GitOps from Development to Production
Continuous Integration and Continuous Delivery on OpenShift

Johana Limka
Associate Solution Architect, Red Hat

Mark Roberts
Principal Solution Architect, Red Hat
Objective

To create an automated build and deploy process
GitOps Principles

- The system is described declaratively
- The desired state is versioned in Git
- Approved changes can be applied automatically
- A controller exists to detect and act on drift
Objective

To create an automated build and deploy process in which the assets are stored in and managed from a Git repository
Source content and objectives

- From source content to running microservice
  - Source code of application
  - Container images (both a target and a source asset)
  - Kubernetes resources - services, configuration maps, secrets etc.

- Multiple steps to manage
  - Software build process
  - Container image creation and storage
  - Deployment of container image from storage
  - Application of Kubernetes resources

- Two major phases
  - Build - Continuous integration
  - Deploy - Continuous delivery
Source code to container image cycle - Continuous Integration

- Store source code in a Git repository
- Pull the source code and execute a build process
  - The build definition assets are also stored in a Git repository
- Create a new container image as part of the build process
  - This process will use a source container image that may be stored in the container registry or an external registry
- Push the new container image to the container registry with a new identification tag
Container image deployment - Continuous Delivery

- Store resource definitions in a Git repository
- Pull the resource definitions from Git
- Create Kubernetes resources based on resource definition
  - Refer to the container image in the container registry
- Deploy assets based on the desired state of Kubernetes resources
- Result: A new running application
- Two inputs that can drive change
  - Deployment configuration resources
  - Container images
**OpenShift GitOps**

- Delivered as an operator on OpenShift clusters
- Based on ArgoCD open source project
- Synchronisation maintained between the Git repository and resulting assets
- ‘Application’ definition Git repository -> Kubernetes resource

**OpenShift Pipelines**

- Delivered as an operator on OpenShift clusters
- Based on Tekton open source project
- Cloud native continuous integration process
- Tasks execute in isolated pods
  - Each step of a task is a specific and unique container
- Pipeline and task definitions stored in Git repository
- Kubernetes resources based on Git content managed by OpenShift GitOps
Git pull request process

- To merge changes back to main create a pull request
  - Request that a person with appropriate permission ‘pulls’ the changes from the development branch
- Branch protection rules
  - Require the pull request is used
  - Require approvals prior to merge
  - Require status checks on commits evaluate to ‘pass’
  - etc.

Diagram:
- Development user creates the pull request - target is ‘main’ branch
- Branch protection rules are evaluated - require commit status of ‘Success’
- Appropriate user pulls the commit(s)
Update the deployment content with new image tag

- New image tag identifier
- To be used in the deployment process
- Update process
  - Update the deployment asset
  - Commit to Git

Solution? Kustomize.io
Kustomize

- A text replacement and resource build process
- Template file -
  - ‘Deployment time’ text replacement process
  - Text patching directives

### Git repository files

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: myapp
namespace: myapp-prod
spec:
  template:
    spec:
      containers:
        - name: myapp
          image: quay.io/<repository>/myapp-runtime
```

### Kustomize file

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
- service.yaml
- route.yaml
- configmaps.yaml
images:
  - name: quay.io/<repository>/myapp-runtime
    newTag: 27e8
```

### Deployment time ‘dynamic files’

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: myapp
namespace: myapp-prod
spec:
  template:
    spec:
      containers:
        - name: myapp
          image: quay.io/<repository>/myapp-runtime:27e8
```

