

Version 1.1  
8/10/19

# Prerequisites

- Modern system with 16G of memory and 35G free storage
- VirtualBox installed and running
- Virtual machine (.ova file) installed in VirtualBox
  - <https://www.dropbox.com/s/kmae9ghgdoihsis/jx-intro.ova?dl=0>
- Setup doc is at
  - <https://github.com/brentlaster/safaridocs/blob/master/jxi-setup.pdf>
- Free GitHub account (signup at [github.com](http://github.com))
- Labs doc for class
  - <https://github.com/brentlaster/safaridocs/blob/master/jxi-labs.pdf>
- Previous Jenkins 2 class or experience recommended

# Introduction to Jenkins X

Brent Laster

# About me

- Senior Manager, R&D
- Global trainer – training (Git, Jenkins, Gradle, Gerriit, Continuous Pipelines)
- 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

# Book – Professional Git

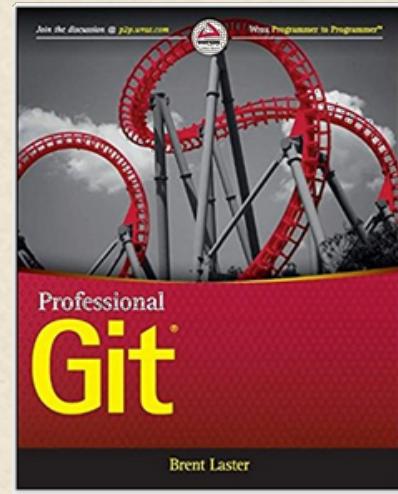
- Extensive Git reference, explanations, and examples
- First part for non-technical
- Beginner and advanced reference
- Hands-on labs

**Professional Git 1st Edition**

by Brent Laster (Author)

 7 customer reviews

[Look inside](#) 



 Amazon Customer

 **I can't recommend this book more highly**

February 12, 2017

Format: Kindle Edition

Brent Laster's book is in a different league from the many print and video sources that I've looked at in my attempt to learn Git. The book is extremely well organised and very clearly written. His decision to focus on Git as a local application for the first several chapters, and to defer discussion about it as a remote application until later in the book, works extremely well.

Laster has also succeeded in writing a book that should work for both beginners and people with a fair bit of experience with Git. He accomplishes this by offering, in each chapter, a core discussion followed by more advanced material and practical exercises.

I can't recommend this book more highly.

 **Ideal for hands-on reading and experimentation**

February 23, 2017

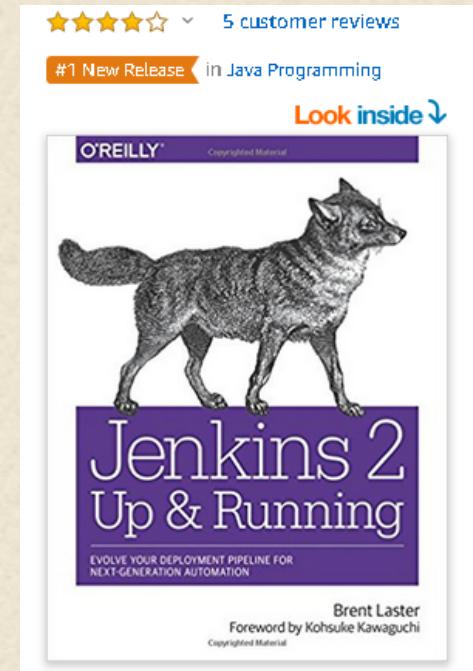
Format: Paperback | **Verified Purchase**

I just finished reading Professional Git, which is well organized and clearly presented. It works as both a tutorial for newcomers and a reference book for those more experienced. I found it ideal for hands-on reading and experimentation with things you may not understand at first glance. I was already familiar with Git for everyday use, but I've always stuck with a convenient subset. It was great to be able to finally get a much deeper understanding. I highly recommend the book.

# Jenkins 2 Book

- Jenkins 2 – Up and Running
- Additional reference for this material
- “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*



★★★★★ This is highly recommended reading for anyone looking to use Jenkins 2 to ...

By [Leila](#) on June 2, 2018

Format: Paperback

Brent really knows his stuff. I'm already a few chapters in, and I'm finding the content incredibly engaging. This is highly recommended reading for anyone looking to use Jenkins 2 to implement CD pipelines in their code.

★★★★★ A great resource

By [Brian](#) on June 2, 2018

Format: Paperback

I have to admit that most of the information I get usually comes through the usual outlets: stack overflow, Reddit, and others. But I've realized that having a comprehensive resource is far better than hunting and pecking for scattered answers across the web. I'm so glad I got this book!

# Agenda

- Introduction to Jenkins X - what it is, benefits, motivation, practices, technologies it uses
- Crash course in Containers and Kubernetes
- Using Jenkins X to create a cluster
- Install jx on the cluster
- Exploring environments
- Quickstarts - how to quickly and easily spin up new projects using Jenkins X
- Preview environments
- Promoting to production in Jenkins X

# A bit of terminology...

- Pipeline – an automated, repeatable set of processes chained together to transform source code into a deployable product
- Continuous Integration (CI) - the process of automatically detecting, getting, and building updates as source code or configuration is changed for a product. CI is the activity that starts the pipeline
- Continuous Delivery (CD) – the ability to produce deployable products quickly, reliably, and consistently through automated processes responding to code or configuration changes
- DevOps – a set of practices (including CI/CD) that automate and simplify the processes and promote collaboration between dev teams and ops teams
- Cloud-ready / native / friendly – working well in a cloud environment where applications may need to be scaled, restarted, etc. and you pay for resources and time used
- Jenkins – a build and workflow orchestration tool
- Jenkins 2 – an evolution of Jenkins that focused on pipelines-as-code
- Jenkins X ...?

# What is Jenkins X?

- A way to automatically create and run CI/CD pipelines in a cloud (or cloud-like) environment.
- A framework with a set of tools and processes for doing the above.
- Think of it as a way to get “on-demand” pipelines via simple commands and develop applications with industry best practices and workflows already built-in.

