DevOps, CI + CD

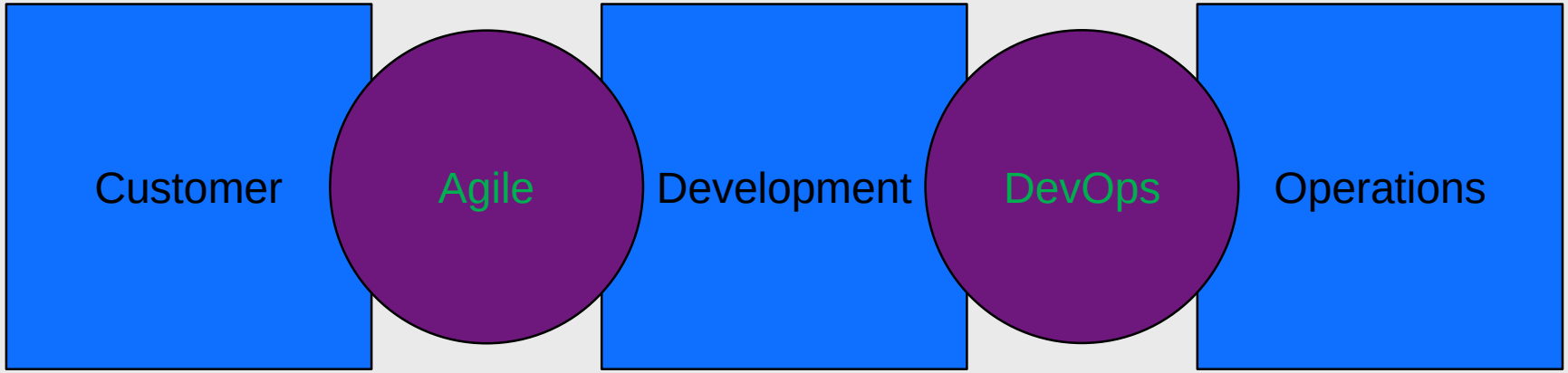
IBM Cloud

Definition

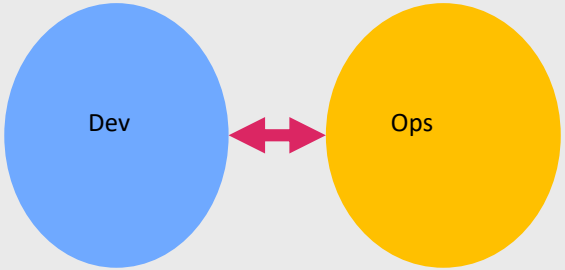
DevOps (*n.*) –

“DevOps is a philosophy, a **cultural shift** that **merges operations with development** and demands a **linked toolchain** of technologies to facilitate collaborative change. DevOps toolchains … can include dozens of non-collaborative tools, making the task of automation a technically complex and arduous one.”

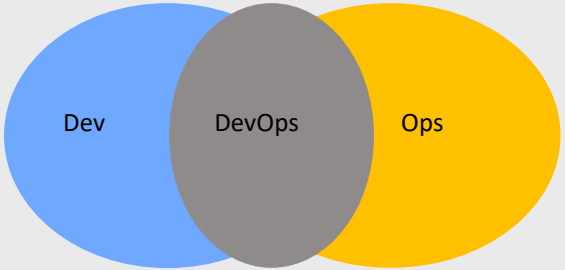
Gartner



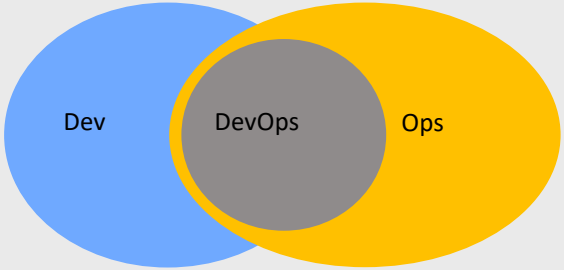
Evolving perspective on DevOps

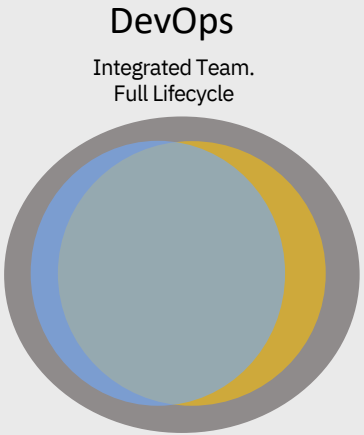


Large Enterprises
Traditional Workloads
(historical view)



