



Version 3.3

04/14/22

Provided by Tech Skills Transformations LLC



Prerequisites

- Modern system with at least 8G of memory and 20G free storage
- VirtualBox installed and running

<http://www.virtualbox.org>

- Virtual machine (.ova file) installed in VirtualBox

<https://www.dropbox.com/s/hdm7qumyjwm91ry/ArgoCD-GitOps-v2.ova?dl=0>

OR

<https://bclconf.s3.us-west-2.amazonaws.com/ArgoCD-GitOps-v2.ova>

- Workshop docs are in <https://github.com/skilldocs/gitops>

- Setup doc is at

<https://github.com/skilldocs/gitops/blob/main/gitops-setup.pdf>

- Labs doc for workshop

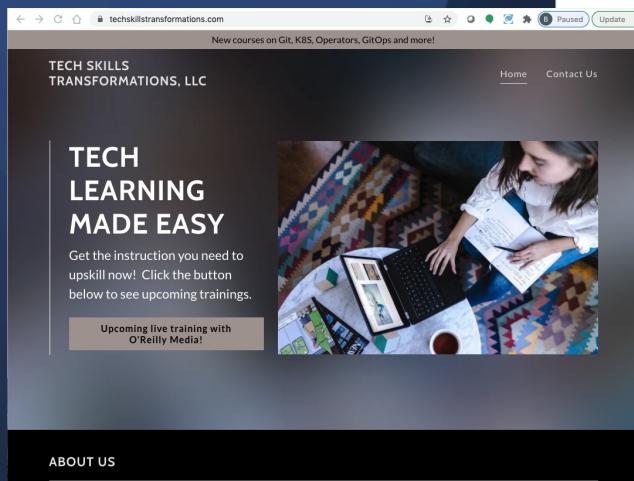
<https://github.com/skilldocs/gitops/blob/main/gitops-labs.pdf>



Continuous Delivery in Kubernetes with ArgoCD

Brent Laster

About me



- Founder, Tech Skills Transformations LLC
- R&D DevOps Director
- Global trainer – training (Git, Jenkins, Gradle, CI/CD, pipelines, Kubernetes, Helm, ArgoCD, operators)
- Author -
 - OpenSource.com
 - Professional Git book
 - Jenkins 2 – Up and Running book
 - Continuous Integration vs. Continuous Delivery vs. Continuous Deployment mini-book on Safari
- <https://www.linkedin.com/in/brentlaster>
- @BrentCLaster
- GitHub: brentlaster



Professional Git Book

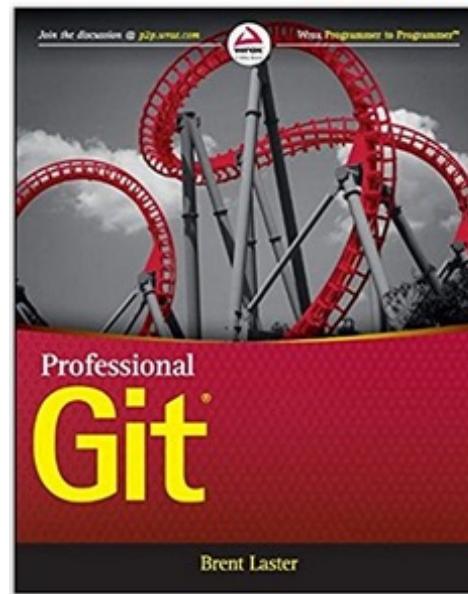
- Extensive Git reference, explanations,
- and examples
- First part for non-technical
- Beginner and advanced reference
- Hands-on labs

Professional Git 1st Edition

by Brent Laster ▾ (Author)

★★★★★ ▾ 7 customer reviews

[Look inside](#) ↴



© 2021 Brent C. Laster &

Tech Skills Transformations LLC



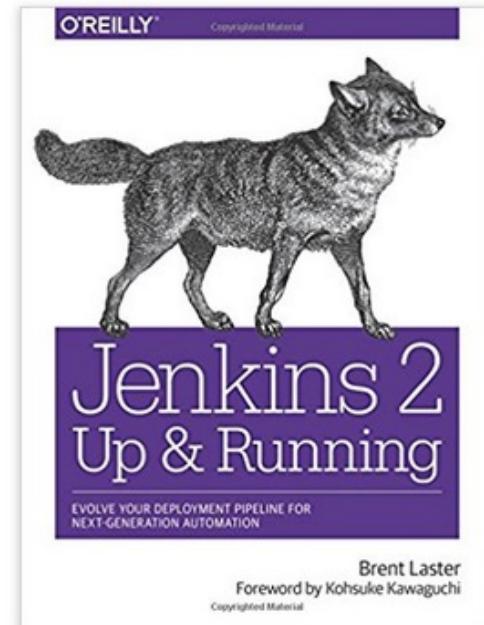
Jenkins 2 Book

- Jenkins 2 – Up and Running
- “It’s an ideal book for those who are new to CI/CD, as well as those who have been using Jenkins for many years. This book will help you discover and rediscover Jenkins.” *By Kohsuke Kawaguchi, Creator of Jenkins*

★★★★★ 5 customer reviews

#1 New Release in Java Programming

[Look inside](#) ↴





O'Reilly Training

April 26 & 27, 2021

Git Fundamentals

Join Brent Laster to gain the knowledge and skills you need to leverage Git to greatly simplify and speed up managing all of the changes in your source code. Once you ...

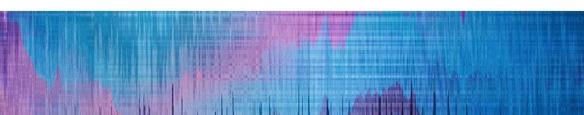
Next Level Git - Master your content

Use powerful tools in Git to simplify merges, rewrite history, and perform automatic updates

Topic: Software Development



BRENT LASTER



Building a deployment pipeline with Jenkins 2

Manage continuous integration and continuous delivery to release software

Topic: System Administration



BRENT LASTER



» LIVE ONLINE TRAINING

Containers A-Z

An overview of containers, Docker, Kubernetes, Istio, Helm, Kubernetes Operators, and GitOps

Topic: System Administration



BRENT LASTER



June 3, 10, 17 & 24, 2021

Git in 4 Weeks

Join author, trainer, and DevOps director Brent Laster to learn how Git works—and discover all the things you can do with it. Over four sessions, Brent walks you through everything you ...

» LIVE ONLINE TRAINING

Next Level Git - Master your workflow

Use Git to find problems, simplify working with multiple branches and repositories, and customize behavior with hooks

Topic: Software Development



BRENT LASTER



» LIVE ONLINE TRAINING

Git Troubleshooting

How to solve practically any problem that comes your way

Topic: Software Development



BRENT LASTER



» LIVE ONLINE TRAINING

Getting started with continuous delivery (CD)

Move beyond CI to build, manage, and deploy a working pipeline

Topic: System Administration



BRENT LASTER



» LIVE ONLINE TRAINING

Helm Fundamentals

Deploying, upgrading, and rolling back applications in Kubernetes

Topic: System Administration



BRENT LASTER



June 28, 2021

Troubleshooting Kubernetes

In this 3-hour course, global trainer, author, and DevOps director Brent Laster will show you how to respond to the most common problem situations you may encounter with Kubernetes. You'll learn ...



May 24, 2021

Continuous Delivery in Kubernetes with ArgoCD

Join expert Brent Laster to explore GitOps and learn how to use Argo CD to implement GitOps in your Kubernetes deployments. APAC time friendly - You're a Kubernetes admin who wants ...



May 17, 2021

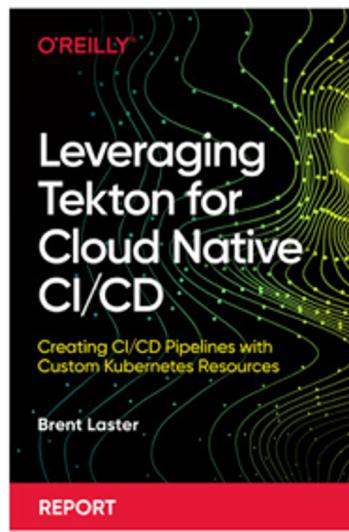
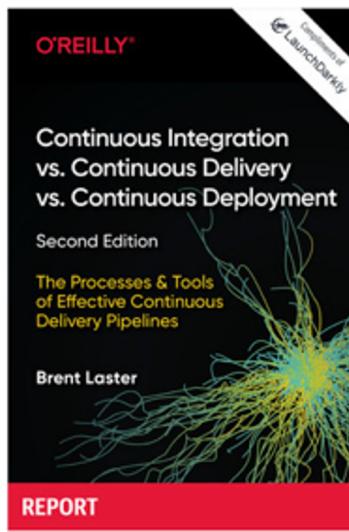
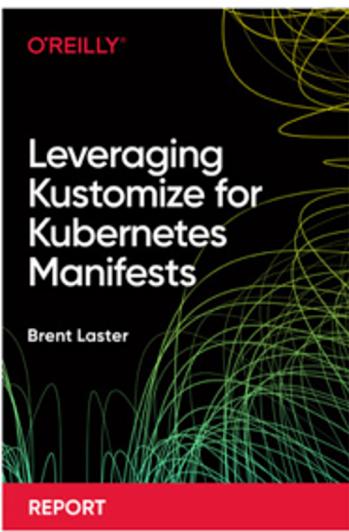
Building a Kubernetes Operator

Join expert Brent Laster to learn how the Operator pattern helps address these kinds of situations by allowing you to create custom controllers that extend the functionality of the Kubernetes API ...





O'Reilly Reports



Agenda

- Introduction to GitOps
- Introduction to Argo CD
- Argo CD Workflow and Usage
- Integrating GitOps with Argo CD and Helm
- Integrating GitOps into a CI/CD pipeline
- Overall entire workflow - from source change to cluster deployment



What is GitOps?