# What benefits does Jenkins X provide?

- **Cloud-ready development environment**
  - Ability to let you develop in any cloud
  - Spin up or tear down environments as needed
  - Easy pipeline for software development
  - Automatically implement CI/CD and DevOps best practices
  - Leverages best-of-breed applications, such as Jenkins and Kubernetes
- **Automates the hard stuff for setup and workflow**
  - Running one or two commands in Jenkins X can create an end-to-end CI/CD pipeline in a Kubernetes cluster, including:
    - » Git Repositories
    - » Promotion environments
    - » Webhooks to trigger actions
  - Includes the ability to create projects tailored for any number of common programming environments
    - » “Quickstart” project templates for Go, Python; build-in templates for Spring, etc.
  - Simple commands to move things through the pipeline
- **Managed promotion model with automatic environments**
- **Automatic feedback**
  - Updates pull requests and GitHub issue based on what is occurring and success/failure

# What are the motivations for Jenkins X?

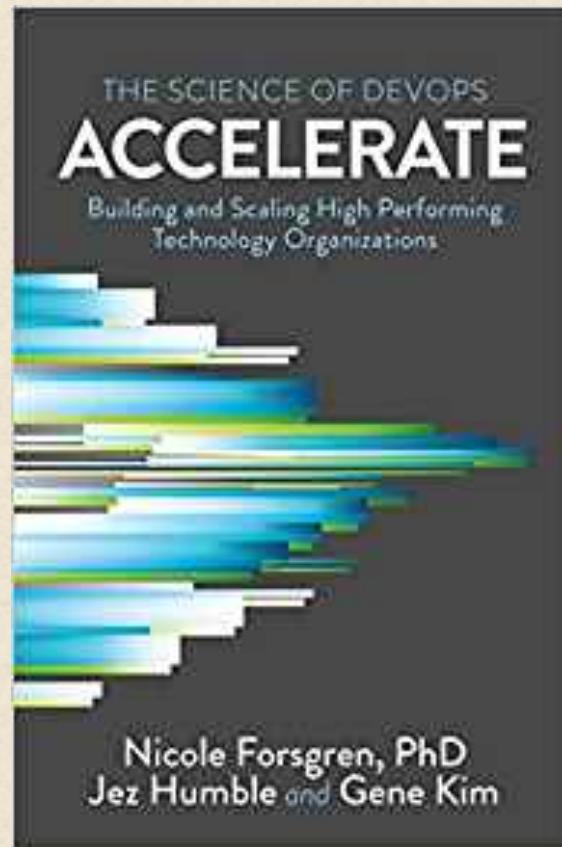
11

- Allow developers and others to focus on the code going through the pipeline rather than having to worry about the pipeline itself.
- Automate all the stuff that usually bogs down people trying to work with the technologies that it uses (namely Kubernetes).
- Be simple to use.
- Be cloud-friendly. Work well in any cloud or on-premise cloud-like environment.
- Leverage the industry-standard technologies
- Make it easy to do the “right thing” – use industry best practices.
  - As defined by research including the “Accelerate” book and “State of Devops” reports.

11

# Accelerate!

- Jenkins X incorporates the best practices from the book “Accelerate” and the DevOps Institute’s State of DevOps reports.
- Those document what works and doesn’t work for high-performing DevOps organizations
- Implemented as seven core capabilities (implementation built-in) for Jenkins X.



# 7 Capabilities of Jenkins X

(<https://jenkins-x.io/about/accelerate>)

- 
-  1 Use version control for all artifacts.
  -  2 Automate your deployment process.
  -  3 Use trunk-based development.
  -  4 Implement continuous integration.
  -  5 Implement continuous delivery.
  -  6 Use loosely coupled architecture.
  -  7 Architect for empowered teams.

# What technologies does Jenkins X use?

14

- Containers
- Docker
- Kubernetes
- Helm
- Git
- Static version
  - Jenkins
- Serverless version
  - Prow
  - Tekton pipelines
- We leverage the static version for an easier transition and understanding

14

# Crash Course on Containers and Kubernetes

# What are Containers?

- A container is a standard unit of software that functions like a fully provisioned machine installed with all the software needed to run an application.
- It's a way of packaging software so that applications and their dependencies have a self-contained environment to run in, are insulated from the host OS and other applications - and are easily ported to other environments.
- A container is NOT a VM. A container leverages several features of the Linux OS to "carve out" a self-contained space to run in.

What's in a container?



**Containers are running instances of images**  
**Images define what goes into a container.**  
**Containers are built from images.**

# What is Docker?

- (Marketing) From docker.com

An open platform for distributed applications for developers and sysadmins.

Open source project to ship any app as a lightweight container

- (Technical)

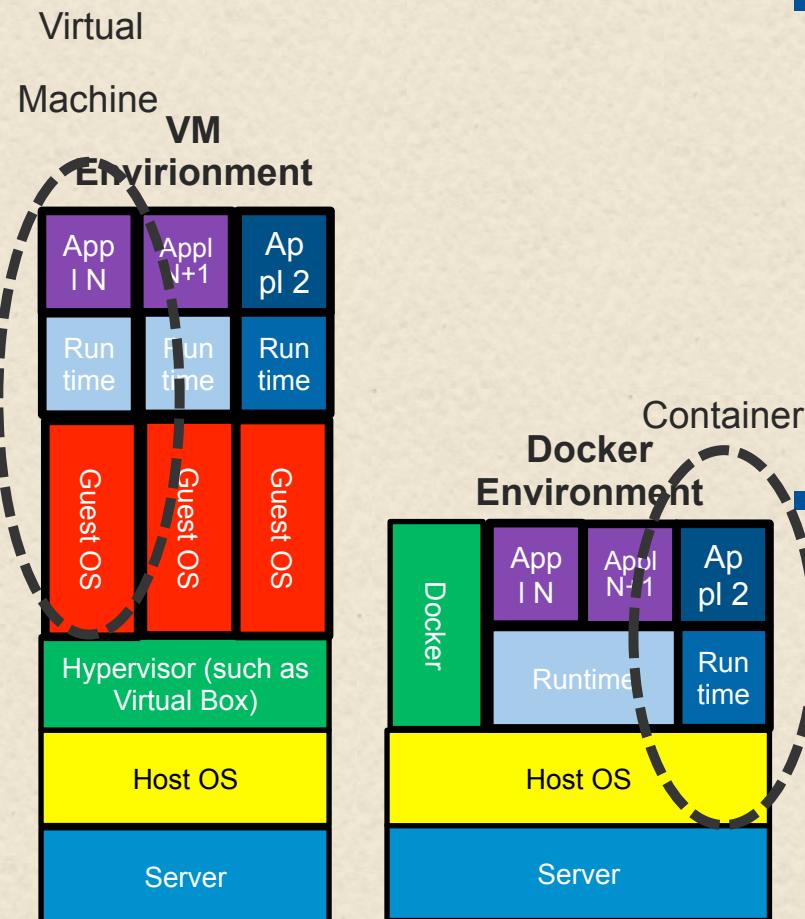
Thin wrapper around Linux Container technology

