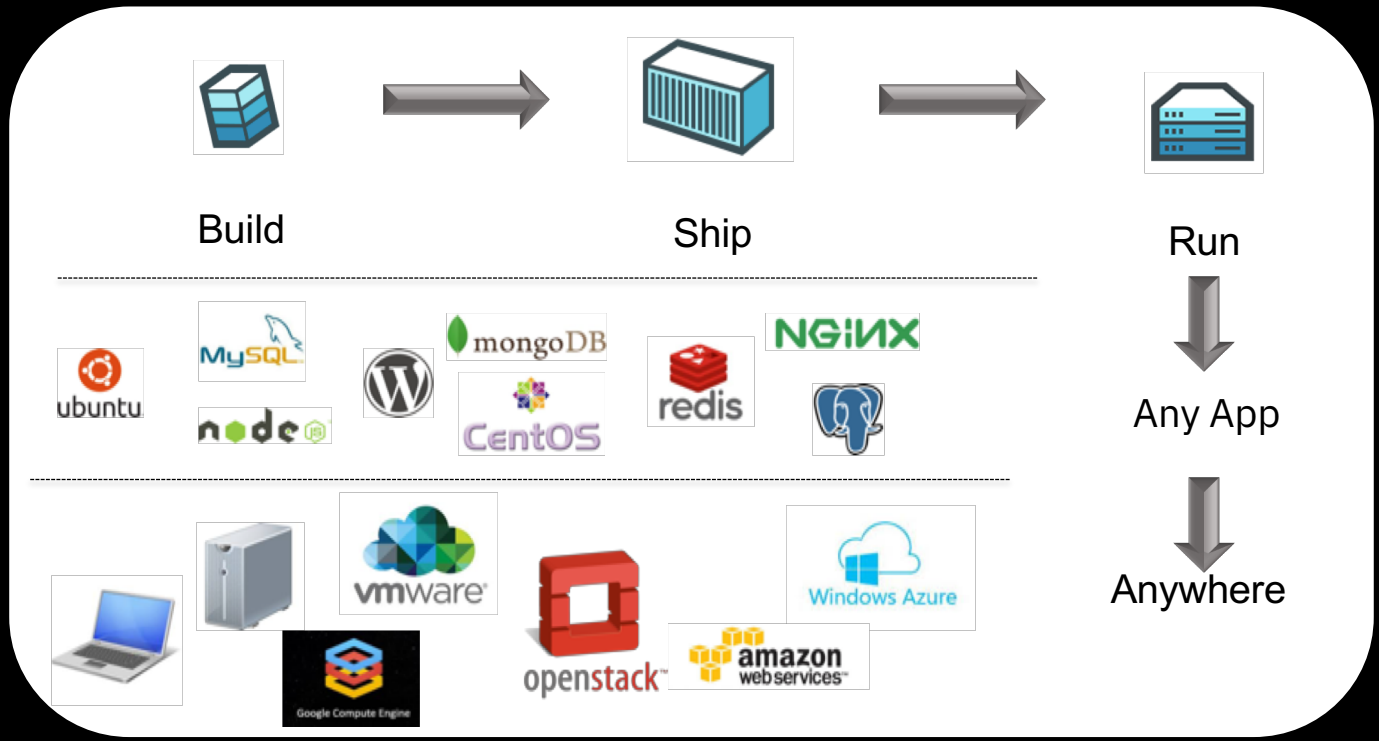
Objectives

After completing this lecture, you will be able to:

- Describe Continuous Integration and Continuous Deployment (CI/CD)
- Describe the Continuous Integration and Continuous Deployment capabilities in IBM Cloud Private (ICP)
- Describe the build automation technologies in IBM Cloud Private
- Describe the best practices for automating enterprise ICP deployments

Docker Workflow

Docker is an **open platform** for building distributed applications for developers and system administrators.



Definitions

CI is Continuous Integration (Build/Ship)

- Builds, unit tests, integration tests, performance tests, …

CD is Continuous Deployment (Run)

- Deploying and maintaining pre production and production environment

CI is Dev, CD is Ops …..CI and CD is DevOps (Build/Ship/Run)

Jenkins

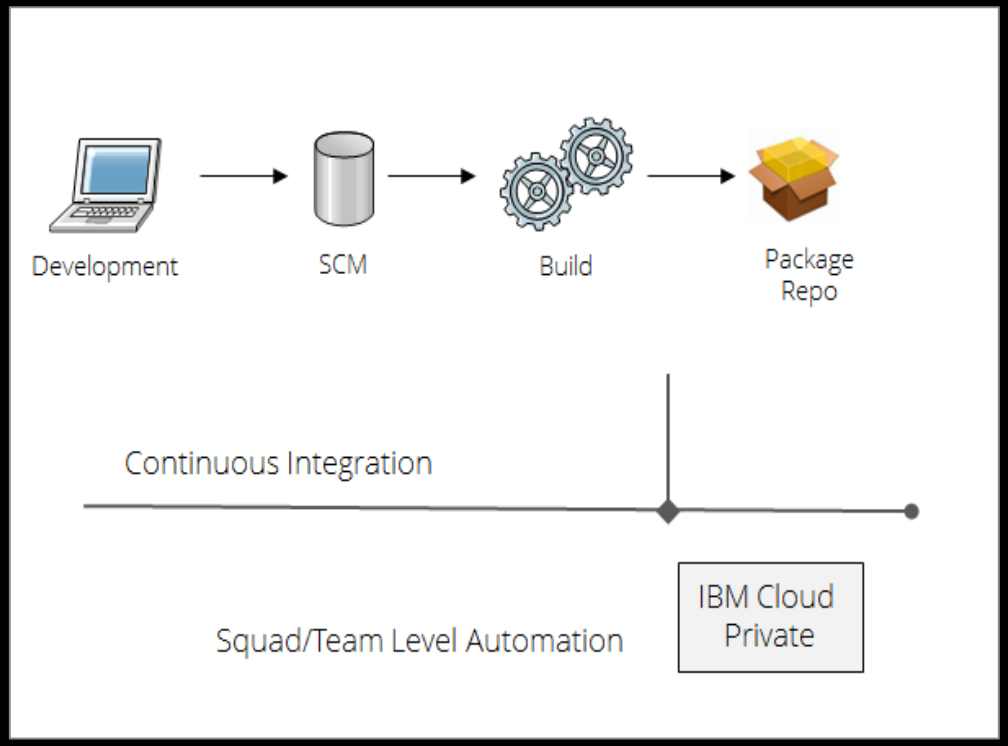
Popular open-source framework for Continuous Integration