Large Enterprises with
Traditional Workloads
Moving to Cloud





Newer companies with
Cloud-Native
Workloads

Continuous Integration

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

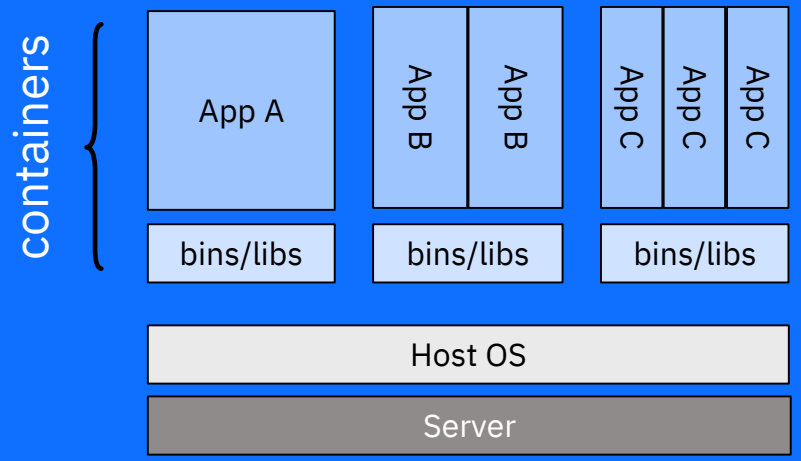
Continuous Deployment

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

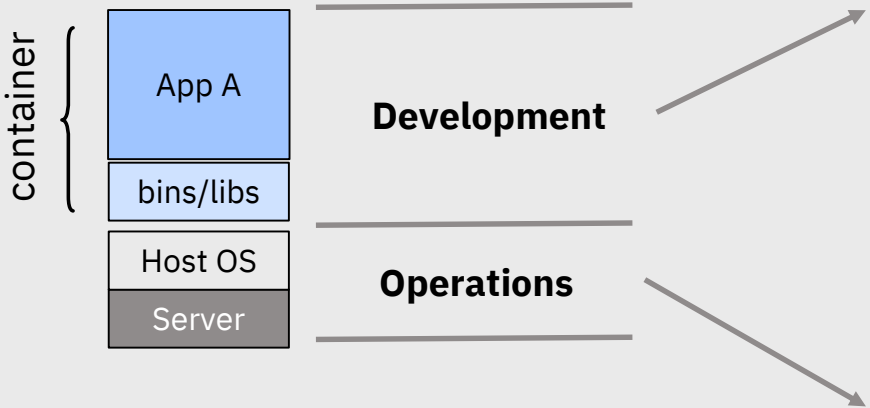
Containers solve a problem

- Ship more software
  - Accelerate development, CI and CD pipelines by eliminating headaches of setting up environments and dealing with differences between environments.
- Resource Efficiency
  - Lightweight containers run on a single machine and share the same OS kernel while images are layered file systems sharing common files to make efficient use of RAM and disk and start instantly
- App Portability
  - Isolated containers package the application, dependencies and configurations together. These containers can then seamlessly move across environments and infrastructures.



- ✓ Application processes on a shared kernel
- ✓ Simpler, lighter, and denser than VMs
- ✓ Portable across different environments
- ✓ Package apps with all dependencies
- ✓ Deploy to any environment in seconds
- ✓ Easily accessed and shared

Containers – Separation of Concerns



Jane the Developer

Worries about what's "inside" the container

- ✓ Code
- ✓ Libraries
- ✓ Package Manager
- ✓ Apps
- ✓ Data

All servers look the same

Todd the Ops Guy

Worries about what's "outside" the container

- ✓ Logging
- ✓ Remote access
- ✓ Monitoring
- ✓ Network config

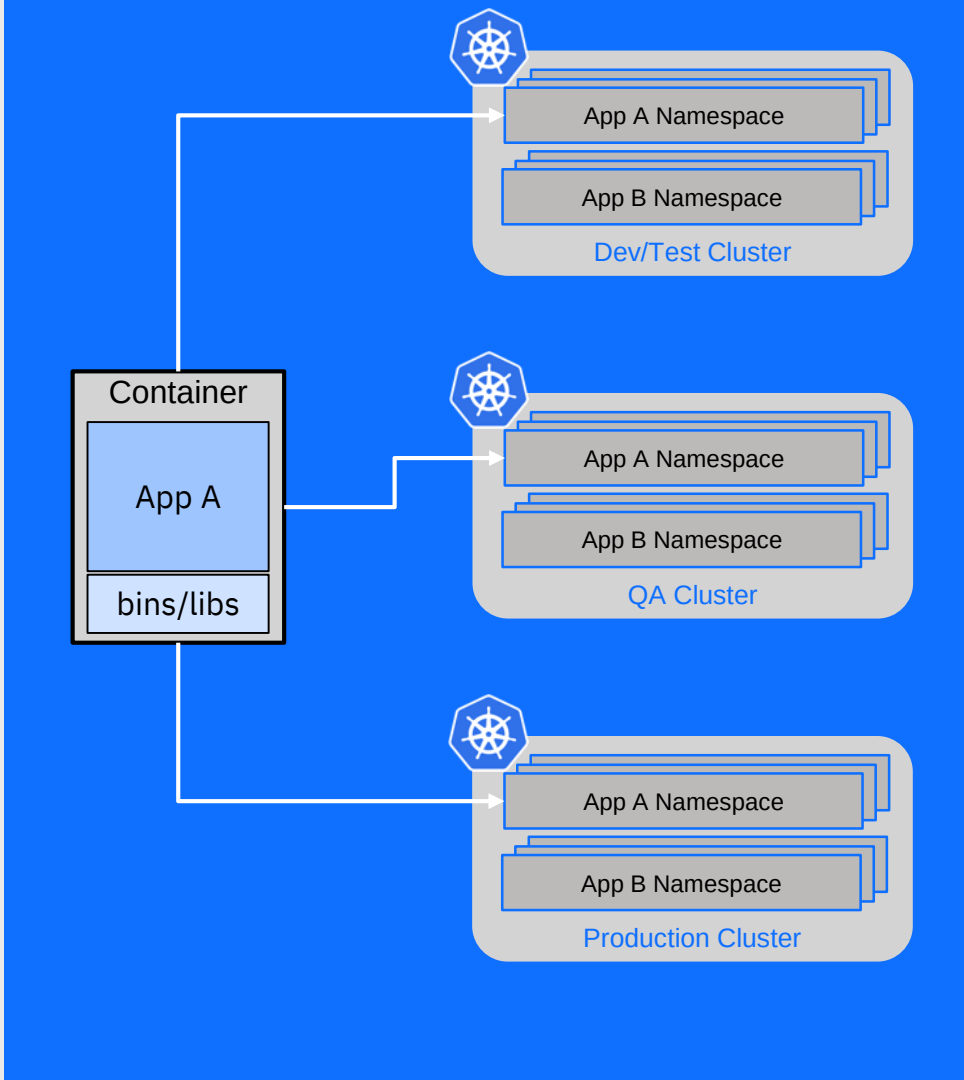
All containers start, stop, copy, attach, migrate, etc. the same way

Immutable Containers

"Build once – Deploy anywhere"

- The same container image is built once and is moved between environments
- Containers contain a "read-only" version of the application code
- Ensures that the exact same application code in used in every environment
- Environment specific configuration is injected in to the container at runtime

How does this work in a multi-architecture environment?