- An approach to Continuous Delivery for infrastructure and workloads
- Works by using Git as source of truth
- Per k8s, this means using pushes/pull requests instead of “kubectl create/apply” or “helm install/upgrade” when ready to add/change spec
- In traditional CI/CD pipeline, CD (per CI process) creates build artifacts and promotes them to production
- In GitOps pipeline model, ANY change to production must be made in source control prior to being applied to cluster
 - Should ideally be done through pull request/merge request
- Advantages - same as for traditional source control
 - Code review for changes
 - Tracking of who did what
 - Rollback is via Git
 - Whole infrastructure can be recreated from source control



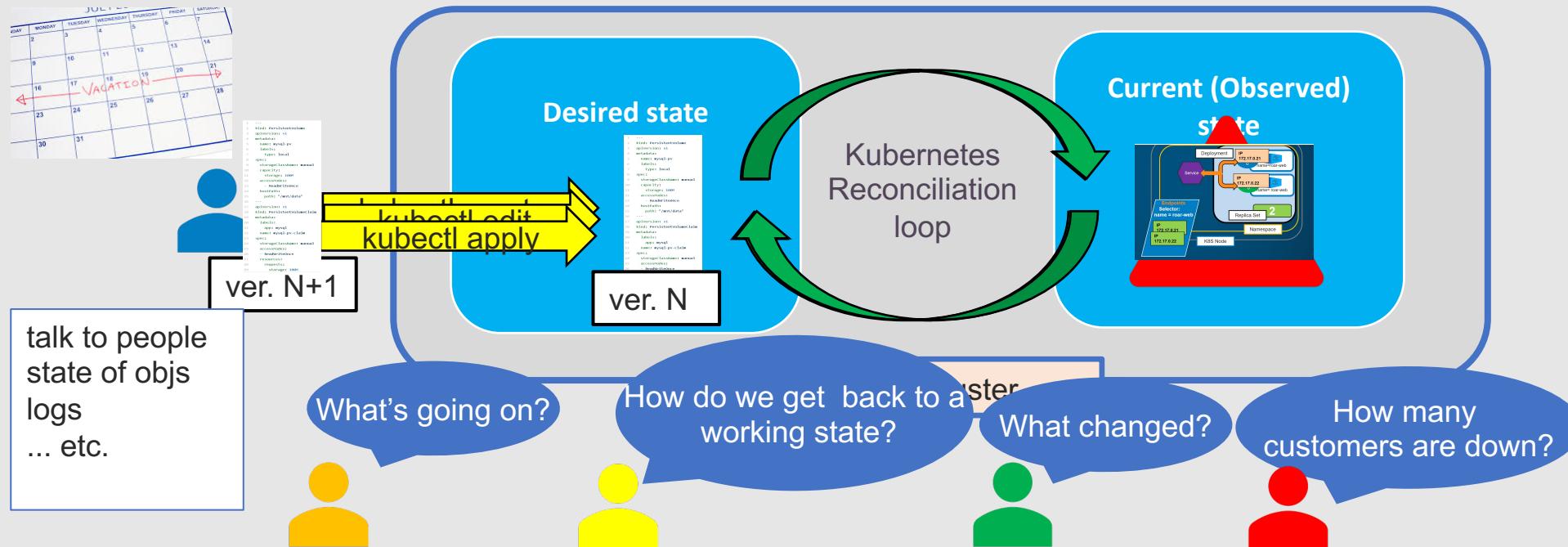
Advantages of GitOps

- Declarative - app definitions, environments, and configurations
- Version-controlled - trackable, declarative, reviewable, etc.
- Automated - no human interaction required after spec
- Repeatable - same inputs result in same target state
- Single source of the truth defining the application state
- Generally faster
- Less opportunity for error
- How does GitOps manage this?



K8s Management w/o GitOps

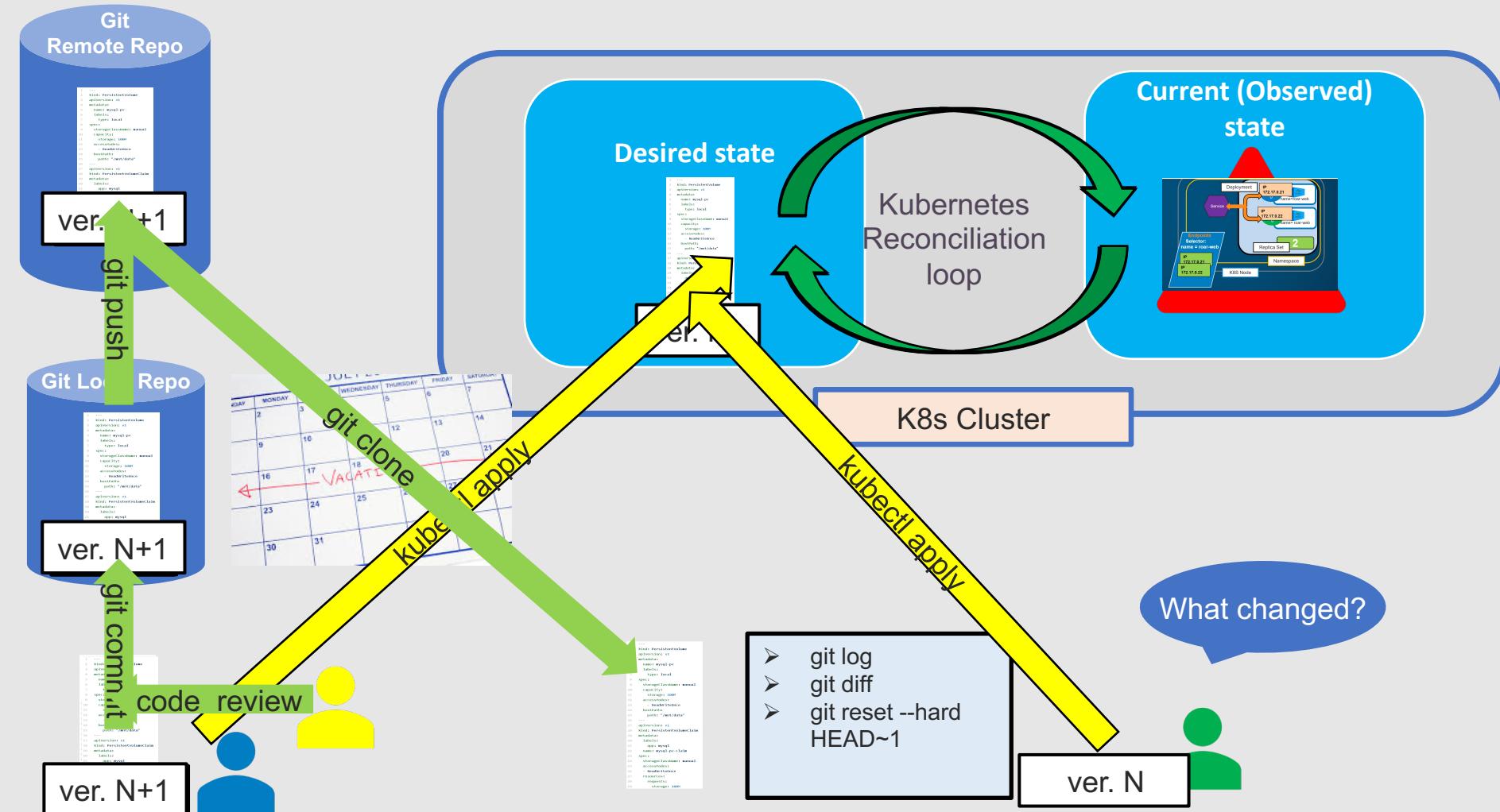
- User supplies desired state via declaring it in manifests and/or interactive commands
- Kubernetes works to balance the current state and the desired state
 - Desired state – what you want your production environment to be
 - Current (observed) state – current status of your production environment





Adding in Git

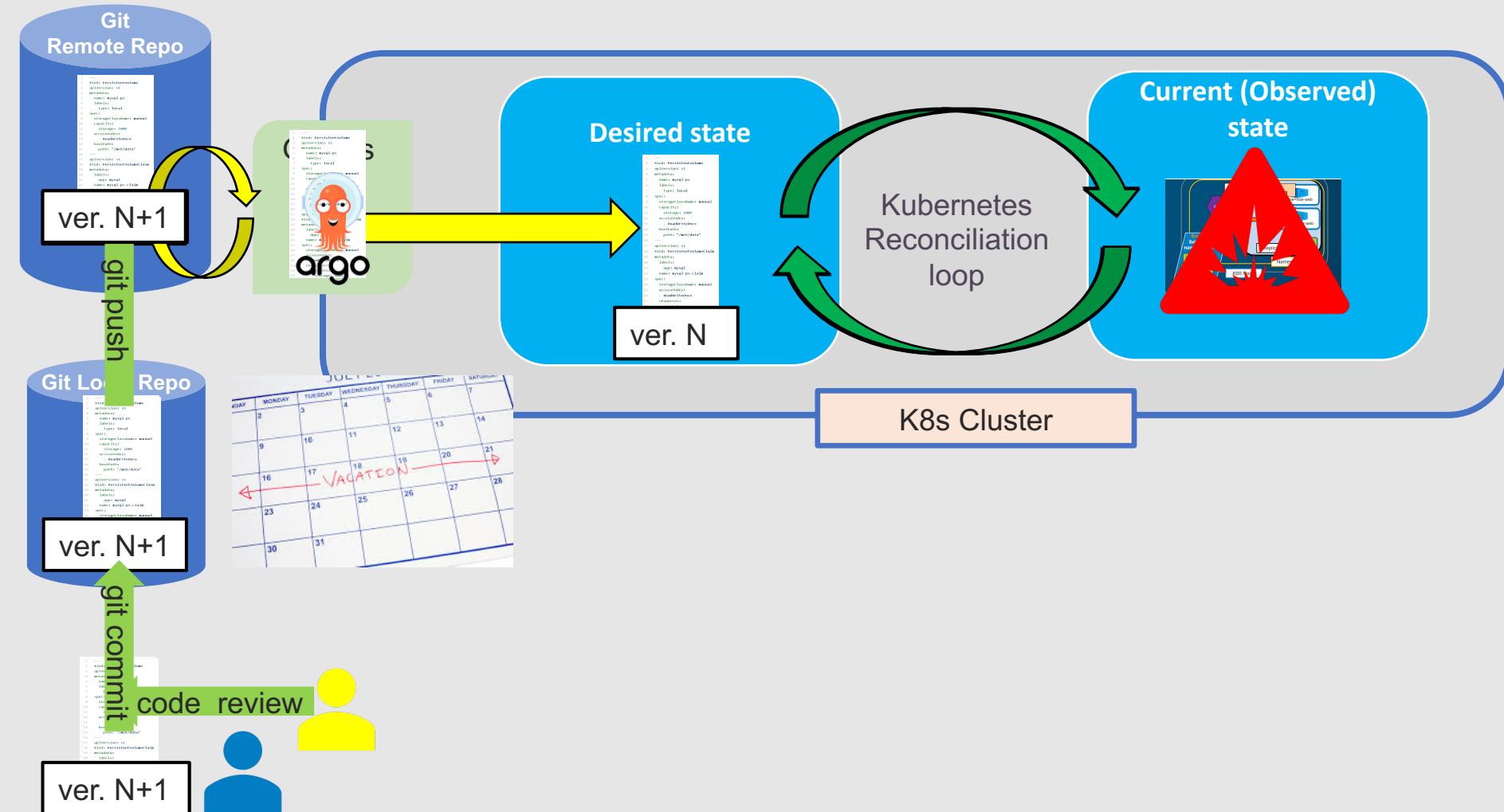
- Managing manifests (specs) in Git repositories