Monitors execution of repeated jobs; examples: cron jobs or builds

Generally used for:

- Building/testing software projects continuously
- Monitoring executions of externally-run jobs, such as cron jobs

ICP Continuous Integration (**Build/Ship**/Run)



Frequently performing all of these steps in sequence:

Development

- Rapidly implementing changes in small, tested batches

Source Code Management

- Merging changes from multiple developers

Build

- Creating new deployment artifacts

Package

- Installing builds into runtimes
- Releasing runtimes as immutable images

Automating build

Jenkinsfile: Script file that is used to describe a Jenkins Pipeline.

podTemplate: Describes the docker containers used by this Jenkinsfile

Extract stage: Extracts the source code from *git* and gets the last *commit id*

docker stage: Builds the *flashboard* docker image, tags it with the *commit id* and pushes it to IBM Cloud Private

```
def volumes = [ hostPathVolume(hostPath: '/var/run/docker.sock', mountPath: '/var/run/docker.sock') ]
volumes += secretVolume(secretName: 'microclimate-icp-secret', mountPath: '/msb_reg_sec')
podTemplate(label: 'icp-build',
    containers: [
        containerTemplate(name: 'maven', image: 'maven:3.5.2-jdk-8', ttyEnabled: true, command: 'cat'),
        containerTemplate(name: 'docker', image: 'ibmcom/docker:17.10', ttyEnabled: true, command: 'cat'),
    ],
    volumes: volumes
)
{
    node ('icp-build') {
        def gitCommit
        stage ('Extract') {
          checkout scm
          gitCommit = sh(script: 'git rev-parse --short HEAD', returnStdout: true).trim()
          echo "checked out git commit ${gitCommit}"
        }
        stage ('maven build') {
          container('maven') {
            sh '''
            mvn clean test install
            '''
          }
        }
        stage ('docker') {
          container('docker') {
            def imageTag = "mycluster.icp:8500/nm-pilot/flashboard:${gitCommit}"
            echo "imageTag ${imageTag}"
            sh """
            docker build -t flashboard .
            docker tag flashboard $imageTag
            ln -s /msb_reg_sec/.dockercfg /home/jenkins/.dockercfg
            mkdir /home/jenkins/.docker
            ln -s /msb_reg_sec/.dockerconfigjson /home/jenkins/.docker/config.json
            docker push $imageTag
            """
          }
        }
    }
}
```

maven stage: Executes a *maven* clean, test and install build process

Dockerfile for Liberty

FROM: Base image for the new image. IBM has published an official set of foundational Docker images containing IBM products, such as WebSphere Liberty

Dockerfile: A text file containing Docker image building instructions

COPY: Instruction to copy a configured Liberty server with your application deployed as a whole

Dockerfile

```
1 FROM websphere-liberty:webProfile7
2 COPY /target/liberty/wlp/usr/servers/defaultServer /config/
3 COPY /target/liberty/wlp/usr/shared/resources /config/resources/
4 COPY /src/main/liberty/config/jvm.options /config/jvm.options
5 COPY /lib/vertica-jdbc-7.2.3-0.jar /config/resources/vertica-jdbc-7.2.3-0.jar
6 RUN installUtility install --acceptLicense defaultServer
```

COPY: Instructions to copy shared resources and jvm.options files to the correct folders in the image

RUN: Execute shell commands to install all required Liberty features

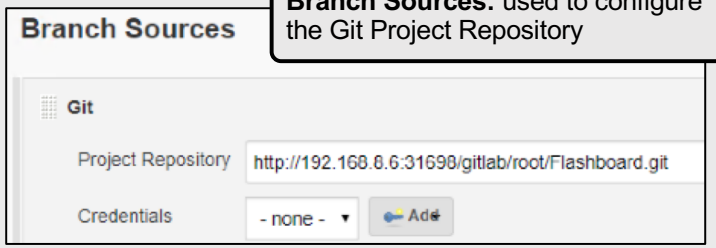
COPY: Instead of copying a configured Liberty server, you may also copy a configured *server.xml* file and application deployable to the image