Kubernetes

Fully open source container orchestrator inspired and informed by Google's experiences and internal systems

Unified API for deploying web applications, batch jobs, and databases maintaining and tracking the global view of the cluster

Supports multiple cloud and bare-metal environments

Manage applications, not machines providing a better framework to support rolling updates, canary deploys, and blue-green deployments

Designed for extensibility

Rich ecosystem of plug-ins for scheduling, storage, and networking

Open source project managed by the Linux Foundation

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

Zero Downtime Deployment

Deploying a new version of an application to replace an old version without incurring a service outage

- Characteristics of zero downtime deployment
  - Application is always available, first old version, then new version
  - User does not experience an outage (a.k.a. downtime)
  - Old version and new version are both deployed at the same time – traffic is directed to both
  - User's only indication of new deployment is improved functionality
- Necessitated by DevOps' practice of frequent redeployment
  - Unacceptable for each new deployment to cause an outage

Implementing Zero Downtime Deployment – Blue Green Deployment

- Some challenges of continuous deployment
  - Application availability during cut-over
  - Rapidly reverting to prior version if necessary

- A zero-downtime deployment technique
  - Also known as Red Black Deployment (Netflix term)

- Two nearly identical production environments, called Blue and Green
  1. Deploy v1 (blue environment), which users use
  2. Deploy v2 (green environment)
  3. Run smoke tests and automated tests to verify the new environment before taking it live
  4. Shift client load from Blue to Green
  5. Delete Blue (or keep as a hot backup)

- Kubernetes incorporates “rolling updates” as its default method of deployment
  - Updates are incrementally rolled out pod by pod, so some pods have the “old” version and some have the ”new” version – traffic can be managed between these pods
  - Updates can be rolled back if there is a problem

“Blue green deployments”
https://www.ng.bluemix.net/docs/manageapps/updapps.html

Twelve Factors related to DevOps

I. **Codebase**: One codebase that is tracked in revision control, with many deployments

II. **Dependencies**: Explicitly declare and isolate dependencies

III. **Configuration**: Store Configuration in the environment

IV. **Backing services**: Treat backing services as attached resources

V. **Build, release, run**: Strictly separate build and run stages

VI. **Processes**: Execute the app as one or more stateless processes

VII. **Port binding**: Export services with port binding

VIII. **Concurrency**: Scale out using the process model

IX. **Disposability**: Maximize robustness with fast startup and efficient shutdown

X. **Development and production parity**: Keep development, staging, and production as similar as possible

XI. **Logs**: Treat logs as event streams

XII. **Admin processes**: Run administrative and management tasks as one-off processes

Twelve Factors related to DevOps

I. **Codebase**: One codebase that is tracked in revision control, with many deployments

II. **Dependencies**: Explicitly declare and isolate dependencies

III. **Configuration**: Store Configuration in the environment

IV. **Backing services**: Treat backing services as attached resources

V. **Build, release, run**: Strictly separate build and run stages

VI. **Processes**: Execute the app as one or more stat

VII. **Port binding**: Export services with port binding

VIII. **Concurrency**: Scale out using the process model

IX. **Disposability**: Maximize robustness with fast startup and efficient shutdown

X. **Development and production parity**: Keep development, staging, and production as similar as possible

XI. **Logs**: Treat logs as event streams

XII. **Admin processes**: Run administrative and management tasks as one-off processes

- **Delivery pipeline is triggered by SCM such as Git repos**
- **Delivery pipeline uses environment variables, both system and user**
- **Delivery pipeline builds deployment artifacts, packages immutable images, and deploys to lifecycle stages (e.g. test, stage, prod)**
- **Delivery pipeline deploys same immutable image to each lifecycle stage**

Continuous Integration

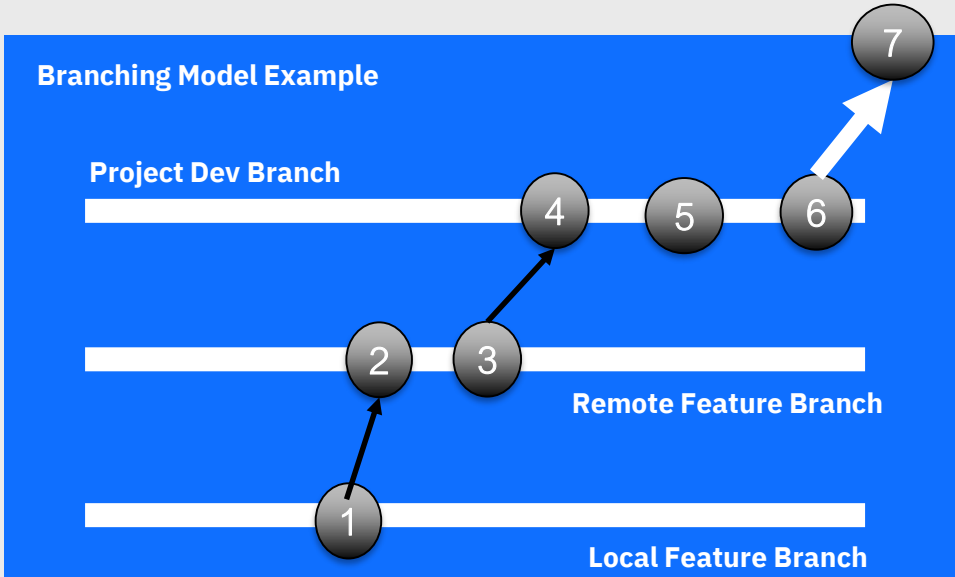
Git Branches

A single Git repository can support 3 - 8 regular code contributors.

Use the Git workflow pioneered by Vincent Driessen's "GitFlow".