Leverages 3 functionalities from Linux to provide isolated environment in the OS

- » union filesystem - data
- » namespaces - visibility (pid)
- » cgroups - control groups (resources)

- Provides restful interface for service
- Provides description format for containers
- Provides API for orchestration

# How Containers from Docker differ from VM



- A VM requires a Hypervisor and a Guest OS to create isolation; Docker uses Linux container technologies to run processes in separate spaces on the same OS

Because it doesn't require the Hypervisor and a Guest OS, Docker is:

- Faster to startup
- More portable (can run an image unchanged in multiple environments)

# Dockerfile to Image to Container

```
FROM node:argon

# Create app directory
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

# Install app dependencies
COPY package.json /usr/src/
RUN npm install

# Bundle app source
COPY . /usr/src/app

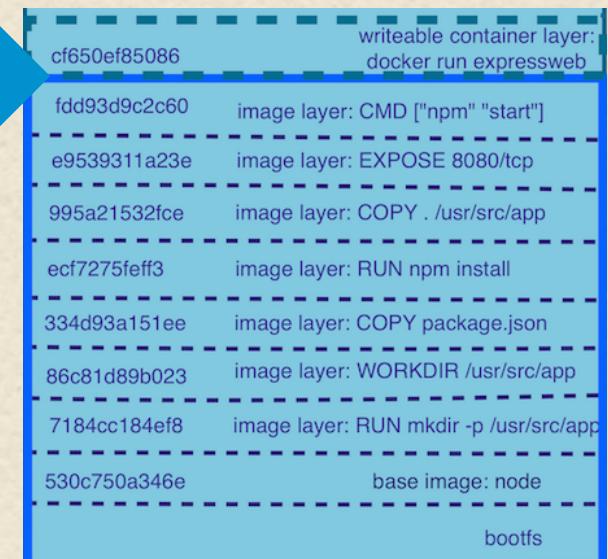
EXPOSE 8080
CMD [ "npm", "start" ]
```

**Build**

```
$ docker build -t expressweb .
Step 1 : FROM node:argon
argon: Pulling from library/node...
...
Status: Downloaded newer image for node:argon
--> 530c750a346e
Step 2 : RUN mkdir -p /usr/src/app
--> Running in 5090fde23e44
--> 7184cc184ef8
Removing intermediate container 5090fde23e44
Step 3 : WORKDIR /usr/src/app
--> Running in 2987746b5fba
--> 86c81d89b023
Removing intermediate container 2987746b5fba
--> 334d93a151ee
Removing intermediate container a678c817e467
Step 5 : RUN npm install
--> Running in 31ee9721cccb
--> ecf7275feff3
Removing intermediate container 31ee9721cccb
Step 6 : COPY . /usr/src/app
--> 995a21532fce
Removing intermediate container a3b7591bf46d
Step 7 : EXPOSE 8080
--> Running in fddb8afb98d7
--> e9539311a23e
Removing intermediate container fddb8afb98d7
Step 8 : CMD npm start
--> Running in a262fd016da6
--> fdd93d9c2c60
Removing intermediate container a262fd016da6
Successfully built fdd93d9c2c60
```

**Run**

docker history <image>			
IMAGE	CREATED	CREATED BY	SIZE
fdd93d9c2c60	2 days ago	/bin/sh -c CMD ["npm" "start"]	0 B
e9539311a23e	2 days ago	/bin/sh -c EXPOSE 8080/tcp	0 B
995a21532fce	2 days ago	/bin/sh -c COPY dir:50ab47bff7	760 B
ecf7275feff3	2 days ago	/bin/sh -c npm install	3.439 MB
334d93a151ee	2 days ago	/bin/sh -c COPY file:551095e67	265 B
86c81d89b023	2 days ago	/bin/sh -c WORKDIR /usr/src/app	0 B
7184cc184ef8	2 days ago	/bin/sh -c mkdir -p /usr/src/app	0 B
530c750a346e	2 days ago	/bin/sh -c CMD ["node"]	0 B



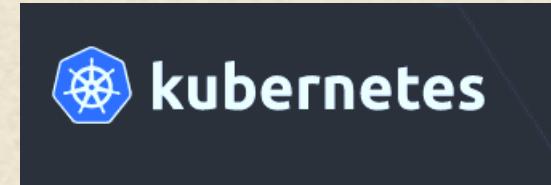
# How do we think about this?

- Consider analogy of installing software on a machine...
- And then provisioning systems for users



# What is Kubernetes?

- A portable, extensible platform for managing containerized workloads and services (cluster orchestration system)
- Name derived from Greek for helmsman, thus the icon
- Frequently abbreviated as “k8s”
- Formerly known as “borg” – an internal Google project
- Open-sourced by Google in 2014
- Groups containers together for an application as a logical unit called a “pod”.
- Goal is to provide a robust platform for running many containers.
- Allows automation of deployment, scaling, and managing containerized workloads.
- Kubernetes provides you with a framework to run distributed systems (of containers) resiliently.
- Takes care of
  - scaling requirements
  - failover
  - deployment patterns



# So how do we think about this?

- Analogy: Datacenter for containers
  - If we think of images/containers as being like computers we stage and use
  - We can think of Kubernetes as being like a datacenter for those containers
  - Main jobs of datacenter
    - » Provide systems to service needs (regardless of the applications)
    - » Keep systems up and running
    - » Add more systems / remove systems depending on load
    - » Deal with systems that are having problems
    - » Deploy new systems when needed
      - Provide simple access to pools of systems
      - Etc..