Alternate

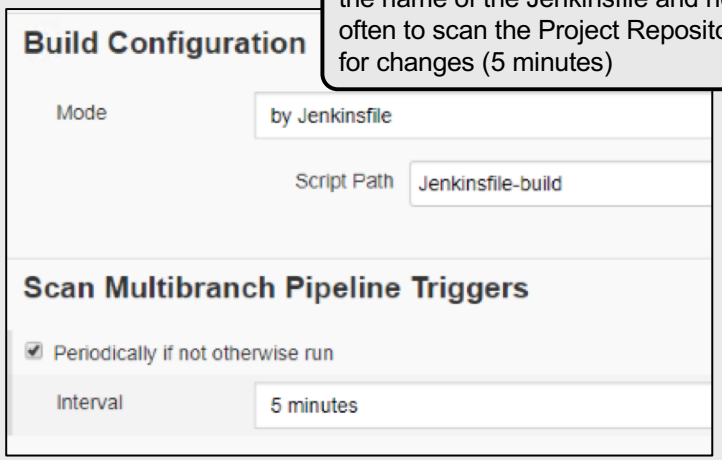
```
2 COPY server.xml /config
3 COPY FB_WAS.war /config/dropins
```

Jenkins Pipeline for automated build

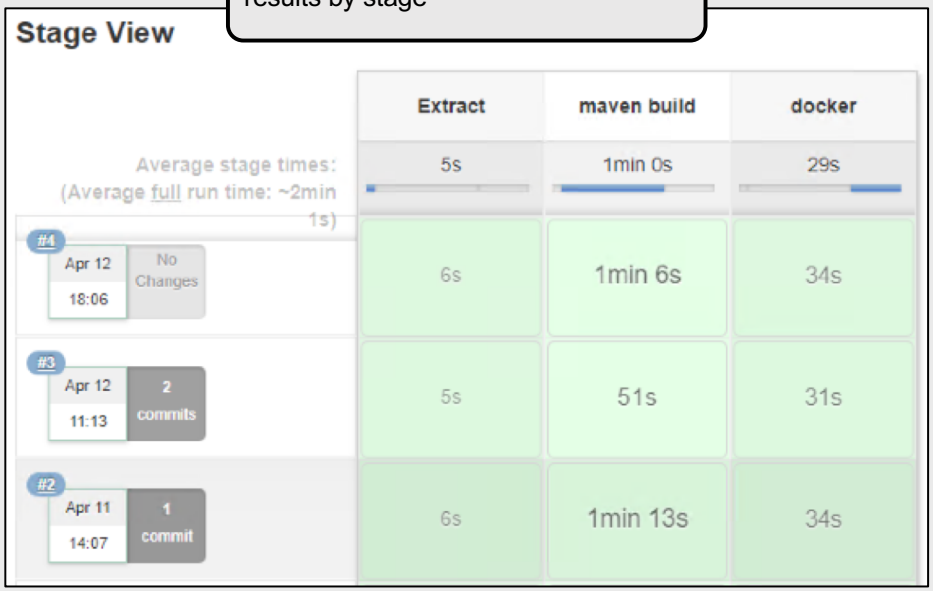
Branch Sources: used to configure the Git Project Repository



Build Configuration: used to specify the name of the Jenkinsfile and how often to scan the Project Repository for changes (5 minutes)



Build Results: displays the most recent pipeline executions and the results by stage



ICP Continuous Deployment (Build/Ship/**Run**)

Rapidly progressing the latest packaged build through the test lifecycle stages and into production