Automating Deployment with GitOps

- GitOps tools such as Argo CD automatically update the cluster from Git



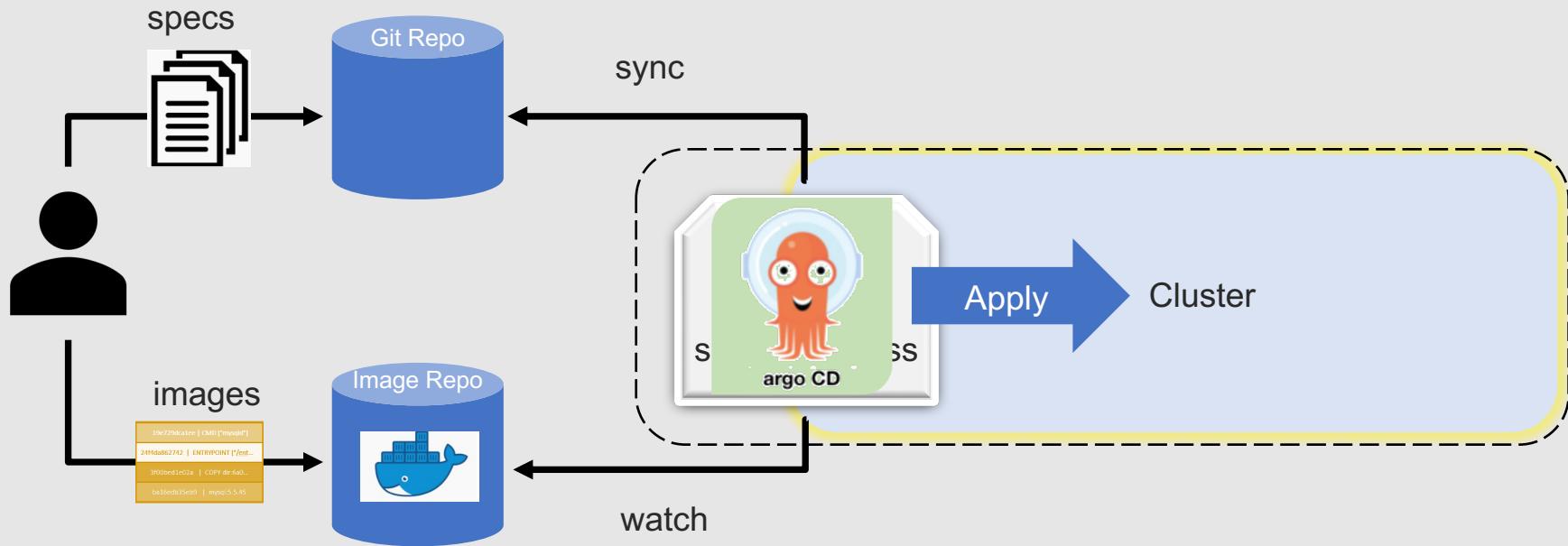


GitOps with Kubernetes – what do you need?

- a Git repository with your workloads definitions in YAML format, Helm charts and any other Kubernetes custom resource that defines your cluster desired state (*config* repository)
- a container registry where your CI system pushes immutable images (no *latest* tags, use *semantic versioning* or git *commit sha*)
- a controller, operator, or other process that runs in your cluster and does a two-way synchronization:
- watches the registry for new image releases
 - based on deployment policies
 - » updates the workload definitions with the new image tag
 - » commits the changes to the config repository
- watches for changes in the config repository and applies them to your cluster



GitOps Basic Model





What is ArgoCD?

- Declarative GitOps tool used to deploy applications to K8s
- Easy to configure, lightweight
- Intended only for K8s
- Designed with GitOps workflow built-in
- How it works...
 - Spins up its own controller in the cluster
 - Watches for changes in a repository
 - Compares against resources deployed in cluster
 - Synchronizes states



Key Concepts

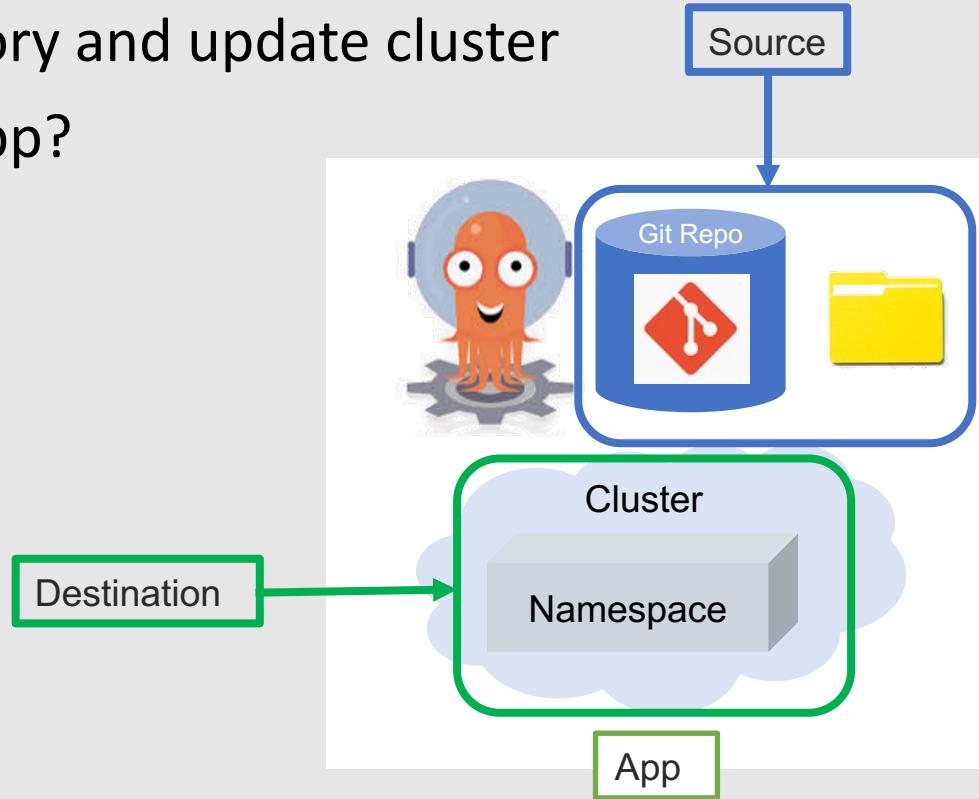
- **App/Application** - group of K8s resources defined by a manifest - CRD
- **Application source type** - which build tool is used to build the app
- **Live state** - current state of application in the cluster
- **Target state** - desired state of application - as stored in Git
- **Sync status** - is live the same as target?
- **Sync** - updating app to the target state
- **Sync operation status** - success/failure of sync
- **Refresh** - comparison of code in Git with live state
- **Health** - is app running correctly? Serving requests?
- **Tool/Configuration management tool** - tool to create manifests from files in directory (Kustomize or Ksonnet for example)
- **Configuration management plugin** - custom tool

From: https://argoproj.github.io/argo-cd/core_concepts/



Argo CD Apps

- aka Application – K8s CRD – a group of resources defined by a manifest
- Used to monitor repository and update cluster
- What's specified in an App?
 - Argo CD project
 - source repo
 - revision
 - path
 - cluster
 - namespace





Adding Argo CD Apps

- Can create via:
 - command line
 - web interface
 - yaml in web
 - k8s manifest (CRD)
- Can be specified in several ways:
 - kustomize
 - helm
 - ksonnet
 - jsonnet
 - plain yaml collection
 - custom config mgmt. tool setup as config mgmt. plugin
- argocd app create - pull in files from repo and deploy on K8s
- argocd list - list apps and see status and configuration

```
$ argocd app create roar-deploy-k8s --repo https://github.com/brentlaster/roar-deploy-k8s --path . --dest-server $CLUSTER_IP --dest-namespace roar
```

The screenshot shows the Argo CD web interface. At the top left, there's a sidebar with a logo and the text 'v1.8.3+'. In the center, there's a list of applications with one item labeled 'Applications' and a red circle with the number '2'. Below this is a button labeled '+ NEW APP' with a red circle containing the number '1'. To the right, there's a large modal window titled 'CREATE' for a new application. The 'GENERAL' tab is selected, showing the 'Application Name' field set to 'roar-deploy-k8s' and the 'Project' field set to 'default'. Under 'SYNC POLICY', it says 'Manual'. Under 'SYNC OPTIONS', there's a checked checkbox for 'USE A SCHEMA TO VALIDATE RESOURCE MANIFESTS'. On the right side of the modal, there's a 'EDIT AS YAML' button, which is circled in red. At the bottom of the modal, there are 'SAVE' and 'CANCEL' buttons. Below the modal, there's a code editor window showing the YAML manifest for the application:

```
1  apiVersion: argoproj.io/v1alpha1
2  kind: Application
3  metadata:
4    name: roar-deploy-k8s
5  spec:
6    destination:
7      name: ''
8      namespace: ''
9      server: ''
10   source:
11     path: ''
12     repoURL: 'https://github.com/brentlaster/roar-deploy-k8s'
13     targetRevision: HEAD
14   project: default
15
```



Our Example App

- R.O.A.R. (Registry of Animal Responders)
- Simple app with two pieces
 - Webapp on the front
 - Mysql database on the backend
- Written in Java
- Simple REST API
- War file is produced for webapp
- Managed in Tomcat

Id	Name	Species	Date of First Service	Date of Last Service	Adversary	Adversary Tech
1	Road Runner	bird	1955-01-20	1995-02-15	Wile E. Coyote	ACME product du jour
2	Scooby	dog	1969-05-19	2000-02-11	fake ghosts	mask
3	Perry	platypus	2013-01-20	2015-04-09	H. Doofensmirtz	...inator
4	Mr. Krabs	crab	2010-06-17	2014-07-07	Plankton	various
5	Bugs Bunny	rabbit	1966-05-22	1988-04-15	E. Fudd	wabbit gun



Argo CD Command Line

Usage:

`argocd [flags]`

`argocd [command]`

Available Commands:

account Manage account settings

app Manage applications

cert Manage repository certificates and SSH known hosts entries

cluster Manage cluster credentials

completion output shell completion code for the specified shell (bash or zsh)

context Switch between contexts

gpg Manage GPG keys used for signature verification

help Help about any command

login Log in to Argo CD

logout Log out from Argo CD

proj Manage projects

relogin Refresh an expired authenticate token

repo Manage repository connection parameters

repocreds Manage repository connection parameters

version Print version information

Use `argocd [command] --help`" for more information about a command.

Flags:

--auth-token string Authentication token

--client-crt string Client certificate file

--client-crt-key string Client certificate key file

--config string Path to Argo CD config (default "/home/diyuser3/.argocd/config")

--grpc-web Enables gRPC-web protocol. Useful if Argo CD server is behind proxy which does not support HTTP2.

--grpc-web-root-path string Enables gRPC-web protocol. Useful if Argo CD server is behind proxy which does not support HTTP2. Set web root.