# Kubernetes is everywhere

- Has effectively “won the war” over other competitors
  - Docker swarm
  - Mesos
- Cloud providers all endorse it and provide ways to get a Kubernetes cluster
- Implementations on other enterprise platforms

The image contains three side-by-side screenshots of web browsers displaying Kubernetes services from major cloud providers:

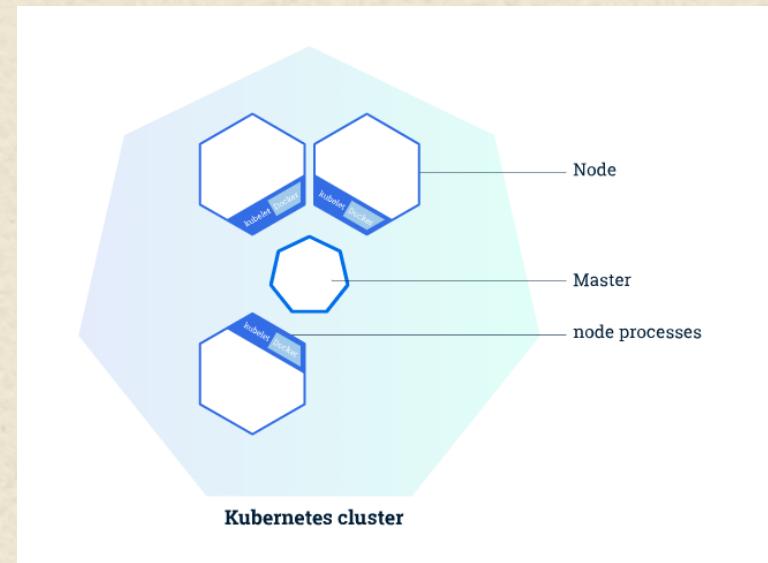
- Amazon EKS:** The screenshot shows the Amazon EKS landing page with a dark blue background featuring hexagonal patterns. It includes a prominent button labeled "Start using Amazon EKS".
- Google Kubernetes Engine:** The screenshot shows the Google Cloud Platform (GCP) interface for Kubernetes Engine. It highlights "KUBERNETES ENGINE" and describes it as a "Reliable, efficient, and secured way to run Kubernetes clusters". It features two buttons: "VIEW KUBERNETES ENGINE DOCS" and "VIEW MY CONSOLE".
- Azure Kubernetes Service (AKS):** The screenshot shows the Microsoft Azure interface for AKS. It describes AKS as a "Highly available, secure, and fully managed Kubernetes service". It includes a "Explore Kubernetes learning path" button and a "certified kubernetes" badge.

The screenshot shows the Red Hat OpenShift website at <https://www.openshift.com/learn/topics/kubernetes/>. The page features a "TECH TOPIC" section titled "Hybrid cloud, enterprise Kubernetes". It states that Red Hat® OpenShift® is supported Kubernetes for cloud-native applications with enterprise security. A red "Download technology detail" button is visible.

The screenshot shows the Microsoft Azure interface for Azure Kubernetes Service (AKS). It describes AKS as a "Highly available, secure, and fully managed Kubernetes service". It includes a "Explore Kubernetes learning path" button and a "certified kubernetes" badge.

# What is a K8s Cluster?

- Kubernetes is about clusters
  - A k8s cluster is an HA set of computers coordinated by k8s to work as a unit.
  - Abstractions we have in k8s allow you to deploy containers in the cluster w/o tying them to a specific host.
  - K8s automates distribution and scheduling of containers across cluster in an efficient manner.
  - Nodes in a cluster are either
    - master – coordinates the work
    - nodes – run the applications
  - Simplest example is a one node cluster, such as minikube



# What is minikube?

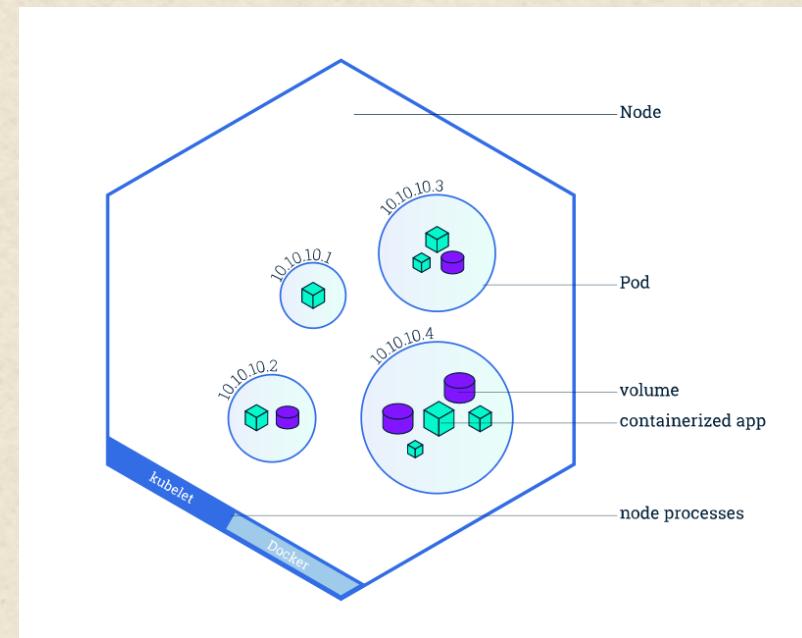
- Minikube is a simple program to learn about or work with small k8s deployments
- Creates a VM on your local machine
- Deploys a cluster containing only one node
- Has a CLI that starts and stops it, etc.
- Otherwise behaves like standard k8s

# K8s Quick Terminology

- Pods – object that contains and manages one or more containers and any attached volumes
- Service – abstraction that groups together pods based on identifiers called labels (or other characteristic)
- Deployment – defines a stateless app with a set number of pod replicas (scaled instances)
- Ingress – resource that lets cluster applications be exposed to external traffic
- Namespace - a logical area that groups k8s items like pods

# How are containers organized on K8S?

- K8s clusters have nodes
- Nodes run pods
- Pods are wrapped around one or more containers
- Pods are front-ended by services
- Pods are scaled (replicated) by deployments
- Namespaces group objects like pods, services, deployments together



# Data Center Analogy

- Functions: Uptime, scaling, redundancy...