Deploy to Test

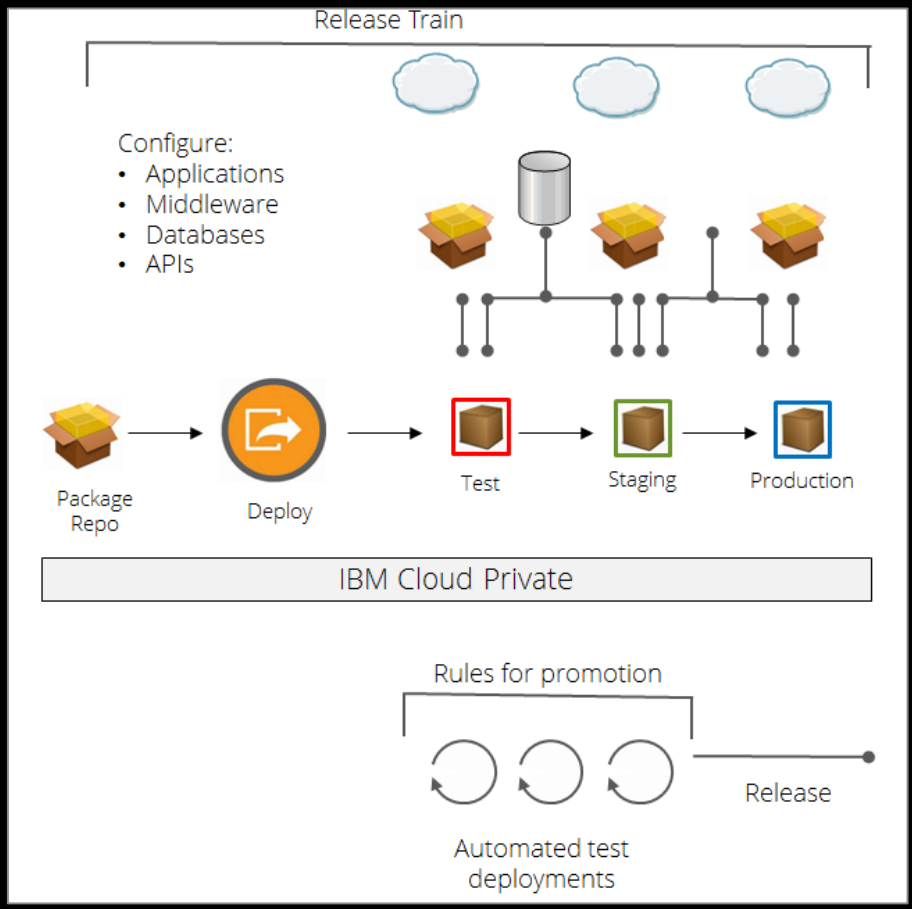
- Perform functional testing

Deploy to Stage

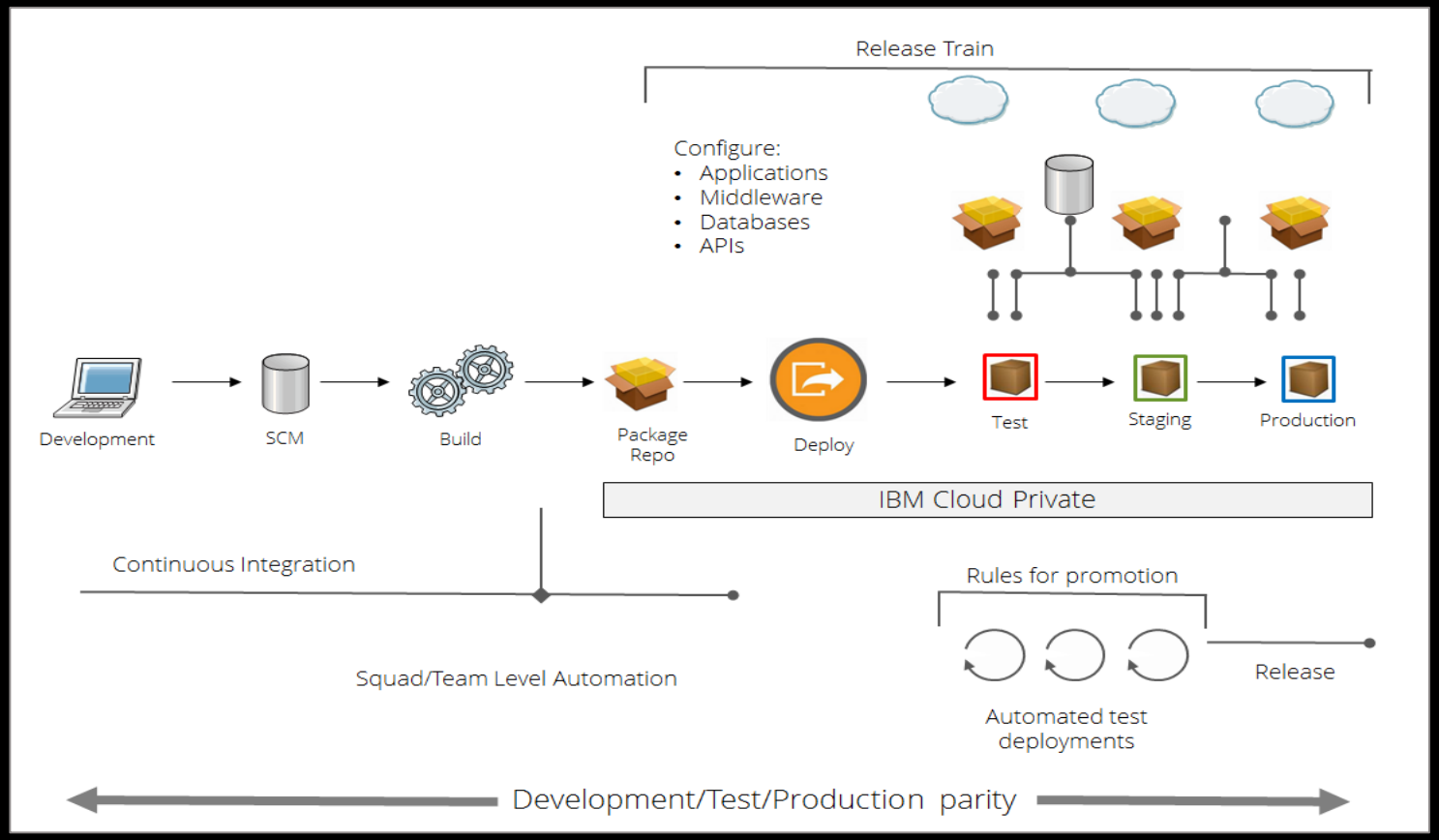
- Rehearse production deployment
- Perform integration testing

Deploy to Prod

- Make the build available to users



ICP CI/CD (Continuous Delivery) pipeline



Automating deployment using Jenkins

podTemplate: Deployment requires a docker container with the kubernetes CLI installed

Extract stage: Extracts the source code from *git* and gets the last *commit id*

deploy stage: uses the Kubernetes CLI to check whether the *deployment* already exists.

If the *deployment* exists then it is updated with a new docker image which triggers the Pods to be destroyed and recreated with the new image.