-H, --header strings Sets additional header to all requests made by Argo CD CLI. (Can be repeated multiple times to add multiple headers, also supports comma separated headers)

-h, --help help for argocd

--insecure Skip server certificate and domain verification

--logformat string Set the logging format. One of: text|json (default "text")

--loglevel string Set the logging level. One of: debug|info|warn|error (default "info")

--plaintext Disable TLS

--port-forward Connect to a random argocd-server port using port forwarding

--port-forward-namespace string Namespace name which should be used for port forwarding

--server string Argo CD server address

--server-crt string Server certificate file

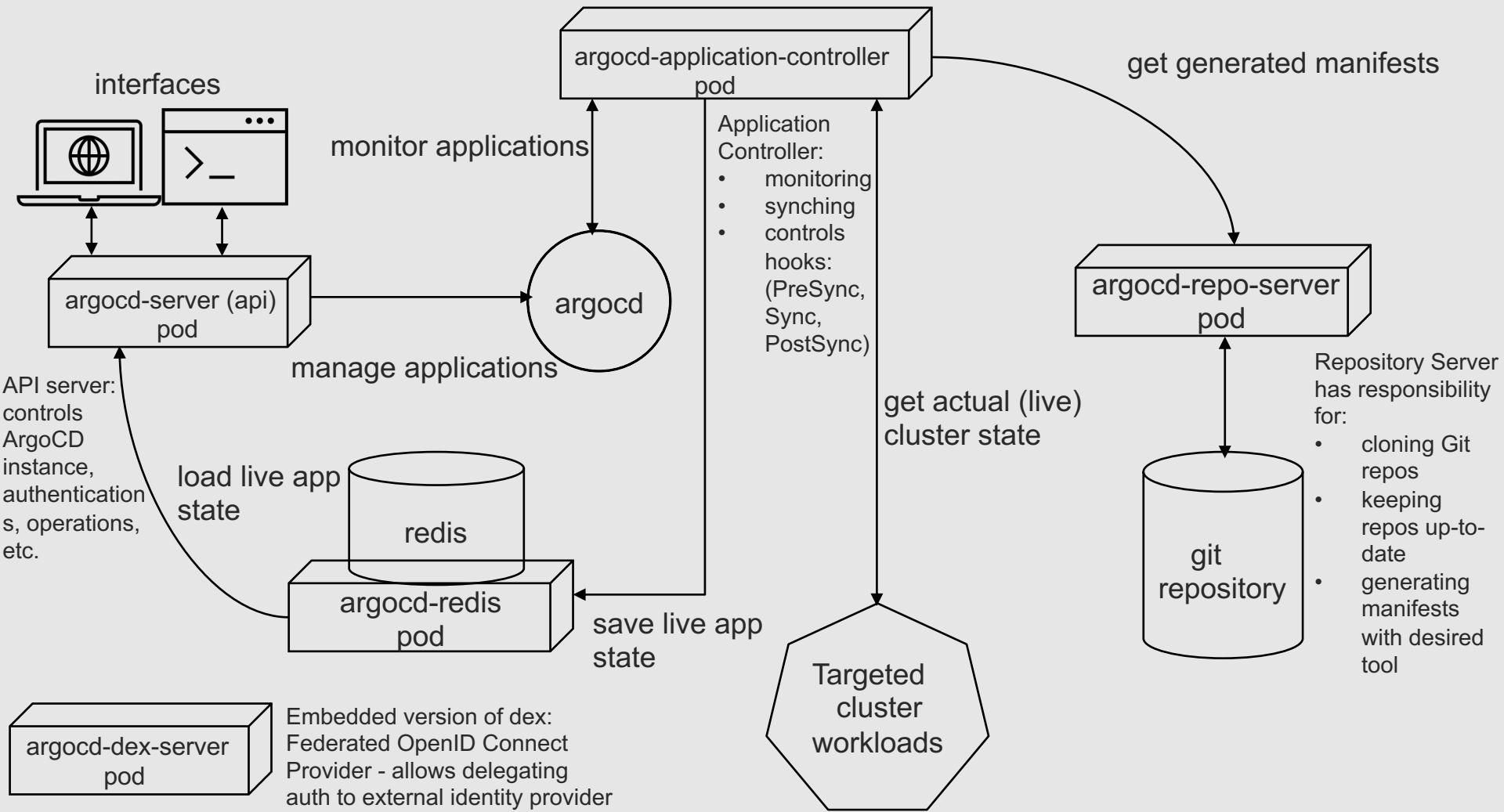


Lab 1 - Getting Started with Argo CD



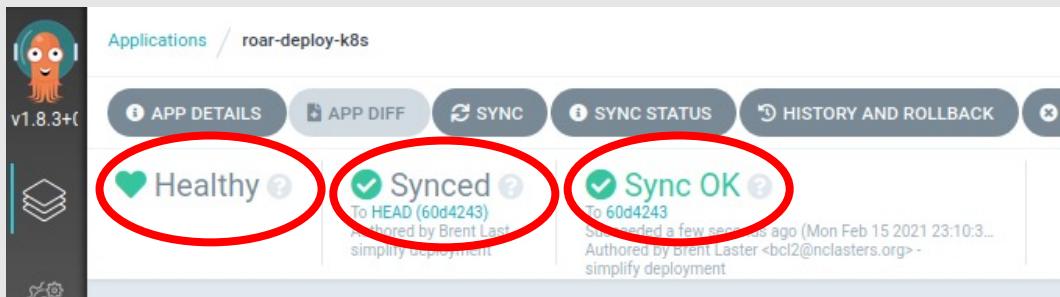
ArgoCD Components and Flow

- Rectangular boxes represent argocd pods running in argocd namespace





ArgoCD Health and Sync Statuses



This screenshot shows the ArgoCD UI for the application 'roar-deploy-k8s'. The top navigation bar includes 'APPLICATIONS' and the specific app name. Below the navigation are several tabs: 'APP DETAILS', 'APP DIFF', 'SYNC', 'SYNC STATUS', and 'HISTORY AND ROLLBACK'. The 'SYNC STATUS' tab is active. Three status indicators are displayed: 'Healthy' (green heart icon), 'Synced' (green checkmark icon), and 'Sync OK' (green checkmark icon). Each status has a tooltip providing more details. The 'Sync OK' status also includes a timestamp and author information.



This screenshot shows the ArgoCD UI for the same application 'roar-deploy-k8s'. The 'SYNC STATUS' tab is active. Three status indicators are displayed: 'Missing' (ghost icon), 'OutOfSync' (yellow exclamation mark icon), and 'Sync failed' (red X icon). The 'Sync failed' status has a tooltip with a timestamp and author information.

- Built-in Resource Health Checks
 - Deployment, ReplicaSet, StatefulSet, DaemonSet
 - Observed = Desired
 - # of updated replicas = # of desired replicas
 - Service
 - If type=LoadBalancer, ingress list is non-empty and has 1 or more values for hostname or IP
 - Ingress
 - ingress list is non-empty and has a hostname or IP
 - PersistentVolumeClaim
 - status.phase = Bound
- Custom Health Checks
 - can be added - written in LUA - added to argocd-cm or bundled

- **Health Status**

- health status of app running in cluster
 - » **Progressing** – waiting for rollout to be completed and observable
 - » **Healthy** – health checks indicate items are present and working as expected
 - » **Degraded** – the rollout is invalid
 - » **Suspended** – the rollout is paused
 - » **Missing** – the rollout hasn't occurred

- **Sync Status**

- whether or not the version of your app is up to date with your cluster deployment

- **Last Sync Status**

- whether or not your last app sync was successful (to the cluster)
- not necessarily the same as sync status
- example: something may have disappeared in cluster – sync status = out of sync, but this is still OK



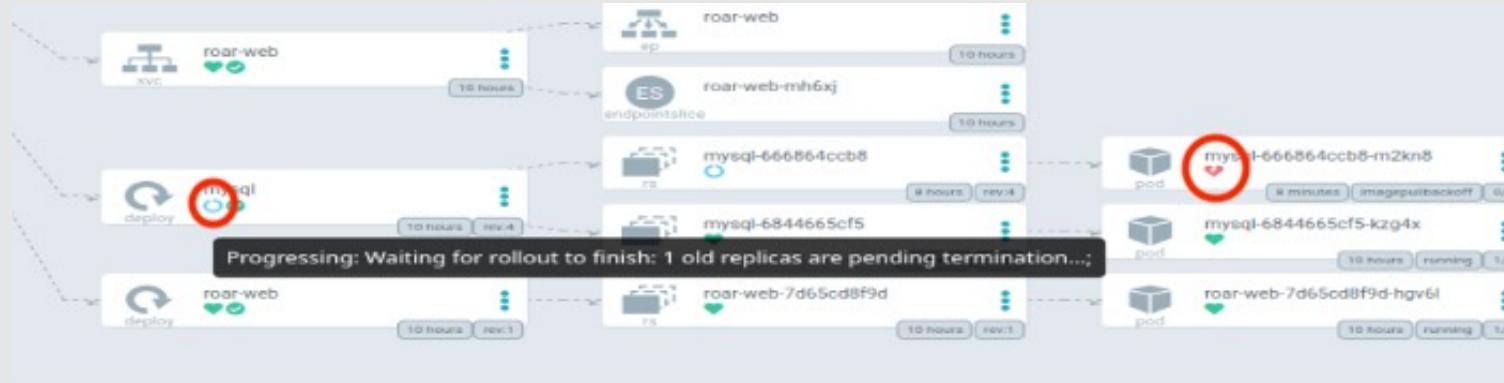
Manual Sync

The screenshot shows the Rancher UI interface for managing Kubernetes applications. On the left, a sidebar has icons for Applications, Catalog, Projects, and Settings. The main area shows an application named "roar-deploy-k8s". A navigation bar at the top includes "APP DETAILS", "APP DIFF", "SYNC" (circled with a red box labeled 1), "SYNC STATUS", "HISTORY AND ROLLBACK", and "DELETE". Below this, a card indicates the application is "OutOfSync" from the repository "https://github.com/brentlaster/roar-deploy-k8s.git". A "Sync OK" status is shown with a green checkmark, indicating a successful sync 22 minutes ago. A diagram shows a "mysql-configmap" connected to a "mysql-pv". To the right, a modal window titled "SYNCHRONIZE" (circled with a red box labeled 2) is open, showing the command "Synchronizing application manifests from https://github.com/brentlaster/roar-deploy-k8s.git". It lists the revision "future" and provides options for "PRUNE", "DRY RUN", "APPLY ONLY", and "FORCE". Under "SYNCHRONIZE RESOURCES", checkboxes are checked for "/CONFIGMAP/ROAR/MYSQL-CONFIGMAP" and "/PERSISTENTVOLUME//MYSQL-PV". At the bottom of the modal, it says "all / out of sync / none".

- **Prune** - allow deleting resources that are unexpected / no longer tracked in git
- **Dry Run** - preview what an apply operation would do without affecting the cluster
 - performs a 'kubectl apply --dry-run' w/o actually doing the sync
- **Apply Only** - skip pre/post sync hooks
- **Force** – use a force apply ('kubectl apply --force')
 - --force flag deletes and re-creates resource(s) when patch encounters conflict after 5 retries



Sync: Understanding what's happening



- In “Progressing” or “Degraded” state
- Items out of sync indicated by broken heart
- Click on item to see Summary, Events, Logs

SUMMARY	EVENTS	LOGS		
REASON	MESSAGE	COUNT	FIRST OCCURRED	LAST OCCURRED
Scheduled	Successfully assigned roar/mysql-666864ccb8-m2kn8 to training1			
Pulling	Pulling image 'quay.io/bclaster/roar-db:1.2.3'	4	2021-01-20T12:26:55Z	2021-01-20T12:28:22Z
Failed	Failed to pull image "quay.io/bclaster/roar-db:1.2.3": rpc error: code = Unknown desc = Error response from daemon: manifest for quay.io/bclaster /roar-db:1.2.3 not found	4	2021-01-20T12:26:55Z	2021-01-20T12:28:23Z
Failed	Error: ErrImagePull	4	2021-01-20T12:26:55Z	2021-01-20T12:28:23Z
BackOff	Back-off pulling image "quay.io/bclaster/roar-db:1.2.3"	6	2021-01-20T12:26:56Z	2021-01-20T12:28:09Z
Failed	Error: ImagePullBackOff	45	2021-01-20T12:26:56Z	2021-01-20T12:37:06Z



