Continuous Delivery Meets OpenShift

Yuxiang Zhu <yuxzhu@redhat.com>
Shiyang Wang <shiywang@redhat.com>
Jan 26th, 2019
PROBLEMS ONE
Environments
Traditional Environments
Can not reproduce bug easily

- Some of our internal projects, for some historical reason, it’s hard to keep dev, qe, stage, and prod environments similar.
Traditional Environments
Can not reproduce bug easily

- Some of our internal projects, for some historical reason, it’s hard to keep dev, qe, stage, and prod environments similar.
Traditional Environments

Can not reproduce bug easily

- Some of our internal projects, for some historical reason, it’s hard to keep dev, qe, stage, and prod environments similar.
Traditional Environments
Can not reproduce bug easily

- Some of our internal projects, for some historical reason, it’s hard to keep dev, qe, stage, and prod environments similar.
Traditional Environments
Can not reproduce bug easily

- Some of our internal projects, for some historical reason, it’s hard to keep dev, qe, stage, and prod environments similar.
Traditional Environments

Can not reproduce bug easily

- Some of our internal projects, for some historical reason, it’s hard to keep dev, qe, stage, and prod environments similar.

![Diagram showing the process of code change, release, and bugs in traditional environments with dev and qe environments unable to reproduce bugs](image-url)
Traditional Environments
Are hard to clone && provide && rebuild same env easily

- Your customers may want to clone your environment for tests.
Traditional Environments

Are hard to clone && provide && rebuild same env easily

- Your customers may want to clone your environment for tests.
- You may want to clone all service dependencies of your project to other teams.
Traditional Environments

Are hard to clone && provide && rebuild same env easily

- Your customers may want to clone your environment for tests.
- You may want to clone all service dependencies of your project before releases.
- Your environments could be lost due to hardware failures.
Traditional Environments
Lowering performance/resource utilization

- Many projects already have pipelines for continuous integration.
- As more and more automations added to your pipelines, how do you scale up or scale out your pipeline workloads?
Traditional Environments

Lowering performance/resource utilization

- Many projects already have pipelines for continuous integration.
- As more and more automations added to your pipelines, how do you scale up or scale out your pipeline workloads?
Traditional Environments
Lowering performance/resource utilization

- Many projects already have pipelines for continuous integration.
- As more and more automations added to your pipelines, how do you scale up or scale out your pipeline workloads?
PROBLEMS TWO
Service dependencies
Service dependencies

- Your service depends on services of someone else.
- Someone else’s service depend on your service.
- How do you ensure that an upgrade of a service will not break the others?
Service dependencies

- Your service depends on services of someone else.
- Someone else’s service depend on your service.
- How do you ensure that an upgrade of a service will not break the others?
Service dependencies

- Your service depends on services of someone else.
- Someone else’s service depend on your service.
- How do you ensure that an upgrade of a service will not break the others?

![Diagram of service dependencies]

- Code change frequency: couple mins/hours
- Release period: 4-5 days
Service dependencies

Makes release cadence slow and code change large

- Your service depends on services of someone else.
- Someone else’s service depend on your service.
- How do you ensure that an upgrade of a service will not break the others?

Code change frequency: couple mins/hours
Release period: 4-5 days
HOW DO YOU SOLVE THEM?
CONTINUOUS DELIVERY + OPENSSHIFT!
Continuous Integration IS NOT Enough!

developers prepares to deploy as operators race to defend themselves
Continuous Delivery & Continuous Deployment

Continuous delivery is a series of practices designed to ensure that code can be rapidly and safely deployed to production by delivering every change to a production-like environment and ensuring business applications and services function as expected through rigorous automated testing.

*It doesn't mean every change is deployed to production ASAP.*
Automation!