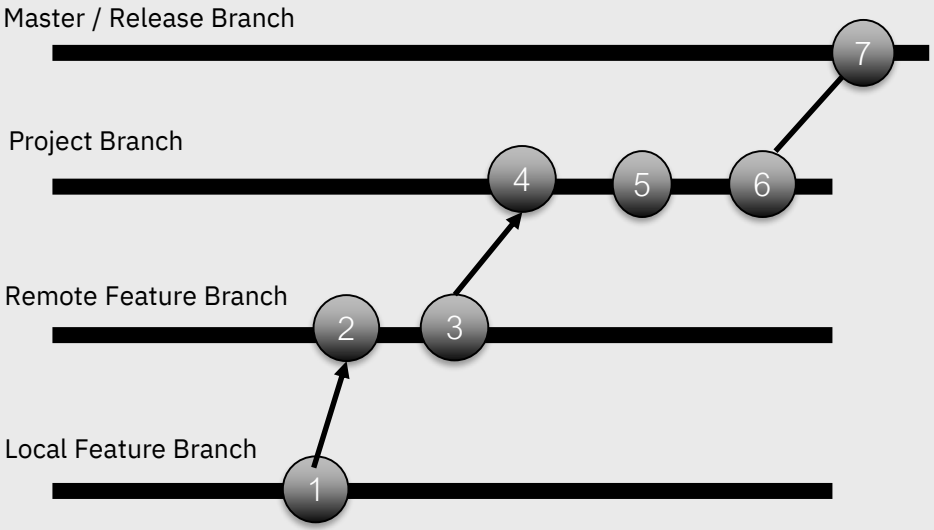
In this scenario, the Git repo will consist of a

- Master branch,
- Develop branch, and
- Fluctuating number of feature, release, and hotfix branches.



1. Developer modifies source for all code change in SANDBOX and perform unit and functional testing (e.g., adding an image)
2. Developer uses PULL REQUEST to push code change to DEVELOP branch where Team lead approves and Jenkins kicks off CI/CD
3. Automated testing along with some exploratory manual tests are used in DEVELOP namespace, Quality tags are added
4. CI/CD automation notifies squad on failure along with links to repos and Jira is updated automatically to alert team Leads / Sponsors.

Git Branches Mapped to ICP Namespaces



IBM Cloud Private DevOps Cluster

QUAL NAMESPACE
(ICP APPLICATION Namespace)

SYSTEM NAMESPACE
(ICP TEAM Namespace)

DEVELOP NAMESPACE
(ICP personal Namespace)

(Local Development)

Jenkins

Popular open-source framework for Continuous Integration

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

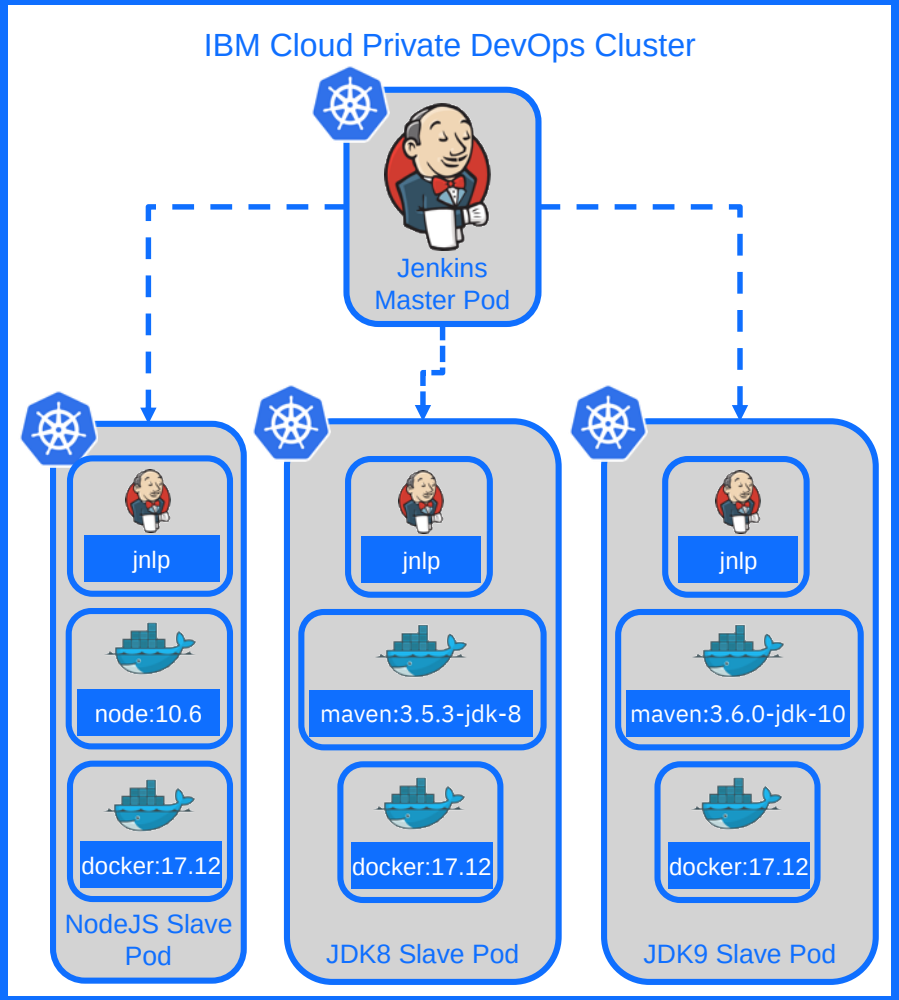
Generally used for:

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

Jenkins in Kubernetes

Jenkins runs in the DevOps IBM Cloud Private Cluster

- Deployed using a Helm Chart
- No need for static, dedicated Jenkins Slave Virtual Machines
- Dynamically provisioned slaves run as Kubernetes Pods
- Slaves are used for the duration of the job and destroyed after their builds are complete, ensuring masters have steady access to clean workspaces and minimizing builds' resource footprint.