Comparing Live vs Desired State

- Select desired item
- Can look at Live Manifest from cluster
- Can look at Desired Manifest from Git
- Can look at Diff of the 2



LIVE MANIFEST	DIFF	DESIRED MANIFEST
196 197 198 199		configMapKeyRef: key: mysql.user name: mysql-configmap image: 'quay.io/bclaster/roar-db:1.0.2'

LIVE MANIFEST	DIFF	DESIRED MANIFEST	Compact diff	Inline Diff
186 186		key: mysql.user	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
187 187		name: mysql-configmap		
188 188		image: 'quay.io/bclaster/roar-db:1.0.2'		
		image: 'quay.io/bclaster/roar-db:1.2.3'		
189 189		imagePullPolicy: IfNotPresent		
190 190		name: mysql		



History and Rollback

The screenshot shows the Argo CD web interface. At the top, there are tabs: STATUS, HISTORY AND ROLLBACK (which is selected), and DEPLOYMENTS. Below the tabs, a green "Sync OK" status message is displayed. The main area lists three previous deployments:

- Deployment 1:** Revision d715e64, Authored by brenlaster <jx.demo.user@gmail.com>, 10 minutes ago (Wed Jan 20 2021 08:29:29 GMT-0500). It includes sections for GPG signature, DIRECTORY, DIRECTORY RECURSE (false), TOP-LEVEL, ARGUMENTS, and EXTERNAL VARIABLES.
- Deployment 2:** Revision d6acaf9, Authored by brenlaster <jx.demo.user@gmail.com>, an hour ago (Wed Jan 20 2021 07:26:54 GMT-0500). It includes sections for GPG signature, DIRECTORY, DIRECTORY RECURSE (false), TOP-LEVEL, ARGUMENTS, and EXTERNAL VARIABLES.
- Deployment 3:** Revision 6938b59, Authored by brenlaster <jx.demo.user@gmail.com>, 12 hours ago (Wed Jan 20 2021 06:51:59 GMT-0500). It includes sections for GPG signature, DIRECTORY, DIRECTORY RECURSE (false), TOP-LEVEL, ARGUMENTS, and EXTERNAL VARIABLES.

In the bottom right corner of the third deployment card, there is a red oval highlighting a "Rollback" button.

- Argo CD keeps track of the various versions deployed
- Allows you to go back to a previous state
- Invoke via “HISTORY AND ROLLBACK”
- Select deployment, 3 dots, and “Rollback”

Lab 2 - Dealing with Failed Deployments and Rollback



Automated Sync

- ArgoCD creates manual triggers by default
- Has the ability to automatically sync application
 - sync happens when differences are detected
 - » between intended manifests in Git and live state in the cluster
 - allows CI/CD pipelines to not need direct access to API server to do deployment
 - pipeline just does commit and pushes to Git repo with manifest changes
 - To turn on automated sync
 - » command line: `argocd app set <APPNAME> --sync-policy automated`
 - » application manifest:

```
spec:  
  syncPolicy:  
    automated: {}
```



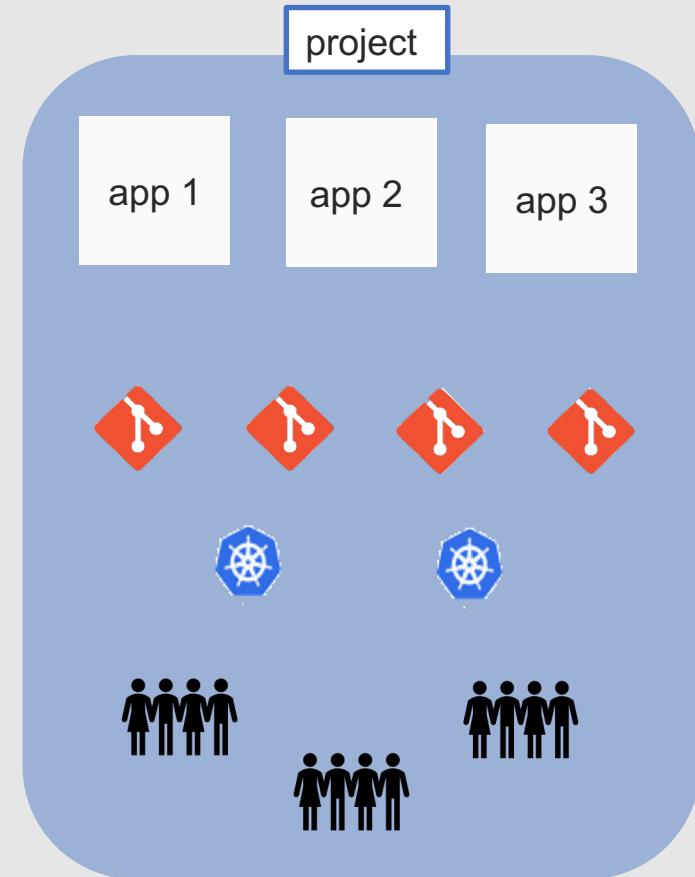
Automated Sync Behavior

- Only occurs if application is “OutOfSync”
- Only attempts one sync per unique combination of commit SHA1 and parameters of the app
 - If most recent sync was successful and was against the same commit SHA1 and app parameters, won’t do 2nd sync - unless “selfHeal” flag is true
 - If selfHeal flag is true, sync will be attempted again after self heal timeout
 - » 5 second default
 - » can be set via --self-heal-timeout-seconds flag in argocd-application-controller deployment
- Not attempted if previous sync against same SHA1 and parameters failed
- Rollback can’t be used on app if automatic sync enabled



ArgoCD Projects

- Functions
 - Logical group of applications
 - Helpful for organization by teams
 - » isolates teams from each other
 - Defines resources available to apps
- Features
 - Defines restrictions (limited set to choose from)
 - » which repos can be used for applications source
 - » where things can be deployed to (clusters, namespaces)
 - » kinds that can be deployed (CRDs, RBAC, NetworkPolicy, etc.)
 - » which teams of users have access to apps in project
 - Roles
 - » provide app RBAC
- Each app belongs to one project
- Default project is “default”
- Others can be added





ArgoCD Projects Creation

- create via:
 - argocd proj create -f <filename>
 - k apply -f <filename> -n argocd
- Can also create via manifest
 - apiVersion = argoproj.io/v1alpha1
 - kind = AppProject
- Add Source
 - git repository
 - argocd proj add-source
 - must be added to argocd first
 - » via argocd repo add
- Add Destination
 - cluster
 - argocd proj add-destination
 - must be added to argocd first
 - » via argocd cluster add
- Add certificates for auth
 - argocd cert list
 - argocd cert add-<type>

```
1 |apiVersion: argoproj.io/v1alpha1
2 |kind: AppProject
3 |metadata:
4 |  name: helproj
5 |  namespace: argocd
6 |spec:
7 |  destinations:
8 |    - namespace: helm-demo
9 |      server: https://10.0.2.15:8443
10 |  sourceRepos:
11 |    - git@10.0.2.15:/git/repos/roar-k8s-helm.git
12 |
```



App Types

```
$ argocd app create -h
```

```
Create an application
```

```
Usage:
```

```
  argocd app create APPNAME [flags]
```

Examples:

```
# Create a directory app
```

```
argocd app create guestbook --repo https://github.com/argoproj/argocd-example-apps.git --path  
guestbook --dest-namespace default --dest-server https://kubernetes.default.svc --directory-recurse
```

```
# Create a Jsonnet app
```

```
argocd app create jsonnet-guestbook --repo https://github.com/argoproj/argocd-example-apps.git --path  
jsonnet-guestbook --dest-namespace default --dest-server https://kubernetes.default.svc --jsonnet-ext-str  
replicas=2
```

```
# Create a Helm app
```

```
argocd app create helm-guestbook --repo https://github.com/argoproj/argocd-example-apps.git --path  
helm-guestbook --dest-namespace default --dest-server https://kubernetes.default.svc --helm-set replicaCount=2
```

```
# Create a Helm app from a Helm repo
```

```
argocd app create nginx-ingress --repo https://kubernetes-charts.storage.googleapis.com --helm-chart  
nginx-ingress --revision 1.24.3 --dest-namespace default --dest-server https://kubernetes.default.svc
```

```
# Create a Kustomize app
```

```
argocd app create kustomize-guestbook --repo https://github.com/argoproj/argocd-example-apps.git --  
path kustomize-guestbook --dest-namespace default --dest-server https://kubernetes.default.svc --kustomize-image  
gcr.io/heptio-images/ks-guestbook-demo:0.1
```

```
# Create a app using a custom tool:
```

```
argocd app create ksane --repo https://github.com/argoproj/argocd-example-apps.git --path  
plugins/kasane --dest-namespace default --dest-server https://kubernetes.default.svc --config-management-plugin  
kasane
```



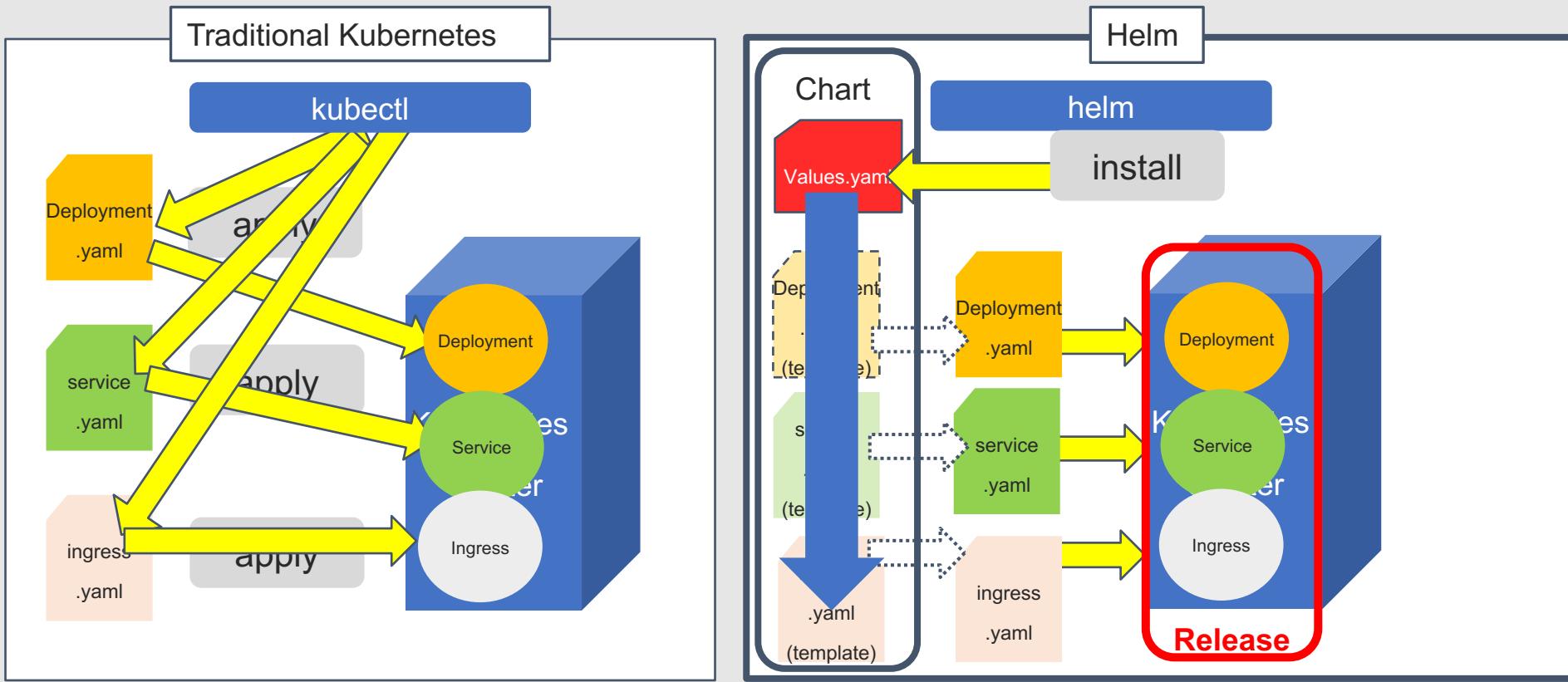
What is Helm?