If the *deployment* doesn't exist then the *deployment script file* is updated with the docker image name and tag and then the *deployment* and *service* are created

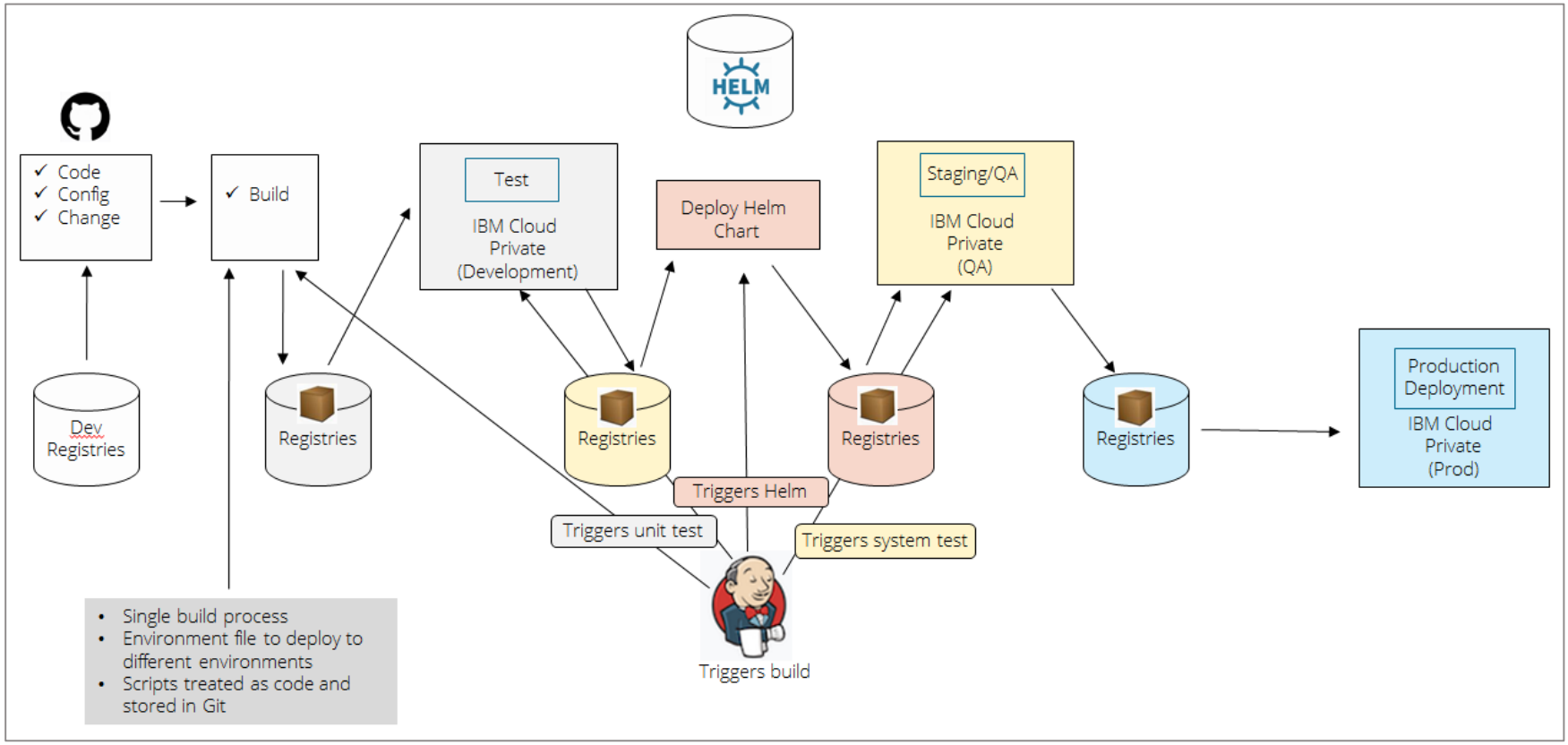
```
def volumes = [ hostPathVolume(hostPath: '/var/run/docker.sock', mountPath: '/var/run/docker.sock') ]
volumes += secretVolume(secretName: 'microclimate-icp-secret', mountPath: '/msb_reg_sec')
podTemplate(label: 'icp-build',
    containers: [
        containerTemplate(name: 'kubectl', image: 'ibmcom/k8s-kubectl:v1.8.3', ttyEnabled: true, command:
        'cat')
    ],
    volumes: volumes
)
{
    node ('icp-build') {
        stage ('Extract') {
            checkout scm
            gitCommit = sh(script: 'git rev-parse --short HEAD', returnStdout: true).trim()
            echo "checked out git commit ${gitCommit}"
        }
        stage ('deploy') {
          container('kubectl') {
            def imageTag = null
            imageTag = gitCommit
            sh """
            #!/bin/bash
            echo "checking if flashboard-deployment already exists"
            if kubectl describe deployment flashboard-deployment --namespace nm-pilot; then
                echo "Application already exists, update..."
                kubectl set image deployment/flashboard-deployment flashboard=mycluster.icp:8500/nm-
                pilot/flashboard:${imageTag} --namespace nm-pilot
            else
                sed -i "s/<DOCKER_IMAGE>/flashboard:${imageTag}/g" manifests/kube.deploy.yml
                echo "Create deployment"
                kubectl apply -f manifests/kube.deploy.yml --namespace nm-pilot
            fi
            echo "Describe deployment"
            kubectl describe deployment flashboard-deployment --namespace nm-pilot
            echo "finished"
            """
          }
        }
    }
}
```