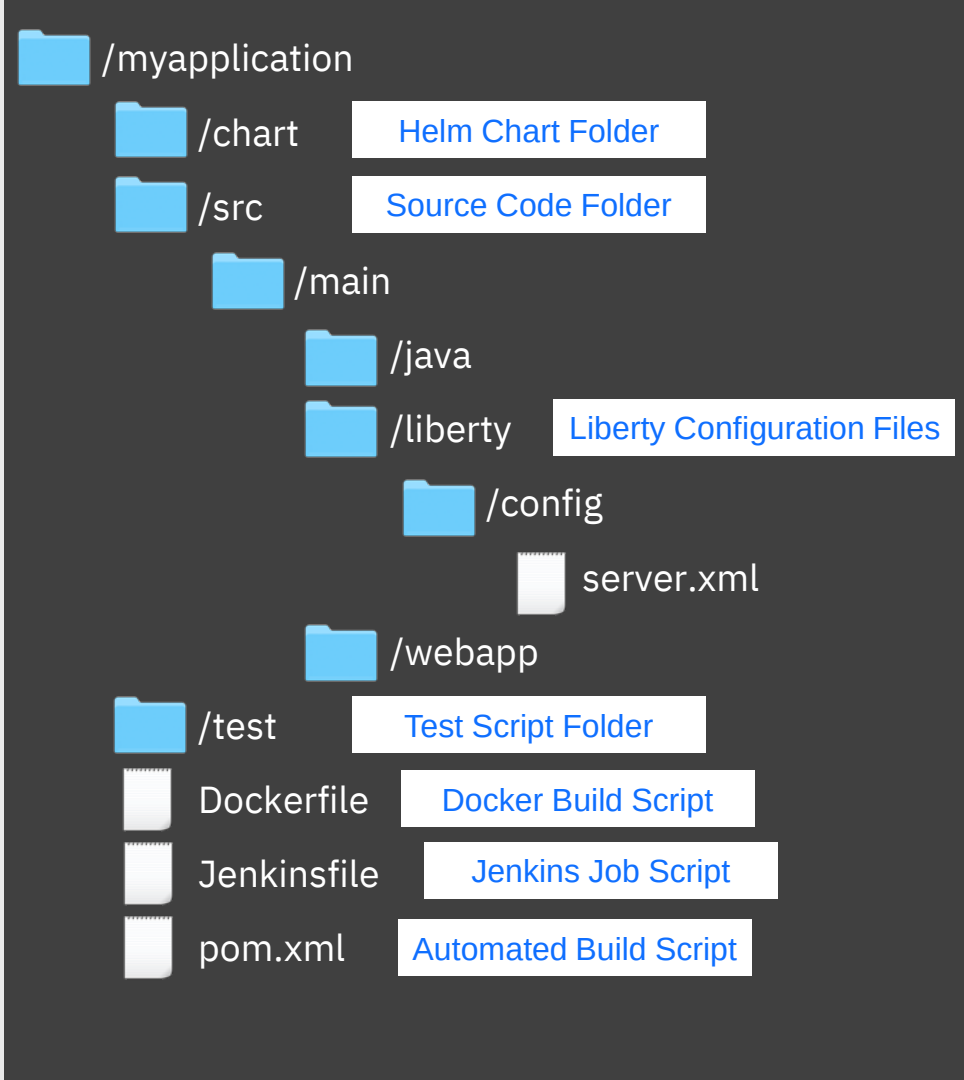


Git Project Structure

Project Git Repository should contain:

- Application Code
- Build Scripts
- Runtime Configuration*
- Test Scripts
- Helm Chart
- General Environment Specific Configuration*

- * Don’t store Passwords in Git!



Automating Build: Jenkinsfile

Jenkinsfile: Script file that is used to describe a Jenkins Pipeline. The file can be stored in a source control system and is loaded whenever the pipeline is triggered.

podTemplate: defines a dynamic pod that can be used by Jenkins as a slave.

node: defines where this portion of the job will be performed

Extract stage: the **jnlp** container is used to extract the source code from SCM in to the shared workspace

Maven build stage: a dedicated **maven** container is used to compile the source code

Docker stage: a dedicated **docker** container is used to build the docker image and push it to the docker registry

```
def gitCommit
def volumes = [ hostPathVolume(hostPath: '/var/run/docker.sock', mountPath: '/var/run/docker.sock') ]
volumes += secretVolume(secretName: 'docker-registry-secret', mountPath: '/docker_reg_sec')
podTemplate(label: 'icp-liberty-build',
    slaveConnectTimeout: 600,
    nodeSelector: 'beta.kubernetes.io/arch=amd64',
    containers: [
        containerTemplate(name: 'maven', image: 'maven:3.5.3-jdk-8', ttyEnabled: true, command: 'cat'),
        containerTemplate(name: 'docker', image: 'docker:17.12', ttyEnabled: true, command: 'cat')
    ],
    volumes: volumes
)
{
    node ('icp-liberty-build') {
        stage ('Extract') {
            checkout scm
            gitCommit = sh(script: 'git rev-parse --short HEAD', returnStdout: true).trim()
            echo "checked out git commit ${gitCommit}"
        }
        stage ('maven build') {
          container('maven') {
            sh '''
            mvn test install
            '''
          }
        }
        stage ('docker') {
          container('docker') {
            def imageTag = "mycluster.icp:8500/default/jenkinstest:${gitCommit}"
            sh """
            ln -s /docker_reg_sec/.dockercfg /home/jenkins/.dockercfg
            mkdir /home/jenkins/.docker
            ln -s /docker_reg_sec/.dockerconfigjson /home/jenkins/.docker/config.json
            docker build -t $imageTag .
            docker push $imageTag
            """
          }
        }
    }
}
```