- Container in a pod ~ server in a rack



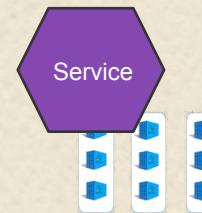
- Pod ~ rack of servers



- Deployment ~ multiple racks (replicas)



- Service ~ central control / login server



- Namespace ~ server room



# Looking at Kubernetes objects

- Command line utility – kubectl
- Dashboard
  - Browse around objects
  - Select by namespace
  - Drill in to see details

The image displays two side-by-side screenshots of the Kubernetes dashboard interface.

**Left Screenshot: Namespaces**

This screenshot shows the "Namespaces" section of the dashboard. The left sidebar has "Namespaces" selected under "Cluster". The main table lists the following namespaces:

Name	Labels	Status	Age
jx-production	env: production team: jx	Active	7 minutes
jx-staging	env: staging team: jx	Active	8 minutes
jx	env: dev team: jx	Active	11 minutes
default	-	Active	an hour
kube-node-lease	-	Active	an hour

**Right Screenshot: Pods**

This screenshot shows the "Pods" section of the dashboard. The left sidebar has "Pods" selected under "Workloads". The main table lists the following pods:

Namespace	Name	Node	Status	Restarts	Age
jx	jenkins-x-heapster-ff6d...	minikube	Running	0	14 minutes
jx	jenkins-x-nexus-6bc78...	minikube	Running	0	14 minutes
jx	jenkins-f94f447fd-4znng	minikube	Running	0	14 minutes
jx	jenkins-x-controllerwor...	minikube	Running	0	14 minutes
jx	jenkins-x-controllerrole...	minikube	Running	0	14 minutes
jx	jenkins-x-docker-registr...	minikube	Running	0	14 minutes
jx	jenkins-x-controllerteam...	minikube	Running	0	14 minutes
jx	jenkins-x-chartmuseum-5...	minikube	Running	0	14 minutes

# Jenkins X Initial Setup

- Prereq: Infrastructure that can run Kubernetes with Kubernetes installed on it
- Can be either on-prem or in external cloud
- Install `jx`
  - » Instructions <https://jenkins-x.io/getting-started/install/>
- Afterwards, ready to create new cluster or attempt to use existing one

# jx – Command Line tool for Jenkins X

- Syntax: jx [options] command [command arguments]
- Options: -b, --batch-mode, --verbose, --help

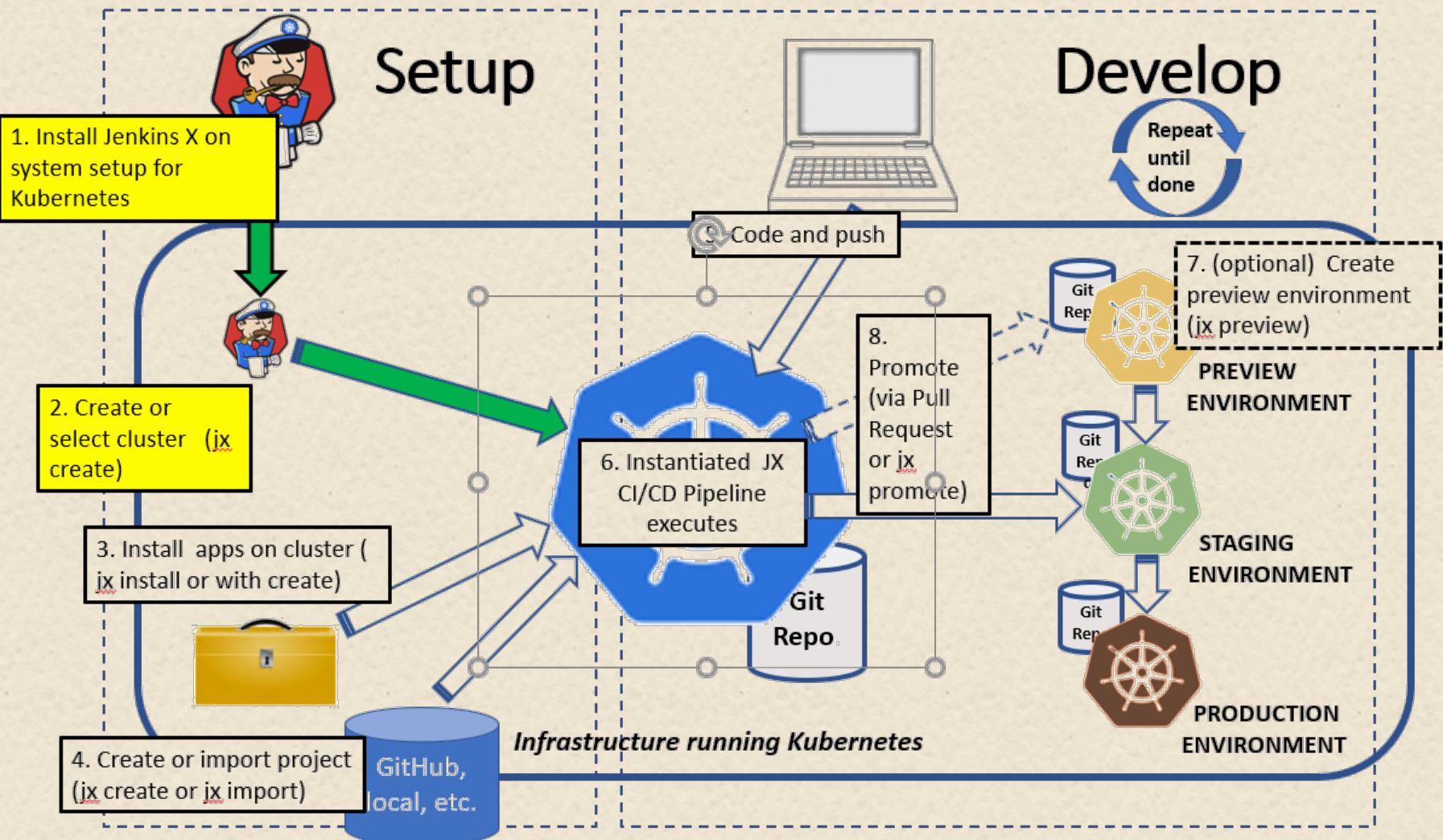
- [jx add](#) - Adds a new resource
- [jx boot](#) - Boots up Jenkins X in a Kubernetes cluster using GitOps and a Jenkins X Pipeline
- [jx cloudbees](#) - Opens the CloudBees app for Kubernetes for visualising CI/CD and your environments
- [jx completion](#) - Output shell completion code for the given shell (bash or zsh)
- [jx compliance](#) - Run compliance tests against Kubernetes cluster
- [jx console](#) - Opens the Jenkins console
- [jx context](#) - View or change the current Kubernetes context (Kubernetes cluster)
- [jx controller](#) - Runs a controller
- [jx create](#) - Create a new resource
- [jx delete](#) - Deletes one or more resources
- [jx diagnose](#) - Print diagnostic information about the Jenkins X installation
- [jx docs](#) - Open the documentation in a browser
- [jx edit](#) - Edit a resource
- [jx environment](#) - View or change the current environment in the current Kubernetes cluster
- [jx gc](#) - Garbage collects Jenkins X resources
- [jx get](#) - Display one or more resources
- [jx import](#) - Imports a local project or Git repository into Jenkins
- [jx init](#) - Init Jenkins X
- [jx install](#) - Install Jenkins X in the current Kubernetes cluster
- [jx login](#) - Onboard an user into the CloudBees application
- [jx logs](#) - Tails the log of the latest pod for a deployment
- [jx namespace](#) - View or change the current namespace context in the current Kubernetes cluster
- [jx open](#) - Open a service in a browser
- [jx options](#) -
- [jx preview](#) - Creates or updates a Preview Environment for the current version of an application
- [jx profile](#) - Set your jx profile
- [jx promote](#) - Promotes a version of an application to an Environment
- [jx prompt](#) - Generate the command line prompt for the current team and environment
- [jx repository](#) - Opens the web page for the current Git repository in a browser
- [jx rsh](#) - Opens a terminal in a pod or runs a command in the pod
- [jx scan](#) - Perform a scan action
- [jx shell](#) - Create a sub shell so that changes to the Kubernetes context, namespace or environment remain local to the shell
- [jx start](#) - Starts a process such as a pipeline
- [jx status](#) - status of the Kubernetes cluster or named node
- [jx step](#) - pipeline steps
- [jx stop](#) - Stops a process such as a pipeline
- [jx sync](#) - Synchronises your local files to a DevPod
- [jx team](#) - View or change the current team in the current Kubernetes cluster
- [jx uninstall](#) - Uninstall the Jenkins X platform
- [jx update](#) - Updates an existing resource
- [jx upgrade](#) - Upgrades a resource
- [jx version](#) - Print the version information