Automation

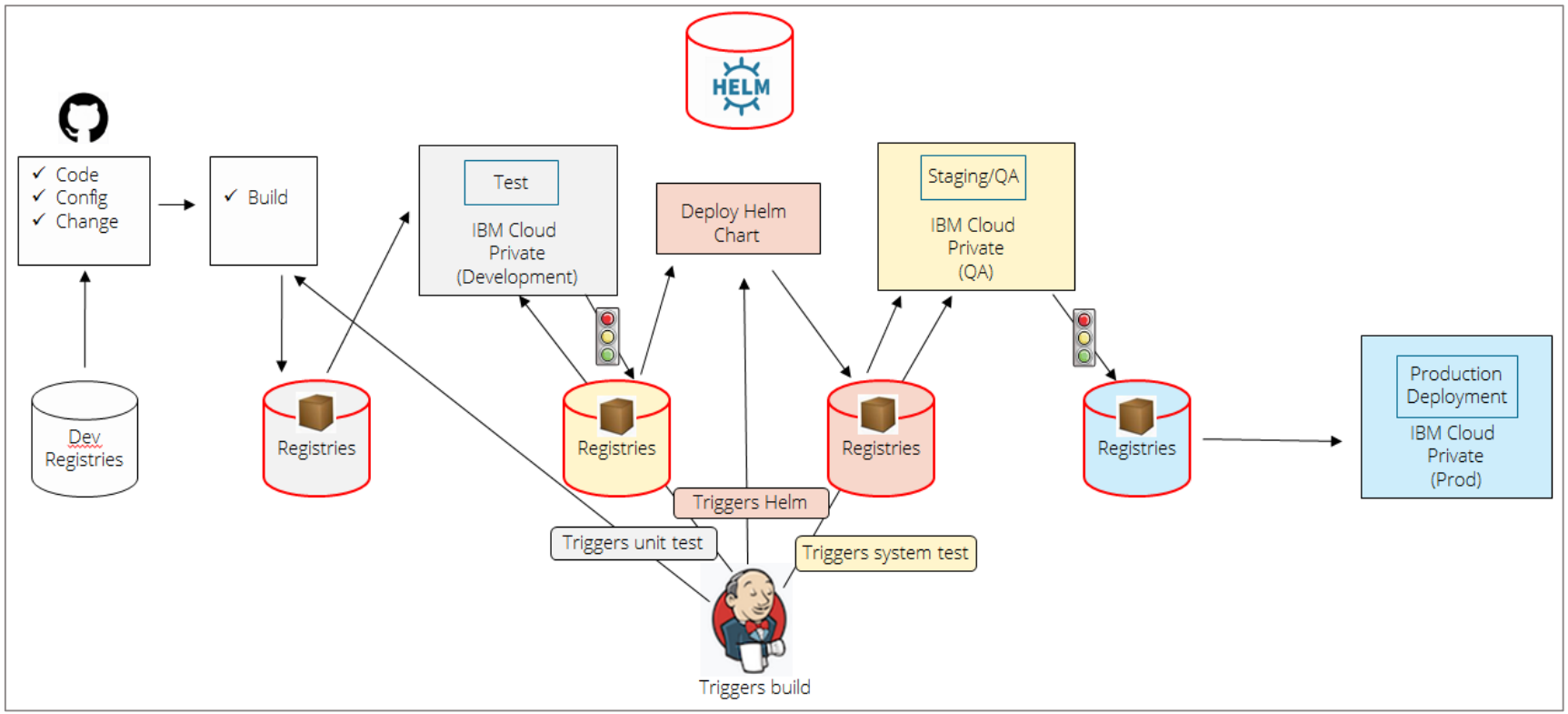
Helm

- A package manager for Kubernetes
- Enables multiple Kubernetes resources to be created with a single command
- Deploying an application often involves creating and configuring multiple resources