- Build
- Deploy
- Test
CONTINUOUS DELIVERY MEETS OPENSSHIFT
OpenShift + Continuous Delivery

- Containers minimized the differences between environments.
- Deploying a production-like environment is as easily as one click.
- Leveraging the hardware resource of whole cluster as workload goes up without considering hardware provisioning and upgrade.
- OpenShift provides features that help you implement pipelines.
OpenShift Pipeline

- is a combination of the Jenkins Pipeline Build Strategy, Jenkinsfiles, and the OpenShift Domain Specific Language (DSL)
- is an extension to the original Jenkins Pipeline
- Steps are also defined in Jenkinsfile.
- has built in synchronization between OpenShift builds and Jenkins builds.
CASE STUDY: CD PIPELINES FOR WAIVERDB
High Level Workflow

- Git repo
- Static checks
- Unit tests
- RPM build
- RPM check
- Artifacts
- Container build
- Registry
- Functional tests
- Option 1
- Option 2
- Devel
- Criteria
- Stage
- Criteria
- Prod
- :latest
- :stage
- :prod

Criteria:latest :stage :prod
Jenkinsfile

A Jenkinsfile

- defines the steps of your pipeline as code.
- is a text file written in Jenkins Pipeline DSL, a subset of the Groovy Programming Language.
- has two syntaxes, Declarative (introduced in Pipeline 2.5) and Scripted Pipeline.

Jenkinsfile (Declarative Pipeline)

```groovy
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building..'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```
Demo

```yaml
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "demo-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfile: |
        pipeline {
          agent {
            kubernetes {
              # ...
        ```

https://gist.github.com/vfreex/88d3bff38d1ad5bd7ec9af11fc130d68
Pipeline Jobs are OpenShift Apps!

The URL of Git repo, destination registries, and credentials are made as parameters of Template.
You can instantiate your own pipeline for your own fork of WaiverDB. Multiple pipelines can coexist within the same OpenShift project.

```
NAME=waiverdb-dev-pipeline  # Use different names for different pipeline instances
oc process --local -f ./waiverdb-dev-pipeline-template.yml
  -p NAME="$NAME"
  -p WAIVERDB_GIT_REPO="https://pagure.io/forks/<username>/waiverdb.git"
  -p WAIVERDB_GIT_REF="<your devel branch>"
  -p WAIVERDB_DEV_IMAGE_DESTINATIONS="docker.io/<username>/waiverdb:latest"
  -p PAGURE_DOC_REPO_NAME="forks/<username>/waiverdb"
oc apply -f -
```
Parallel Builds Are Permitted

Every pipeline build will spawn a new Jenkins slave pod and separated test environments. A unique tag for the built image will be used internally to avoid conflicts. It can be integrated with the PR review process.

```
oc start-build waiverdb-dev-pipeline -e WAIVERDB_GIT_REF=master
oc start-build waiverdb-dev-pipeline -e WAIVERDB_GIT_REF=$COMMIT
oc start-build waiverdb-dev-pipeline -e WAIVERDB_GIT_REF=$PR_1
oc start-build waiverdb-dev-pipeline -e WAIVERDB_GIT_REF=$PR_2
```
Never lose Your Build Environment
The Jenkins cluster can be safely rebuilt without losing the build environment.

- Pipeline options are persist in OpenShift manifests.
- Jenkins slaves are run as OpenShift Pod, which is expected to be ephemeral. This is done by using the Jenkins Kubernetes Plugin.
- Credentials are persist as OpenShift Secrets. Secrets with the special label credential.sync.jenkins.openshift.io=true will be synced into the Jenkins master by the Jenkins OpenShift Sync Plugin.
- Resulting containers are stored in external Container Registries.
Benefits

1. Integrating your pipelines with OpenShift with OpenShift Pipeline DSL.
2. Leveraging the hardware resource of whole cluster by running Jenkins master, slaves, and all environments of your app fully on OpenShift, reducing the cost of maintenance.
3. Minimizing the difference between dev/test/stage/prod environments, never losing any environment.