# Cluster Setup

- Can have jx set one up for you
  - “jx create cluster”
  - Requires parameters
    - » Amount of memory
    - » CPU cores
    - » Disk space
    - » Driver
- Can use existing cluster
  - Must be suitable and meet requirements for running jx
  - To check that, run
    - “jx compliance”

# Lab 1: Creating a Cluster with Jenkins X

# Jenkins X Overall Workflow



# Installing Jenkins X on the Cluster

- Installs set of client binaries
- Configures associated Git repositories
- Installs the Jenkins X platform
- By default, “`jx create cluster`” installs it on cluster
- For existing cluster, can use “`jx install`” command
- Prerequisites:
  - Kubernetes > 1.8
  - RBAC (Role-based Access Controls) enabled
  - Default cluster dynamic storage class for provisioning persistent volumes
- Example – installing to an on premise Kubernetes cluster

```
jx install --provider=kubernetes --on-premise
```

- Choice to install either of 2 versions

# Jenkins X comes in two versions

- Jx comes in two versions – static and serverless
- Overall interface and user experience is same for both versions
- Difference is how core CI/CD functionality is implemented
- Static Jenkins X
  - Jenkins leverages standard Jenkins 2 instance as core piece for CI/CD part of pipeline – runs in Kubernetes
  - Uses Jenkinsfiles
- Serverless Jenkins X (aka Next-Gen Jenkins X)
  - Instead of Jenkins 2, Jenkins X leverages tools called Prow and Tekton to produce pipeline code to run the CI/CD process
  - These tools allow for creating pipelines-as-code that run natively in a Kubernetes cluster instead of needing separate tool (like Jenkins)
  - Uses jenkins-x.yml file – similar syntax to declarative pipeline
  - Default

# What happens when we install Jenkins X into a cluster?

- Creates “jx” namespace and sets it as default
- Prompts for basic Git user info (name and email)
- Installs Helm orchestration/template tool for Kubernetes
  - Helm chart is set of template files that get values filled in to specify applications and configuration to be deployed in that environment
- Sets up ingress if needed
- Gets GitHub username
- Gets token for human user
- Gets token for bot user
- “Installs” tools
- Sets up API token to access static Jenkins instance
- Creates staging and production environments
  - Git repositories
  - Jenkins projects
  - Webhooks

# Managed promotion model

- Jenkins X provides a built-in model for code promotion
  - Creates several default environments for you
    - » Staging
    - » Production
  - Environments are effectively a pipeline in themselves
    - » Git repository
    - » Kubernetes namespace
  - Predefined promotion policies
    - » Always
    - » Never
  - Promotions done via Pull Requests (PRs)
  - Items (Git repo, K8s namespace are in addition to the ones for managing your projects' code)

# What gets “installed” with (static) Jenkins X?

39

- Jenkins X comes with and installs a minimum number of “best of breed” core applications to do CI/CD on K8s
- Helm – templating language for K8s manifests
- Applications get installed as Kubernetes objects in “jx” namespace

Standard Deployment Name	Purpose
jenkins	Standard Jenkins to provide CI/CD automation
jenkins-x-chartmuseum	Registry for publishing Helm charts
jenkins-x-nexus	Dependency cache for Nodejs and Java apps to improve build time
jenkins-x-monocular-api	UI used for discovering and running Helm charts
jenkins-x-docker-registry	Local docker registry
jenkins-x-controller*	Controllers in K8S take care of routine tasks to ensure desired state of cluster matches the observed state – processing for custom objects
jenkins-x-mongodb	Nosql database

- May also have K8s jobs (one-shot pods) named “gc\*”
  - Garbage collection that runs periodically

39

# Git and Jenkins X

- Jenkins X uses Git for
  - Repositories for managed environments
  - Quickstart repositories
  - Pull-requests and automation
- Defaults to GitHub
  - Can also use GitLab
  - Can also use Gitea (limited)
- Two users
  - Developer (jx human user)
    - » Need userid, name, and email
  - jx-bot (system automation process)
- jx needs Git API tokens to work with both