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

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

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
```

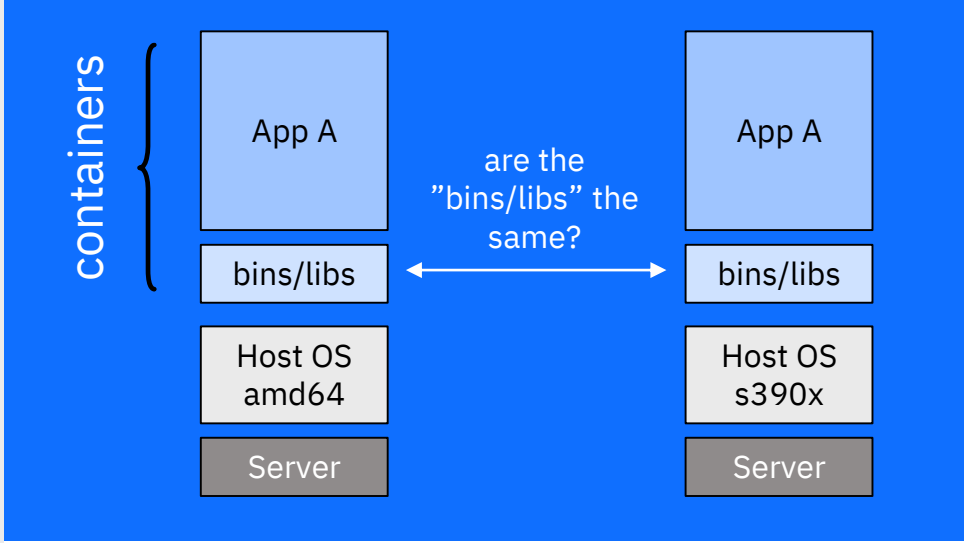
Multi-architecture Docker

Docker images can only be **run** on the **architecture** they were **built** on

- an image built on amd64 (x86) contains binaries and libraries for amd64 and will only run on an amd64 host
- an image built on s390x (zLinux) contains binaries and libraries for s390x and will only run on an s390x host

In an environment with amd64 and s390x worker nodes, pipelines must execute **docker build** on both amd64 **and** s390x

- each image will have a different tag:
  - **myimage:amd64**
  - **myimage:ppc64le**
  - **myimage:s390x**



Q: How does websphere-liberty:webProfile7 work on multiple architectures without different tags?

A: Manifests

image: myimage:latest
manifests:
- image: myimage:ppc64le platform: architecture: ppc64le ...
- image: myimage:amd64 platform: architecture: amd64 ...
- image: myimage:s390x platform: architecture: s390x ...

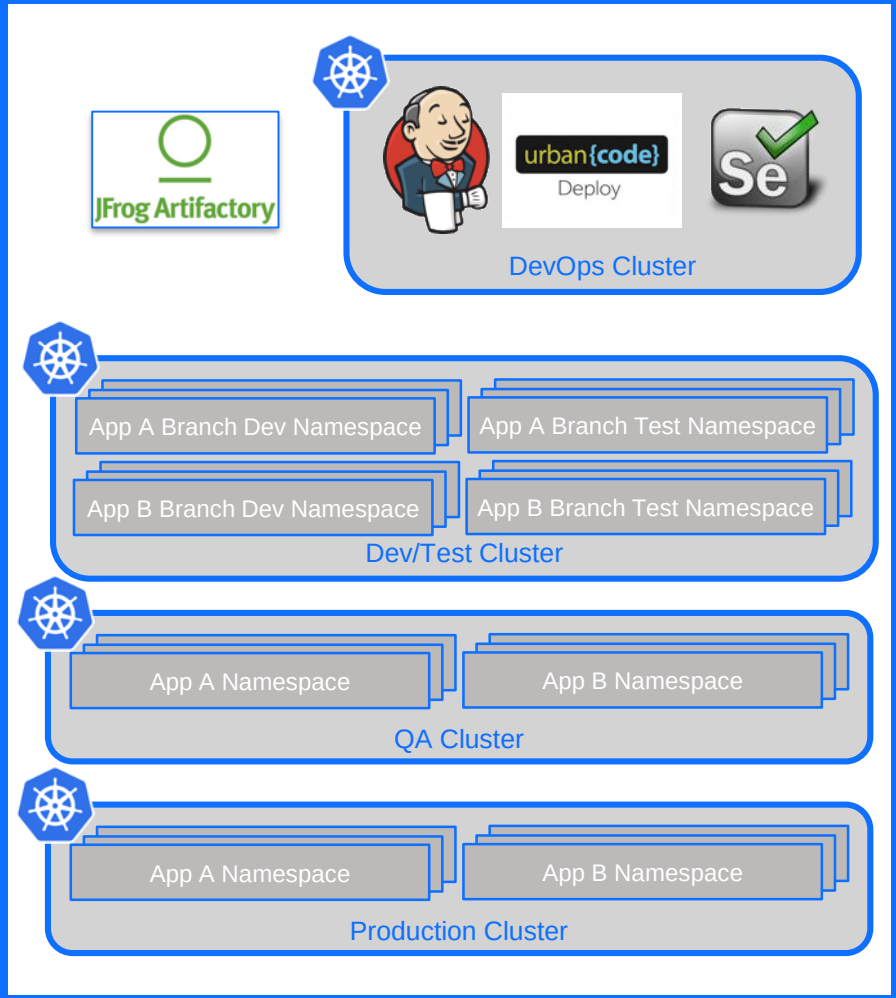
https://github.com/estesp/manifest-tool

Continuous Deployment

IBM Cloud Private Clusters

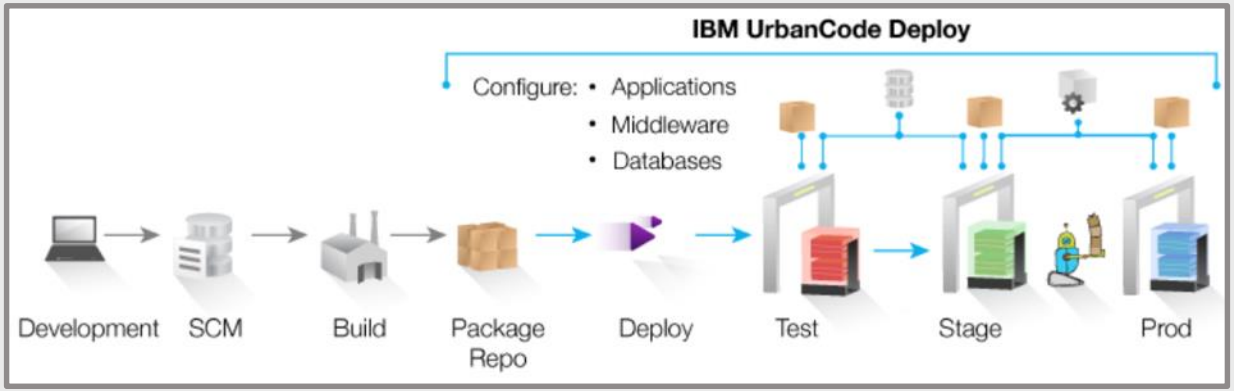
Multiple IBM Cloud Private Clusters are often used:

- DevOps Cluster hosts the DevOps tools:
  - Jenkins
  - UrbanCode Deploy
  - Automated Testing Tools
- Shared Dev/Test Cluster
- QA Cluster
- Production Cluster



UrbanCode Deploy

IBM UrbanCode Deploy is a tool for automating application deployments through environments.



- Automated, consistent deployments and rollbacks of applications
- Automated provisioning, updating, and de-provisioning of cloud environments
- Orchestration of changes across servers, tiers and components
- Configuration and security differences across environments
- Clear visibility: what is deployed where and who changed what
- Integrated with middleware, provisioning and service virtualization

UrbanCode Deploy with IBM Cloud Private

UrbanCode Deploy makes the management and deployment of Kubernetes Helm charts simple.

- Manage different releases across multiple environments
- Dynamically replace values in Helm Charts.
- Audit tracking shows which versions are deployed in each environment and who deployed them
- Access control and quality gates may be added to restrict who may deploy certain Helm charts.
- Charts can be compared using UrbanCode Deploy, highlighting differences.

UCD helps to solve some of the more complex issues that Jenkins and other tools can’t:

- Deployment to multiple clusters, environments and clouds
- How do I visualize which version of my *application* is deployed across many environments
- How do I manage my security tokens for accessing multiple clusters/environments (including Production)
- How do I manage my cluster-wide ConfigMaps, Secrets and Certificates

Automated Testing with ICP

https://techbeacon.com/7-steps-choosing-right-devops-tools

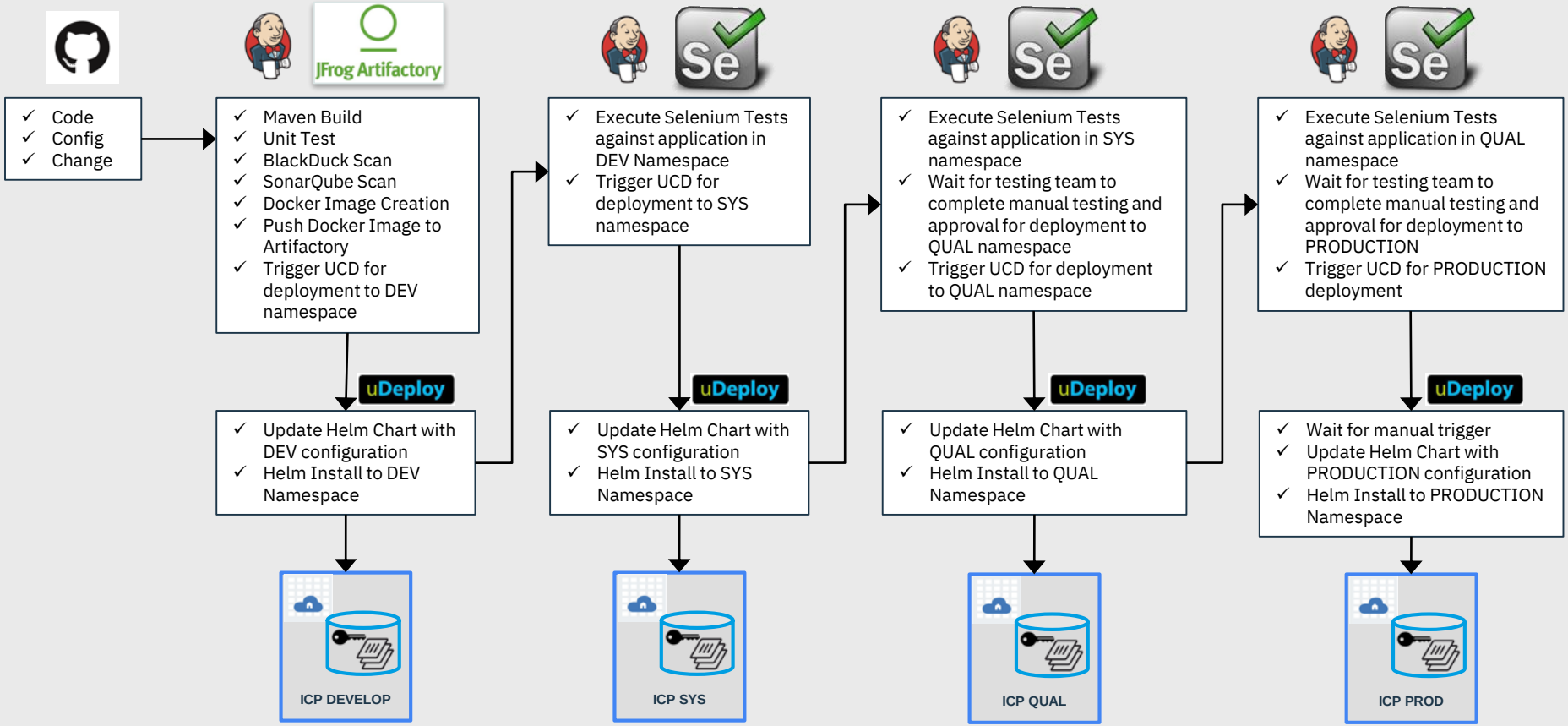
Here are some good testing practices that we found worked well