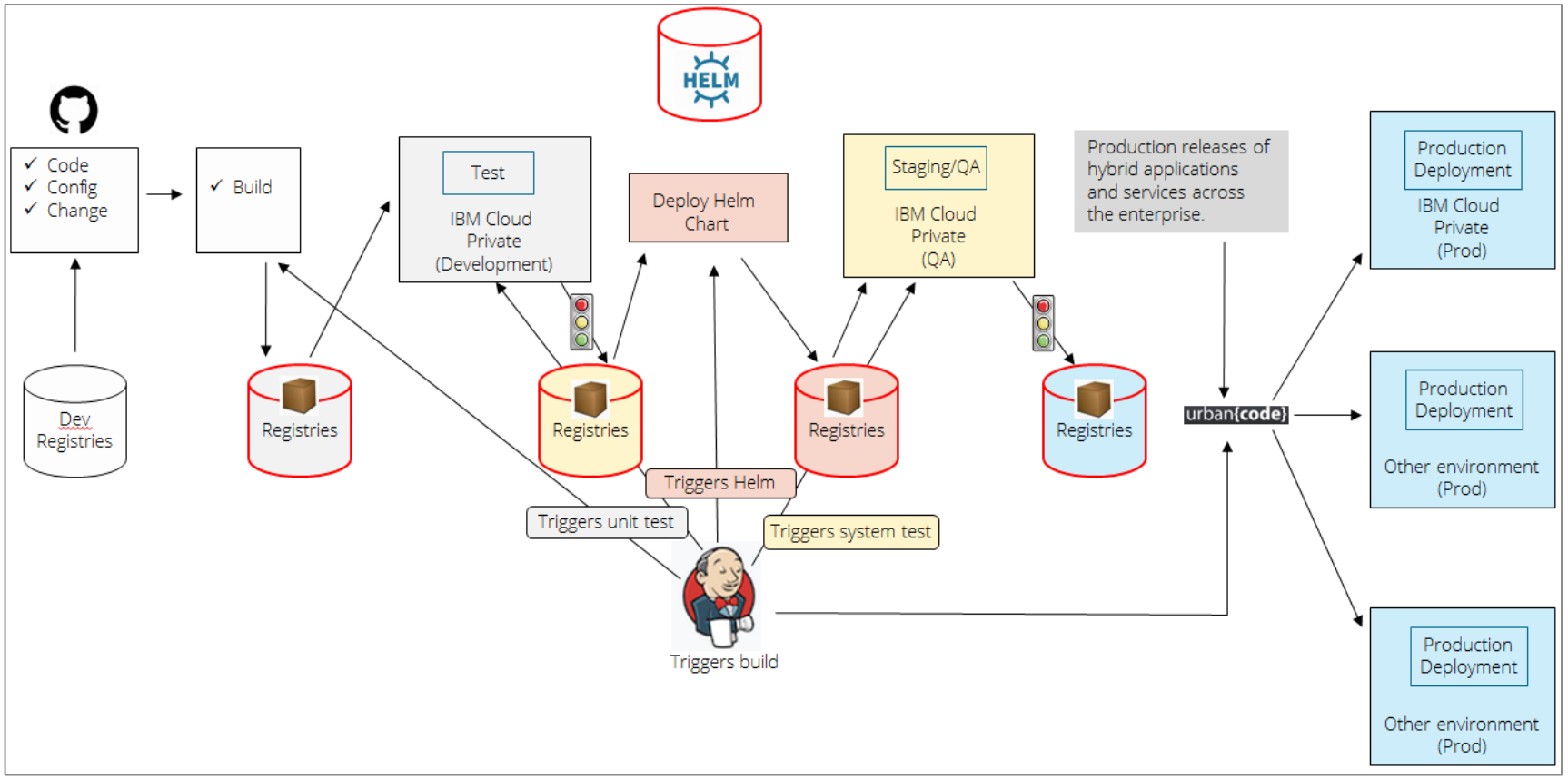
Helm deployment



Quality gates and trusted, secure image repository



Deployment to Hybrid

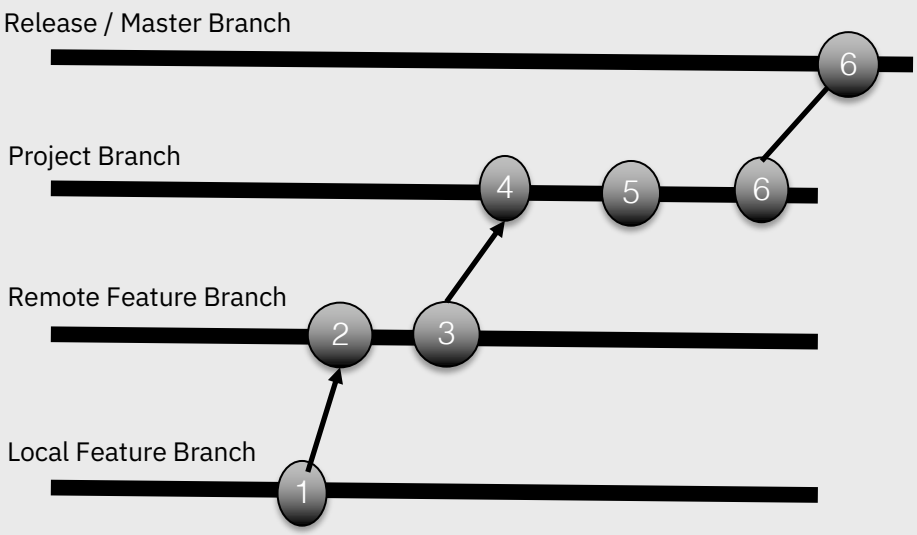


Git Branches Mapped to Environments

In this scenario we will use a single Git repository to support a 3 - 8 regular code contributors. We will also follow the Git workflow pioneered by Vincent Driessen's "GitFlow".