- Package Manager and Lifecycle Manager for K8s
 - Like yum, apt but for K8s
 - Bundles related manifests (such as deployment.yaml, service.yaml, etc.) into a “chart”
 - When installing chart, Helm creates a “release”
 - Lifecycle management
 - Create, Install, Upgrade, Rollback, Delete, Status, Versioning
 - Benefits
 - Templating, Repeatability, Reliability, Multiple Environment, Ease of collaboration



How does Helm simplify things?

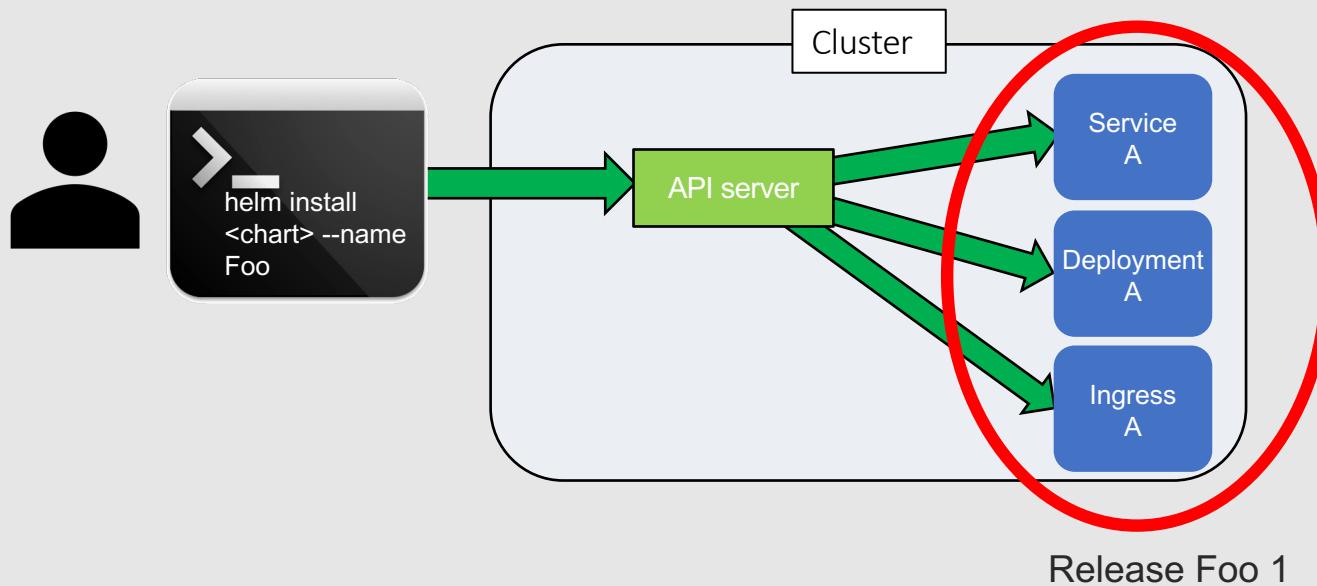
- Traditional deployment in Kubernetes is done with kubectl across files into separately managed items
- Helm deploys units called charts as managed releases





Helm Install a Chart

- helm install <chart>





How Values are resolved in Helm

templates/deployment.yaml

```
spec:  
  containers:  
    - name: {{ .Chart.Name }}  
      image: bclaster/roar-db-image:v1  
      imagePullPolicy: Always  
    ports:  
      - name: {{ .Values.deployment.ports.name }}  
        containerPort: {{ .Values.deployment.ports.containerPort }}  
    env:  
      {{- include "roar-db.environment-values" . | indent 10 }}  
      
```

values.yaml

```
# Default values for roar-db chart.  
# This is a YAML-formatted file.  
# Declare variables to be passed  
# into your templates.  
replicaCount: 1  
nameOverride: mysql  
deployment:  
  ports:  
    name: mysql  
    containerPort: 3306  
    
```

deployment.yaml

```
...  
  containerPort: 3306  
  
```

- string indicates hierarchical path
- dot notation separates levels
- .Values refers to top-level values from values.yaml



Traditional K8s yaml vs Helm template

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: roar-web
  labels:
    app: roar-web
  namespace: roar
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: roar-web
    spec:
      containers:
        - name: roar-web
          image: localhost:5000/roar-web-v1
        ports:
          - name: web
            containerPort: 8080
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ template "roar-web.name" . }}
  labels:
    app: {{ template "roar-web.name" . }}
  chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
  release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-web.name" . }}
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          {{- if .Values.image.pullPolicy }}
            imagePullPolicy: {{ toYaml .Values.image.pullPolicy }}
          {{- end }}
          ports:
            - name: {{ .Values.deployment.ports.name }}
              containerPort: {{ .Values.deployment.ports.containerPort }}
```



Adding a Helm project in Argo CD

CREATE CANCEL

Application Name
helm-demo

Project
helmpoj

SYNC POLICY
Automatic

PRUNE RESOURCES (optional)
 SELF HEAL (optional)

SYNC OPTIONS
 USE A SCHEMA TO VALIDATE RESOURCE MANIFESTS
 AUTO-CREATE NAMESPACE

SOURCE

Repository URL
git@10.0.2.15:/git/repos/roar-k8s.git

Revision

HEAD

Path
helm

DESTINATION

Cluster URL
https://10.0.2.15:8443 URL ▾

Namespace
helm-demo

HELM

VALUES FILES

VALUES

```
values.yaml
# Default values for roar charts.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates
roar-db:
  image:
    repository: quay.io/bclaster/roar-db
    tag: 1.0.2
    pullPolicy: Always
  resources:
    limits:
      memory: 128Mi
      cpu: 100m
    requests:
      memory: 64Mi
      cpu: 50m
  env:
    - name: ROAR_DB_HOST
      value: "10.0.2.15"
    - name: ROAR_DB_PORT
      value: "8080"
    - name: ROAR_DB_NAME
      value: "roar-db"
    - name: ROAR_DB_USER
      value: "roar"
    - name: ROAR_DB_PASSWORD
      value: "roar"
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  selector:
    matchLabels:
      app.kubernetes.io/name: roar-db
      app.kubernetes.io/instance: roar
  serviceAccountName: roar-db
  tolerations:
    - key: "node-role.kubernetes.io/master"
      operator: Exists
      effect: NoSchedule
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: "node-role.kubernetes.io/master"
                operator: In
                values:
                  - "true"
```

PARAMETERS

roar-db.image.pullPolicy	Always
roar-db.image.repository	quay.io/bclaster/roar-db
roar-db.image.tag	1.0.2
roar-web.image.pullPolicy	Always
roar-web.image.repository	quay.io/bclaster/roar-web
roar-web.image.tag	1.10.1

Lab3 - Helm Projects and Automated Sync



Tools to Create Manifest

- Different tools can be used to create K8s manifests in ArgoCD
 - Helm charts
 - Kustomize applications
 - Ksonnet applications
 - Directory of YAML/JSON/Jsonnet manifests
 - A custom config management tool configured as a plugin*
 - plugin configuration
 - add required binaries in argocd-repo-server pod
 - can be via volume mounts or custom image
 - register plugin in argocd-cm ConfigMap
- For development only – upload local manifests
 - example: \$argocd app sync AppName --local /path/to/directory
 - requires user with override permission
- Useful in constructing Continuous Delivery (CD) pipelines



Continuous Delivery Practice

- Theme is automation of software production process
- Combines 3 core practices/disciplines
 - Continuous Integration
 - Continuous Testing
 - Continuous Deployment (if desired)
- Includes Configuration Management
- May also include other areas, such as metrics.



Carl Caum published on 30 August 2013

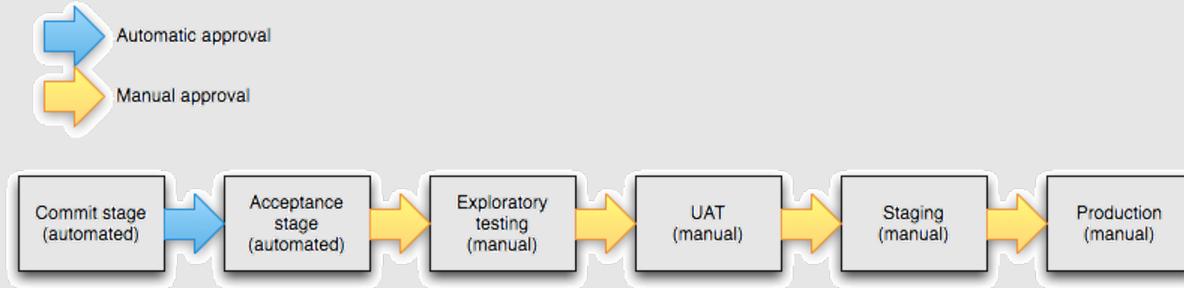
Continuous Delivery doesn't mean every change is deployed to production ASAP. It means every change is proven to be deployable at any time

— Carl Caum (@ccaum) August 28, 2013



Deployment Pipeline

- “... an automated implementation of your application’s build, deploy, test, and release process.



- Every change made to configuration, source code, environment or data triggers a new instance of the pipeline.
- Change motivates production of binaries and then a series of tests are done to prove it is releasable.
- Levels of testing provide successive levels of confidence.
- Typically implemented using CI/CD or orchestration tooling such as Jenkins.



Jenkins – What is it?

- Open-source framework for CI, CD, creating pipelines
- Monitors execution of repeated jobs, provides notifications, records results, distributes workloads across systems
- Generally used for:
 - **Creating Continuous Delivery Pipelines**
 - **Building/testing software projects continuously**
 - **Monitoring executions of externally-run jobs, such as cron jobs**
- Created by Kohsuke Kawaguchi
- Over 1000 plug-ins
- Maintained by Jenkins Community
- Enterprise support by CloudBees



The Jenkins “Object Model”

Jenkins Dashboard (Latest Build Jobs Statuses)

The screenshot displays the Jenkins interface with two main windows open:

- Build History:** This window shows a timeline of build jobs for the project "Userjob2". The history includes builds from Sep 20, 2015, at 11:12 AM to Sep 19, 2015, at 12:25 PM. Each entry is represented by a small icon and a timestamp.
- Build Job:** This window provides detailed information for the most recent build, "Build #7 (Sep 20, 2015 8:09:18 AM)". It shows the status as "Failed" (indicated by a red circle), the duration as "Started 4 hr 8 min ago Took 70 ms on master", and the fact that "No changes" were detected. It also notes that the build was "Started by user sadicl".

Both windows include standard Jenkins navigation links like "Back to Project", "Status", "Changes", "Console Output", and "Delete Build". The bottom of the screen shows the Jenkins footer with links for "Help us localize this page", "Page generated: Sep 20, 2015 9:25:15 AM REST API Jenkins ver. 1.622", and "RSS for all RSS for failures".



What is Jenkins 2 (2.0+)?

- Features

- Next evolution of Jenkins
- Includes more integrated support for pipelines-as-code
- Provides DSL - pipeline “programming language”
- Support for pipeline scripts stored in source control - Jenkinsfiles
- Automatic project creation based on Jenkinsfile presence in branches
- Improved DSL structure and processing via Declarative Pipelines
- Advanced interface - Blue Ocean

- Motivations

- Treat pipelines as a first class citizen
- Build support around them as a construct
- Allow to express in coding
- Use programming logic
- Treat like code
 - » Store in SCM
 - » Reviewable
- Easy to test
- Text-based
- Handle exceptional cases
- Restarts



The Jenkinsfile

- Pipeline script stored in SCM
- Can develop in the job and then transfer to Jenkinsfile
- Granularity is per branch, per project
- Used as marker for Jenkins to identify branches (including creation and deletion) in multibranch and organization projects

projects / gradle-greetings.git / blob

[summary](#) | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#)
[history](#) | [raw](#) | [HEAD](#)

add new test files

[gradle-greetings.git] / Jenkinsfile