1. Adopt Shift-Left testing practices such as TDD and BDD to accelerate development in Dev, QA, Perf namespaces
2. Focus on key automation testing tools such as SonarQube, Selenium, Postman, API testing
3. Automate Test infrastructure deployment using HELM to allow testers to dynamically provision testing capability rapidly while allowing management of scare resources
4. Create and Deploy specialized Jenkins containerized agents to dynamically provision testing capability as required

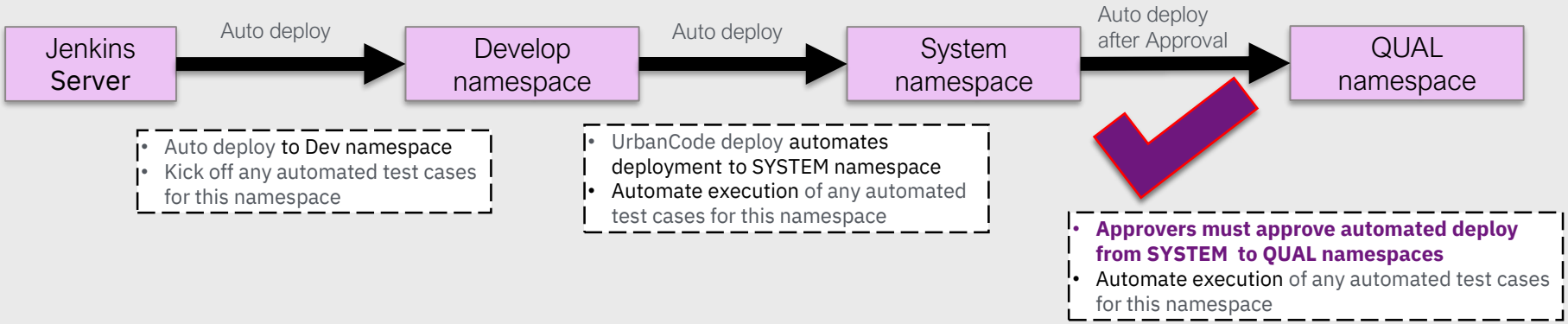
Terminology

No	Term	Description
1	git	Git is a distributed version control system used for tracking changes in computer files and coordinating work on those files among multiple people. Created by Linus Torvalds, git has become one of the most widely used source code management in software development
2	Jenkins	Jenkins is an open source continuous integration (CI) / continuous delivery (CD) tool used for software development. Jenkins has a large number of integrations, called “plugins”, which are used to extend its functionality such as integrations with git, Artifactory, UrbanCode, SonarQube, etc.
3	Black Duck	Black Duck's multi-factor open source detection capabilities used to identify vulnerability, and license information to mitigate security and license compliance risks. Black Duck is often integrated with your , existing DevOps tools and processes.
4	SonarQube	SonarQube is an open source continuous inspection tool used for code quality, automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on 20+ programming languages. SonarQube provides fully automated analysis and integration with Maven, Ant, Gradle, MSBuild and continuous integration tools such as Jenkins
5	Artifactory	Artifactory is a Binary Repository Manager for software artifacts. It offers advanced proxying, caching and security facilities and provides a robust, reproducible build environment when using Maven, Ant/Ivy, Gradle or parallel build technologies.
6	UrbanCode Deploy	UrbanCode Deploy is built to support mission-critical deployments to thousands of servers in numerous data centers. Master server clustering support provides high availability and horizontal scalability of the deployment automation tool. UrbanCode Deploy uses light-weight deployments agents which provide an immediate presence on or near the target.

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

DevOps Roles

Roles	Description of Responsibilities
Developer	Developers today must take a more holistic, operational mindset approach to their work. Developers often must write code in their "local system", using tools like git, maven and IDEs. They also • are responsible for designing and coding solutions based on user requirements • must use SCM tools, like git, to version control their code changes, and tools like Maven and Jenkins to build their code. • must understand full-stack engineering, including deployment automation, and possess an understanding of the infrastructure supporting their application and the non-functional characteristics of the application.