# GitHub Personal Access Tokens

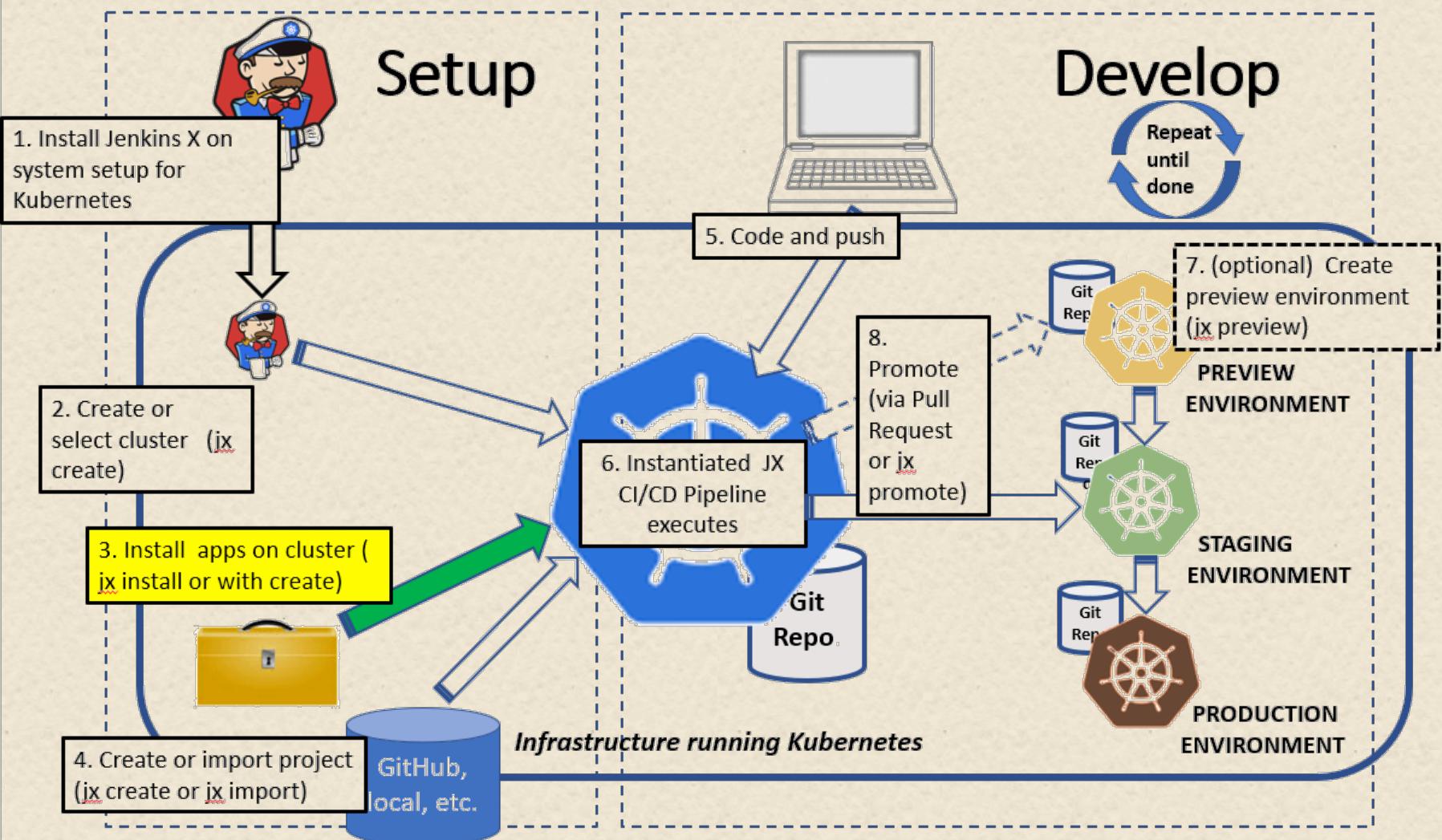
- Can create a personal access token and use in place of password when
  - Doing operations over HTTPS
  - Using command line or API
- Need existing account
- Fill in info for token and copy it – won't be displayed again
- Jenkins X supplies URL to generate token

The screenshots show the process of generating a new personal access token:

- Screenshot 1: New personal access token creation screen.** Shows a note field containing "jx-class" and a "Select scopes" section with several checkboxes. Scopes listed include: repo (repo, repo\_status, repo\_deployment, public\_repo, repo\_invite), write:packages, and read:packages.
- Screenshot 2: Available scopes list.** A detailed list of GitHub scopes with their descriptions:
  - user:follow: Follow and unfollow users
  - delete\_repo: Delete repositories
  - write:discussion: Read and write team discussions
  - read:discussion: Read team discussions
  - admin:enterprise: Full control of enterprises
    - manage\_billing:enterprise: Read and write enterprise billing data
    - read:enterprise: Read enterprise profile data
  - write:packages: Upload & delete packages in github package registry
  - read:packages: Download packages from github package registry
  - admin:gpg\_key: Full control of user gpg keys (Developer Preview)
    - write:gpg\_key: Write user gpg keys
    - read:gpg\_key: Read user gpg keys
- Screenshot 3: Personal access tokens list.** Shows a generated token: "a5e6c73234ffb04139ece9ea0". A message says "Make sure to copy your new personal access token now. You won't see it again!"