```
1 #!groovy
2 import static org.foo.Utilities.*
3 node ('worker_node1') {
4     // always run with a new workspace
5     step([$class: 'WsCleanup'])
6     try {
7         stage('Source') {
8             checkout scm
9             stash name: 'test-sources', includes: 'build.gradle,src/test/'
10    }
11    stage('Build') {
12        // Run the gradle build
13        gbuild this, 'clean build -x test'
14    }
15    stage ('Test') {
16        // execute required unit tests in parallel
17
18        parallel (
19            .master: { node ('master'){
20                // always run with a new workspace
21                step([$class: 'WsCleanup'])
22                unstash 'test-sources'
23                gbuild this, '-D test.single=TestExample1 test'
24            },
25            worker2: { node ('worker_node2'){
26                // always run with a new workspace
27                step([$class: 'WsCleanup'])
28                unstash 'test-sources'
29                gbuild this, '-D test.single=TestExample2 test'
30            },
31        }
32    }
33 }
34 catch (err) {
35     echo "Caught: ${err}"
36 }
37 stage ('Notify') {
38     // mailUser('<your email address>', "Finished")
39 }
40
41 }
42 }
```



Multibranch Pipeline



Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.

- This type of job scans branches in a Git repository, and looks for Jenkinsfiles.

The screenshot shows the configuration for a Multibranch Pipeline job. Under the 'Branch Sources' section, the 'Git' provider is selected. The 'Project Repository' field contains the URL 'git@diyvb:repos/gradle-greetings'. The 'Credentials' dropdown is set to '- none -'. There is an 'Add' button next to it. The 'Ignore on push notifications' checkbox is unchecked. The 'Repository browser' dropdown is set to '(Auto)'. The 'Additional Behaviours' dropdown has an 'Add' option. The entire configuration is contained within a light gray box.

- Configured to look for Jenkinsfiles, which are expected to have a pipeline script.

The screenshot shows the 'Build Configuration' section. The 'Mode' dropdown is set to 'by Jenkinsfile'. The entire configuration is contained within a light gray box.

- If a Jenkinsfile is found in a branch in the designated repository, then Jenkins will create a "read-only" sub-job (with the branch name) to do the build processing outlined in the Jenkinsfile.



Multibranch Pipeline processing

- “Scan Multibranch (branch indexing) function initiated and detects any branch with Jenkinsfile
- Pipeline job automatically created in folder for branch and build started
- Branch job configuration is read-only (can be viewed but not changed)

Scan Multibranch Pipeline Log

```
Started by user Jenkins Admin
[Sun Mar 26 19:32:46 EDT 2017] Starting branch indexing...
> git rev-parse --is-inside-work-tree # timeout=10
Setting origin to git@diyvb2:/opt/git/gradle-demo
> git config remote.origin.url git@diyvb2:/opt/git/gradle-demo # timeout=10
Fetching & pruning origin...
Fetching upstream changes from origin
> git --version # timeout=10
> git fetch --tags --progress origin +refs/heads/*:refs/remotes/origin/* --prune
Getting remote branches...
Seen branch in repository origin/decl
Seen branch in repository origin/master
Seen branch in repository origin/test
Seen 3 remote branches
Checking branch decl
'Jenkinsfile' found
Met criteria
Scheduled build for branch: decl
Checking branch test
'Jenkinsfile' found
Met criteria
Changes detected: test (2ef42dd5d23e29942ffffecd9e4e58a0faf41d855 →
c0ef4193e327528941ccc701e96f7a16b531e1c8)
Scheduled build for branch: test
Checking branch master
'Jenkinsfile' not found
Does not meet criteria
Done.
[Sun Mar 26 19:32:47 EDT 2017] Finished branch indexing. Indexing took 1.1 sec
Finished: SUCCESS
```

Build Executor Status

Node	Status	Idle	Building	Queued	Running	Aborted	Blocked	Waiting	Retrying	Unstable	Unassigned
master	Idle	1	0	0	0	0	0	0	0	0	0
demo-all - test	Building	0	0	0	1	0	0	0	0	0	0
worker_node1	Idle	1	0	0	0	0	0	0	0	0	0
worker_node2	Idle	2	0	0	0	0	0	0	0	0	0
worker_node3	Idle	1	0	0	0	0	0	0	0	0	0

demo-all

Branches (2)

S	W	Name	Last Success	Last Failure	Last Duration	Fav
		decl	3 days 23 hr - #7	N/A	56 sec	
		test	3 days 23 hr - #8	N/A	45 sec	

Icon: S M L

Legend: RSS for all RSS for failures RSS for just latest builds

Lab 4 - Working with CI/CD Pipelines and Jenkins

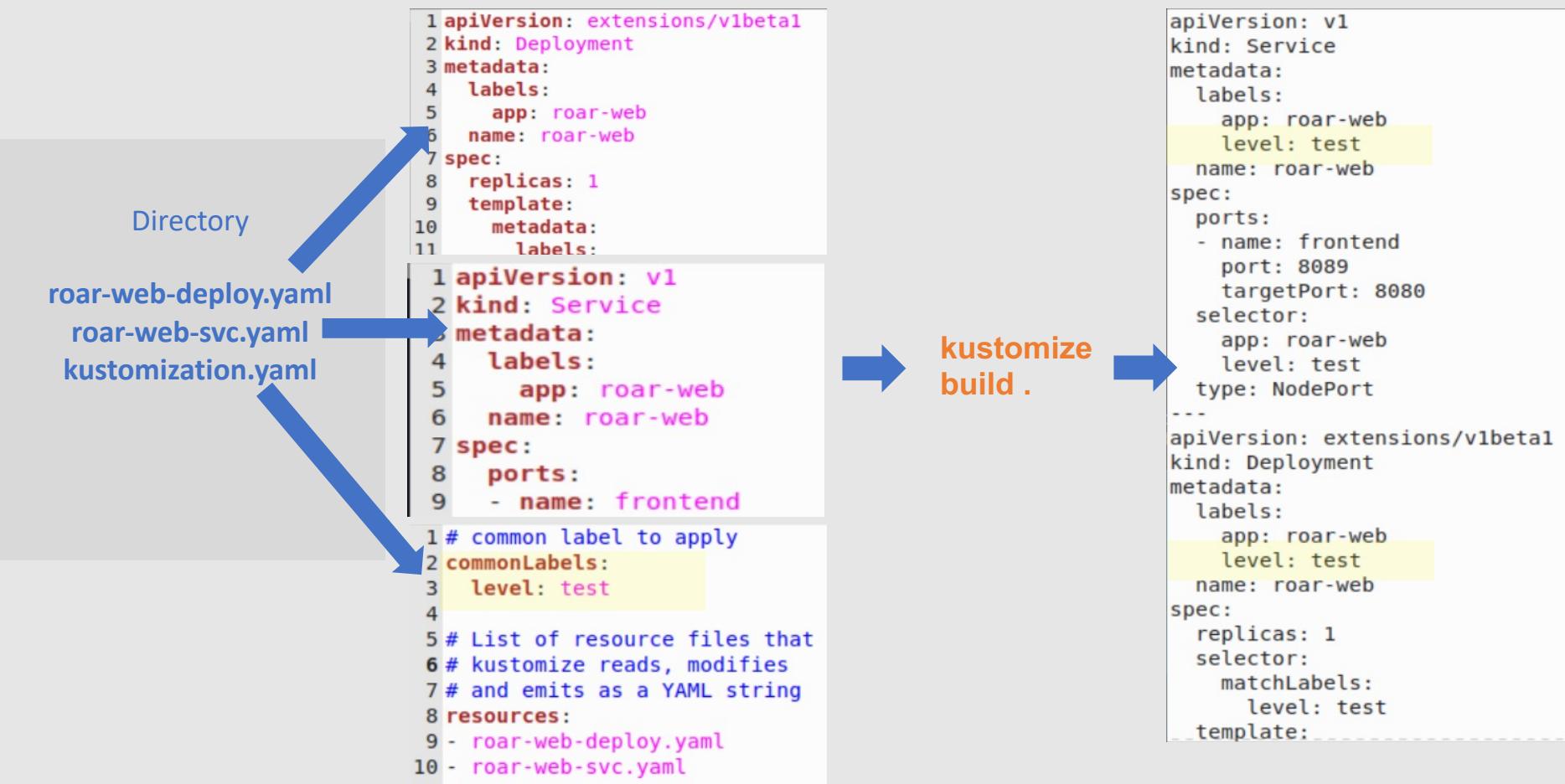


Kustomize

- Kustomize creates modified Kubernetes manifests by “overlays” declarative specifications on top of existing Kubernetes manifests
- Changes to make/overlay are stored in a separate file called *kustomization.yaml*
- Kustomize reads the *kustomization.yaml* file and the Kubernetes spec files (manifests) and dumps out completed resources to stdout with the changes from *kustomization.yaml* applied “on top of” the standard Kubernetes files
- *kustomize build* command causes files to be processed and output created
- Kustomize leverages the idea of “transformers”
- Transformers are functionality built into Kustomize that “transforms” your Kubernetes manifests given a simple set of declarative rules



Simple Example





Variants

- Common use case is needing multiple variations (variants) of a set of common resources (i.e. dev, stage, prod)
- To do this, Kustomize has concept of “overlay” and “base”
 - Both represented via kustomization.yaml file (includes resources + customizations)
 - Base – declares things variants have in common
 - Overlays – declare the differences
- Variants can also apply “patches”
 - Patch – partial deployment spec – used to update some existing spec as opposed to adding new information



Jenkinsfile - Part 1