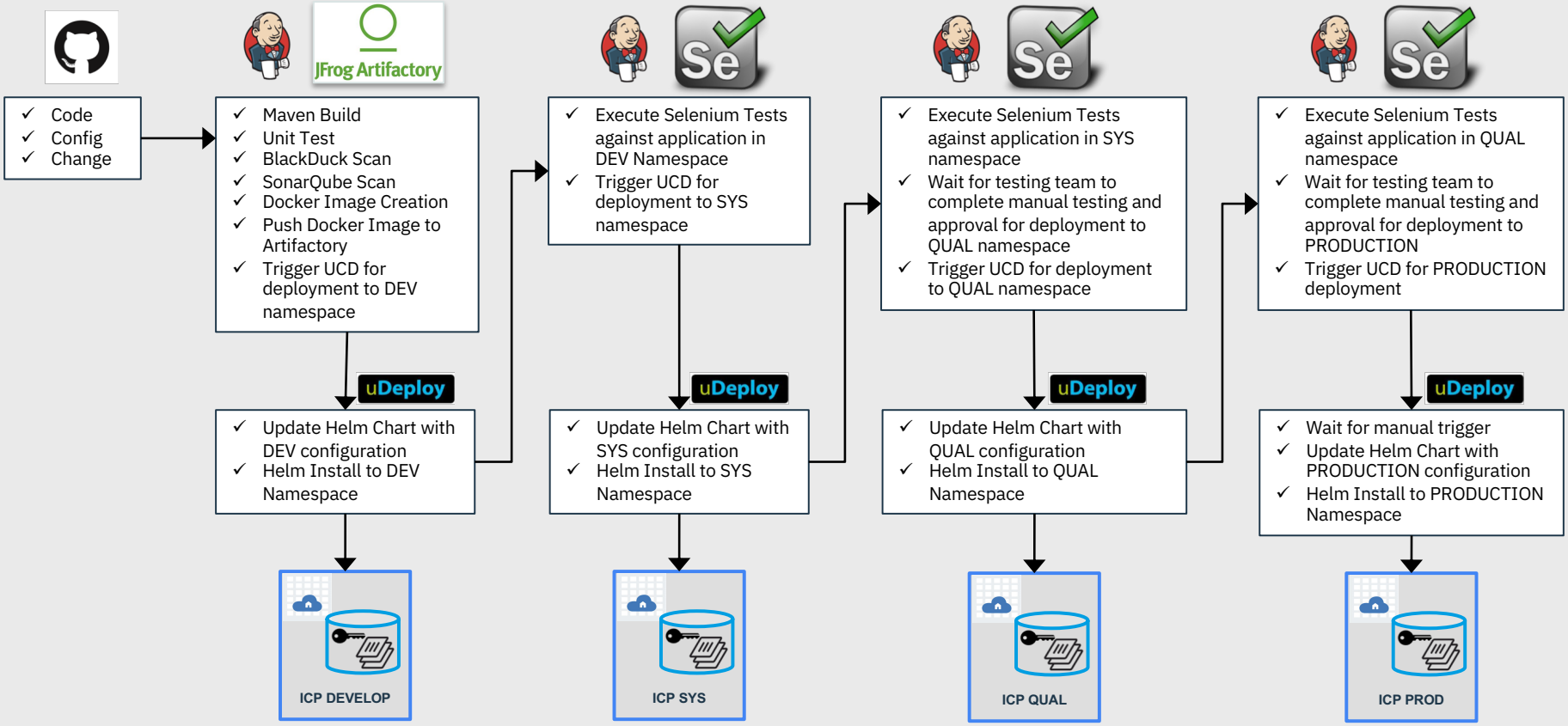
In this scenario, the Git repo will consist of a

- Release / Master branch,
- Project branch, and
- Fluctuating number of feature, release, and hotfix branches.

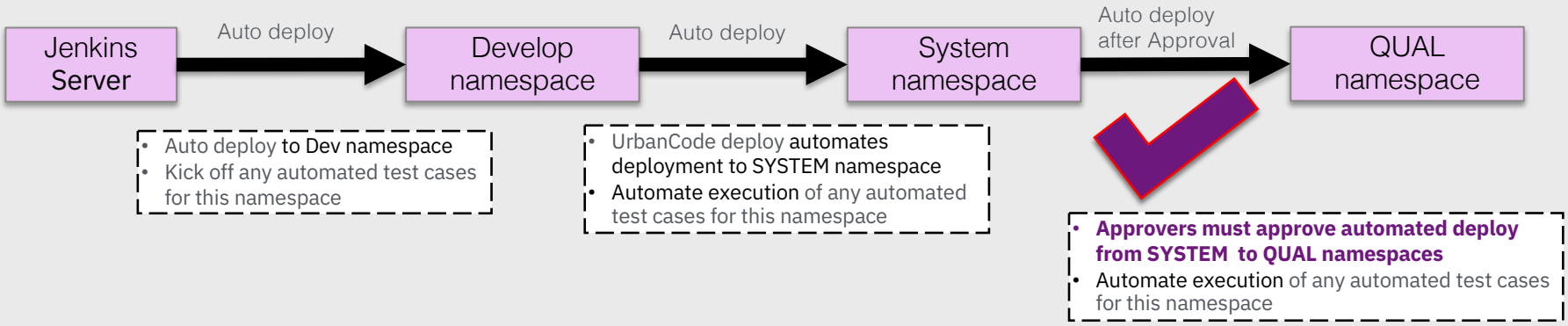


1. Developer modifies source in SANDBOX for a single line code change requiring very little testing (e.g., adding an image)
2. Developer pushes change to Remote Branch, Jenkins build pushed to UrbanCode which deploys update to DEVELOP
3. Developer completes Unit and Functional in their DEVELOP namespace, may also run any automated testing created by QA team, Add Quality tags to UCD Components under test
4. Developer, Tester, Lead are notified that new code promoted to SYSTEM, automated tests cases executed, updates Quality tags in UCD

Deployment to ICP Clusters



Approvals and Quality Gates



Approvals

- Deployment approvals are created in a process that specifies the job that needs approval and the role of the approver. When a request for approval is made, the users with the corresponding role
- UrbanCode Deploy uses Quality gates to ensure the code changes passes specific quality checks prior to being deployed
- Each stages usually has different requirements based on the needs of the team, governance
- For example, if the code satisfies the defined quality requirements; for instance,

1. the local build may need to run successfully and have all tests pass locally.
2. Build, test, and metrics should pass out of the central development environment, and then
3. automated and manual acceptance tests are needed to pass the system test.

In our case, we have identified one quality gate to pass is the one from the SYSTEM to QUAL