```
kustomize edit set image quay.io/<repository>/myapp-runtime:27e8
```
Pipeline process for development and QA

- Use a simple update and deploy process for development
- Use a pull request to create a review point for QA
QA pipeline - Use a pull request to manage the release

1. Git clone source
2. Build JAR file
3. Create runtime container image
4. Image build check
5. Update and commit Kustomize file - Dev
   - Causes ArgoCD to trigger the deployment of the updated resources

   - Image validation - gate

6. Set commit status to 'pending'
7. Git clone resources (qa-ready branch)
8. Configure deployment assets
9. Execute the Kustomize build process to perform text replacements
10. Resource deployment check
    - Resource validation - gate

11. Set commit status to 'success'
12. cd repository qa-ready branch
13. New commit
14. Enable the pull request to merge the QA changes from qa-ready to main branch

15. cd repository main branch
16. Update and commit Kustomize file - QA & create a pull request to merge changes to main branch
17. Blocks the merge of the pull request until resources are validated

18. Perform ACS resource deployment check against policies
Pipeline in action - before resources are approved

Update and commit Kustomize file - QA & create a pull request to merge changes to main branch

Set commit status to ‘pending’

Git clone resources (qa-ready branch)

Configure deployment assets

Resource deployment check

Set commit status to ‘success’

Pull request summary with orange ‘pending’ indicator

Pull request detail showing that it cannot be merged yet

Update and commit Kustomize file - QA & create a pull request to merge changes to main branch

Set commit status to ‘pending’

Git clone resources (qa-ready branch)

Configure deployment assets

Resource deployment check

Set commit status to ‘success’
Pipeline in action - after resources have been approved

1. Update and commit Kustomize file - QA & create a pull request to merge changes to main branch
2. Set commit status to 'pending'
3. Git clone resources (qa-ready branch)
4. Configure deployment assets
5. Resource deployment check
6. Set commit status to 'success'

Pull request summary with green ‘success’ indicator

Pull request detail showing that it can be merged yet
QA pipeline - Use a pull request to manage the release

1. Git clone source
2. Build JAR file
3. Create runtime container image
4. Update and commit Kustomize file - Dev

- Image build check

- Update and commit Kustomize file - QA & create a pull request to merge changes to main branch
- Set commit status to 'pending'
- Git clone resources (qa-ready branch)
- Configure deployment assets
- Resource deployment check
- Image validation - gate
- Resource validation - gate

- QA Deployment process

- cd repository main branch

Causes ArgoCD to trigger the deployment of the updated resources
Extending the pipeline to production

1. Git clone source
2. Build JAR file
3. Create runtime container image
4. Image build check
5. Update and commit Kustomize file - Dev
6. Push image to container registry (quay)
7. cd repository main branch
8. Trigger ArgoCD deployment to Dev
9. Webhook URL
10. Trigger ArgoCD deployment to QA
11. QA Deployment process
12. QA Deployment process
13. Push image to container registry (quay)
14. Git clone
15. Build JAR file
16. Create runtime container image
17. Image build check
18. Update and commit Kustomize file - Dev
19. Push image to container registry (quay)
20. cd repository main branch
21. Trigger ArgoCD deployment to Dev
22. Webhook URL
23. Trigger ArgoCD deployment to QA
24. QA Deployment process
25. QA Deployment process
26. Push image to container registry (quay)
27. Git clone
28. Build JAR file
29. Create runtime container image
30. Image build check
31. Update and commit Kustomize file - Dev
32. Push image to container registry (quay)
33. cd repository main branch
34. Trigger ArgoCD deployment to Dev
35. Webhook URL
36. Trigger ArgoCD deployment to Production
37. Production Deployment process
38. Production Deployment process
39. Push image to container registry (quay)
40. Git clone
41. Build JAR file
42. Create runtime container image
43. Image build check
44. Update and commit Kustomize file - Dev
45. Push image to container registry (quay)
46. cd repository main branch
47. Trigger ArgoCD deployment to Dev
48. Webhook URL
49. Trigger ArgoCD deployment to Production
50. Production Deployment process
Summary

OpenShift Pipelines delivers a cloud-native continuous integration process

OpenShift GitOps delivers a deployment automation process driven by content in Git repositories

Git provides a secure source code repository in which branch protection rules govern when content is merged to branches

Kustomize provides a controlled mechanism to update and patch Kubernetes resources for each environment

Most important: A structured and controlled process that is understood by the team
Thank you

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos
twitter.com/RedHat