```
1 pipeline {
2     agent { label 'worker_node1' }
3     environment {
4         STAGE_VERSION = "0.0.${BUILD_NUMBER}"
5         RC_VERSION = "1.0.${BUILD_NUMBER}"
6     }
7     stages {
8         stage('Source') {
9             steps {
10                 cleanWs()
11                 checkout scm
12                 // sh "git clone -b main git@localhost:/git/repos/roar-min"
13             }
14         }
15         stage('Compile') {
16             tools {
17                 gradle 'gradle5'
18             }
19             steps {
20                 sh 'gradle -PSTAGE_VERSION=$STAGE_VERSION clean compileJava assemble'
21                 stash includes: '**/web*.war', name: 'roar'
22             }
23         }
24         stage('Package-Test') {
25             steps {
26                 unstash 'roar'
27                 git branch: 'test', url: 'https://github.com/brentlaster/roar-min-docker'
28                 sh "docker build -f Dockerfile_roar_db_image -t localhost:5000/roar-db:$
{STAGE_VERSION} ."
29                 sh "docker build -f Dockerfile_roar_web_image --build-arg warFile=web/build/libs/web-$
{STAGE_VERSION}.war -t localhost:5000/roar-web:${STAGE_VERSION} ."
30                 sh "docker push localhost:5000/roar-db:${STAGE_VERSION}"
31                 sh "docker push localhost:5000/roar-web:${STAGE_VERSION}"
32             }
33         }
34     }
35 }
```



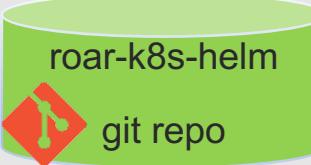
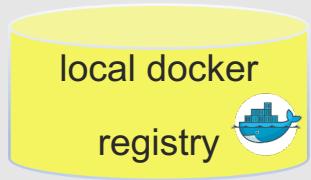
Jenkinsfile - Part 2

```
34     stage('Package-Prod') {
35         steps {
36             unstash 'roar'
37             sh "mv web/build/libs/web-${STAGE_VERSION}*.war web/build/libs/web-${RC_VERSION}*.war"
38             git branch: 'main', url: 'https://github.com/brentlaster/roar-min-docker'
39             sh "docker build -f Dockerfile_roar_db_image -t localhost:5000/roar-db:${RC_VERSION} ."
40             sh "docker build -f Dockerfile_roar_web_image --build-arg warFile=web/build/libs/web-$
41             ${RC_VERSION}*.war -t localhost:5000/roar-web:${RC_VERSION} . "
42             sh "docker push localhost:5000/roar-db:${RC_VERSION}"
43             sh "docker push localhost:5000/roar-web:${RC_VERSION}"
44         }
45     }
46     stage('Deploy STAGE') {
47         steps {
48             git branch: 'main', url: 'git@localhost:/git/repos/roar-min-deploy'
49             sh "git config --global user.email 'argocd@ci.com' && git config --global user.name
50             'argocd_user'"
51             sh "git checkout main"
52             sh "cd ./overlays/stage/db && kustomize edit set image localhost:5000/roar-db:$
53             ${STAGE_VERSION}"
54             sh "cd ./overlays/stage/web && kustomize edit set image localhost:5000/roar-web:$
55             ${STAGE_VERSION}"
56             sh "git commit -am 'Publish new staging release' && git push origin main:main || echo
57             'no change'"
58         }
59     }
60     stage('Deploy RC') {
61         steps {
62             sh "git checkout main"
63             sh "cd ./overlays/prod/db && kustomize edit set image localhost:5000/roar-db:$
64             ${RC_VERSION}"
65             sh "cd ./overlays/prod/web && kustomize edit set image localhost:5000/roar-web:$
66             ${RC_VERSION}"
67             sh "git commit -am 'Publish new release candidate' && git push origin main:main ||
68             echo 'no change'"
69         }
70     }
71 }
```

Lab 5 - Deploying to Production



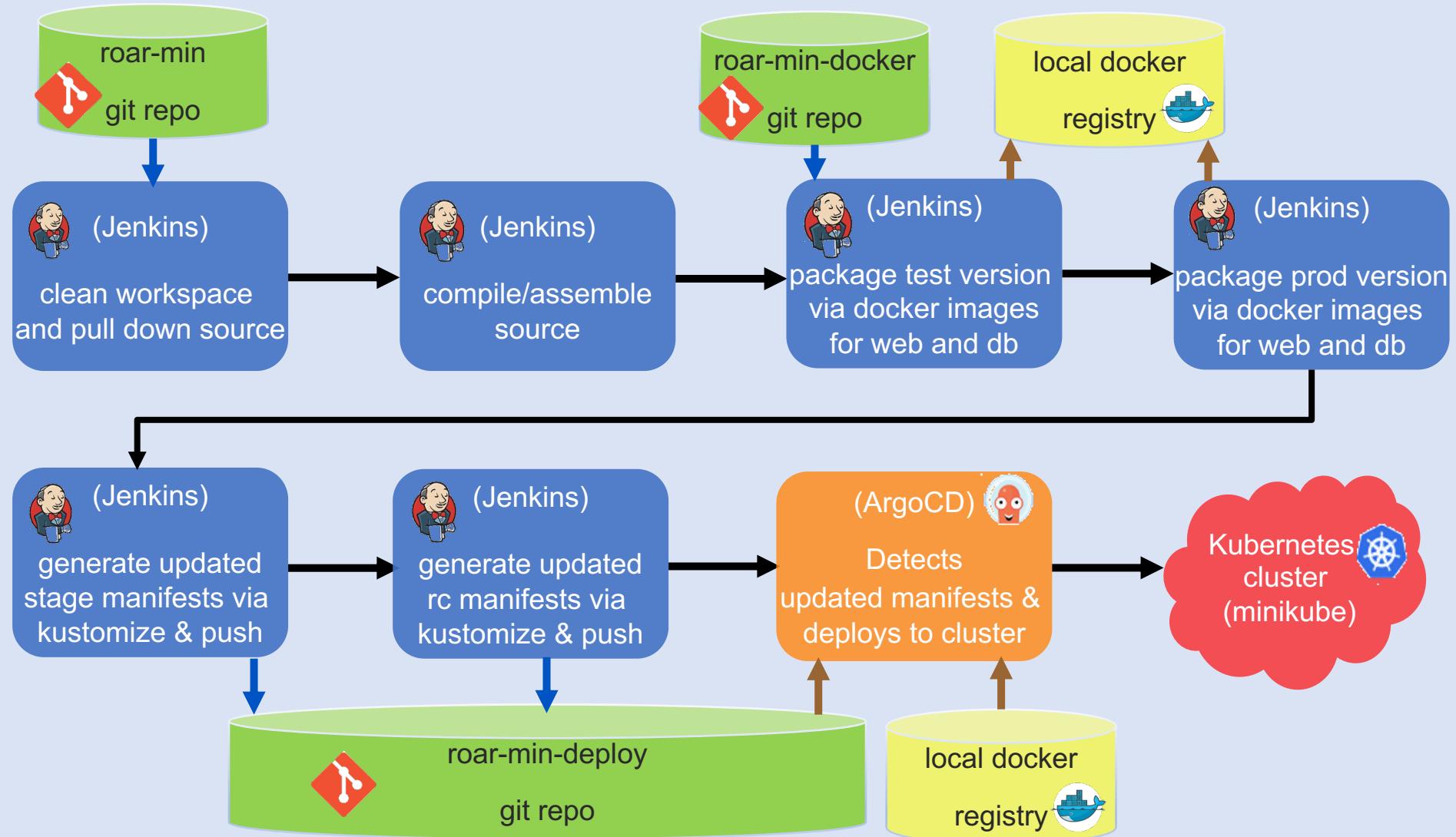
Project Repositories



- Set of source code for example app
- Docker files for packaging webapp and database pieces as Docker images
- Local registry to hold built images from above and serve to Kubernetes
- Kustomize files for updating versions numbers in app images
- Version of sample app managed via Helm



CI/CD Pipeline with ArgoCD





Environment Variables for Builds

- Used by Helm, custom tools, Jsonnet
- KUBE_API_VERSIONS - version of Kubernetes API
- KUBE_VERSION - version of Kubernetes
- ARGOCD_APP_SOURCE_TARGET_REVISION - target revision from spec, such as “master”
- ARGOCD_APP_SOURCE_REPO_URL - repo’s URL
- ARGOCD_APP_SOURCE_PATH - app path within repo
- ARGOCD_APP_REVISION - resolved revision such as Git SHA
- ARGOCD_APP_NAMESPACE - application namespace destination
- ARGOCD_APP_NAME - application name



Additional Topics: Resource Hooks

- Allows running scripts before, after, or during a Sync operation
- Can also be run if Sync operation fails
- Implemented as K8s manifests in source repository of App
 - annotated with argocd.argoproj.io/hook
- Use cases
 - PreSync hook - validate schema or such before deploying a new version
 - Sync hook - orchestrate complex deployment
 - PostSync hook - validation checks after deployment
 - SyncFail hook - clean-up/finalizer if operation fails
 - Skip hook - tells ArgoCD to skip application of the manifest

```
apiVersion: batch/v1
kind: Job
metadata:
  generateName: schema-validate-
  annotations:
    argocd.argoproj.io/hook: PreSync
```



Additional Topics: Notifications

- Not part of Argo CD
- Provided by integrations with 3rd-party pieces
 - Prometheus, Grafana – monitoring performance & health of applications
<https://argoproj.github.io/argo-cd/operator-manual/metrics/>
 - ArgoCD Notifications – continuously monitors Argo CD applications; integrates with SMTP, Slack, Discord, etc.
<https://github.com/argoproj-labs/argocd-notifications>
 - ArgoCD Kube Notifier – generic K8s resource controller for monitoring any resource and sending notification when a configured rule is met
<https://github.com/argoproj-labs/argo-kube-notifier>
 - Kube Watch – watches cluster for resource changes and sends notifications to Slack, mattermost, etc. via webhooks
<https://github.com/bitnami-labs/kubewatch>

Lab 6 - Seeing the entire workflow



That's all - thanks!

The screenshot shows a web browser window with the URL techskillstransformations.com. The page features a dark header with the company name "TECH SKILLS TRANSFORMATIONS, LLC". Below the header, a large banner on the left displays the book cover for "Professional Git 1st Edition" by Brent Laster, showing a roller coaster track. The banner text includes "Professional Git 1st Edition", "by Brent Laster (Author)", "5 customer reviews", and a "Look Inside" button. The main content area has a dark background with white text. It features the slogan "TECH LEARNING MADE EASY" and a call-to-action button "Upcoming live training with O'Reilly Media!". To the right of the text is a photograph of a person sitting at a desk, looking at a laptop screen while holding a notebook. Below this is an advertisement for the book "Jenkins 2 Up & Running" by Brent Laster, featuring a drawing of a fox.

New courses on Git, K8S, Operators, GitOps and more!

TECH SKILLS TRANSFORMATIONS, LLC

Home Contact Us

TECH LEARNING MADE EASY

Get the instruction you need to upskill now! Click the button below to see upcoming trainings.

Upcoming live training with O'Reilly Media!

ABOUT US

techskillstransformations.com
getskillsnow.com

O'REILLY®

Jenkins 2 Up & Running

EVOLVE YOUR DEPLOYMENT PIPELINE FOR NEXT GENERATION AUTOMATION

Brent Laster