Automation Engineer	An automation engineer is responsible for • identifying and automating manual processes to promote continuous integration, testing, staging, and then deployment and operations. • for defining the Continuous Integration / Continuous Delivery (CI/CD) processes and implementing them using such tools as git, Jenkins, Artifactory, Maven, Black Duck and others In addition, automation engineers work closely with cloud operations to streamline workflows (such as user administration, security and infrastructure provisioning) to help the organization adopt infrastructure as code.
Developer Lead	While the exact responsibilities Developer Lead may vary from company to company, in general they are responsible for the underlying software architecture, overseeing the work being done by the other developers on the team and often act as the Integration when critical pieces of code must be delivered and tested.
Tester	A Tester is responsible for verifying and validating the application being tested. Where verification is about proving that the application is working properly and doing the job that the developer intended. While validation, on the other hand, confirms that the program is performing the task requested by the customer.

DevOps Roles – cont.…

Roles	Description of Responsibilities
Project Lead	A Project Lead often plays multiple roles of such as Project Manager, Team Lead and sometimes implementer as part of an agile team. • The Project Lead is intimately familiar with the product's user needs and related features. • They know the team's implementation plan and can creatively consider alternatives if constraints challenge the original plan. • They also must be able to anticipate constraints early on and manage scope appropriately.
Application Owner	An Application Owner is the individual, usually from the Business organization, responsibility to ensure that the software projects, which make up the application, meet the specified objective set by the business organization and user requirements established for that application.
Release Coordinator	A Release Coordinator typically is responsible for planning and coordination of all phases and activities involved in the release of a system update into the production environment. Typically this roles: • facilitates release meetings and develops the release recommendation and release deployment plan • identify priorities, conflicts, dependencies, and risks for the release. • Works with other Ops resources such as the SRE, Project Lead and Developer Leads to ensure that the plans are in place to mitigate those risks (e.g., rollback plans)
Site Reliability Engineer	A Site Reliability Engineer (SRE) will spend up to 50% of their time doing "ops" related work such as issues, on-call, and manual intervention. Typically the Site Reliability Engineer role will involve some or all of the following tasks • eliminating performance bottlenecks by refactoring services into more scalable units. • isolating failures through use of the cloud design patterns like the 'circuit breaker' and 'bulkhead' • creating runbooks to ensure fast service recovery. • automation of day to day ops processes.