# Lab 2: Creating a default CI/CD pipeline in the cluster using Jenkins X

# Jenkins X Overall Workflow



# About Jenkins X Environments

- Jenkins X installation will have
  - Development Environment (dev)
    - » K8S namespace where tools are put (Jenkins, Nexus, etc.)
    - » Not linked to Git repository – will never promote changes
    - » Just K8S namespace for running applications while developing them
    - » Applications here are minimum to run CI/CD on K8s
    - » Scope: each team in an organization can have its own independent dev environment
  - Permanent Environments (staging and production to start)
    - » Environments where applications will run
    - » Can have more permanent ones if desired
    - » Linked to Git repositories
      - » Contains Helm chart defining
        - Which application charts to be installed
        - Which versions
        - Environment configuration
        - Additional resources needed
      - » To deploy to these, use Pull Request
      - » When PRs merged into environment's Git repo,
        - » Pipeline runs
        - » Applies helm chart in repo to K8s namespace
  - (Optional) Preview Environment – more later

# Staging and Production environments

Available after “`jx install`” completes

NAME	LABEL	KIND	PROMOTE	NAMESPACE	ORDER	CLUSTER	SOURCE	REF	PR
dev	Development	Development	Never	jx	0				
staging	Staging	Permanent	Auto	jx-staging	100		<a href="https://github.com/brentlaster/environment-daggersun-staging.git">https://github.com/brentlaster/environment-daggersun-staging.git</a>		
production	Production	Permanent	Manual	jx-production	200		<a href="https://github.com/brentlaster/environment-daggersun-production.git">https://github.com/brentlaster/environment-daggersun-production.git</a>		

```
brentlaster
environment-daggersun-production
env
  Chart.yaml
  requirements.yaml
  templates
  values.yaml
Jenkinsfile
jenkins-x.yml
LICENSE
Makefile
README.md
environment-daggersun-staging
env
  Chart.yaml
  requirements.yaml
  templates
  values.yaml
Jenkinsfile
jenkins-x.yml
LICENSE
Makefile
README.md
```

NAME	STATUS	AGE
default	Active	9h
jx	Active	9h
jx-production	Active	9h
jx-staging	Active	9h
kube-node-lease	Active	9h
kube-public	Active	9h
kube-system	Active	9h

brentlaster / environment-daggersun-production

No description, website, or topics provided.

Manage topics

31 commits · 1 branch · 0 releases · 6 contributors · Apache-2.0

Branch: master · New pull request

gwstudent5 Add environment configuration · Latest commit saes4cr · 9 hours ago

env · Add environment configuration · 9 hours ago

.gitignore · Initial commit · 2 years ago

.pre-commit-config.yaml · chore: added precommit configuration · last month

Jenkinsfile · Use correct namespace for environment · 9 hours ago

LICENSE · Initial commit · 2 years ago

Makefile · Use correct namespace for environment · 9 hours ago

README.md · Initial commit · 2 years ago

jenkins-x.yml · Use correct namespace for environment · 9 hours ago

README.md

default-environment-charts

The default git repository used when creating new GitOps based Environments

# Environments have a pipeline

- Git repositories for environments are separate from one you use for your project code
- Environments are used for promotion: dev->staging, staging->production
- Jenkins uses Git repositories to manage what is deployed to each environment
- Repositories contain Helm charts to specify applications to be deployed
- Promotions done via Git pull requests (PR)
- Above is known as GitOps
- Environments have pipeline to handle promotions

```

1 + jx step helm apply
2 Modified file '/home/jenkins/workspace/onment-divemirror-staging_master/env/Chart.yaml' to set the
3 chart to version 1
4 Copying the helm source directory '/home/jenkins/workspace/onment-divemirror-staging_master/env' to a
5 temporary location for building and applying '/tmp/jx-helm-apply-046268026/env' as release name jx to namespace jx-staging
6 Applying helm chart at '/tmp/jx-helm-apply-046268026/env' as release name jx to namespace jx-staging
7 verifying the helm requirements versions in dir: . using version stream URL: and git ref:
8 Deleting and cloning the Jenkins X versions repo https://github.com/jenkins-x/jenkins-x-versions
9 Cloning the Jenkins X versions repo https://github.com/jenkins-x/jenkins-x-versions.git with ref
10 refs/heads/master to /home/jenkins/.jx/jenkins-x-versions
11 Ignoring templates/.gitignore
12 Wrote chart values.yaml /tmp/jx-helm-apply-046268026/env/values.yaml generated from directory tree
  
```

# Viewing a Pipeline associated with Jenkins X

- Can use links in output
- Can use “jx console”
  - Brings up Jenkins instance running in jx namespace
  - Sign in
- Shows Blue Ocean interface with pipelines
  - Multi-branch pipelines (based off of Jenkinsfiles in Git branches)
- Select pipeline
- Select branch
- Explore details

The image contains four screenshots of the Jenkins X Blue Ocean interface:

- Welcome to Jenkins!**: Shows the Jenkins login screen with a placeholder for 'admin' and a note about an unsecured connection.
- Jenkins Pipelines**: Shows a list of pipelines. One pipeline, "brentlaster / environment-divemirror-production", is shown with a yellow sun icon, 1 passing test, and no branches. Another pipeline, "brentlaster / environment-divemirror-staging", is also listed with similar status.
- Jenkins Pipeline Details**: Shows the details for the "brentlaster / environment-divemirror-staging" pipeline. It lists one run (f7ec791) on the master branch, which completed 4m 38s ago.
- Update Environment Step**: Shows the Jenkins X command-line interface (CLI) executing a "jx step helm apply" command. The command runs a shell script to apply a Helm chart to the "environment-divemirror-staging" namespace, updating the environment.

# What is a Pull Request?

- GitHub model for contributing and gating changes into a repository.
- Proposed change is made in one place (source) and targeted for another place (destination)
- Source and destination can be:
  - Branch-to-branch in the same project
  - Project-to-project
- Source and destination are specified when you open (create) a new PR
- PR's can be automatically built for verification
- PR's can be automatically approved
- Notifications about changes from GitHub side are via Webhooks

# Webhooks

- Webhook is a way for an app/site (provider) to send a signal across the web when an event occurs
- Sends information (payload) to a URL that is “listening” (listener)
- Listener receives the webhook and performs a predefined action based on the information in the payload
- Webhooks are event-based while APIs are request based
- Jenkins X uses webhooks setup in Git repository to notify Jenkins X when event happens in Git that Jenkins X should react to (examples: push, PR merged, etc.)

# Webhooks in GitHub for Jenkins X

- Automatically created by Jenkins X
- Accessible via project->Settings->Webhooks
- Defaults to sending notifications to Jenkins URL

The screenshot shows the GitHub settings page for the repository `brentlaster / environment-daggersun-production`. The left sidebar has a red circle around the `Webhooks` link, with a red number '2' next to it. The main content area shows the `Webhooks` section with a red circle around the `Settings` button at the top right. Below it, a red circle highlights the `Edit` button for a webhook entry. The entry itself has a red circle around the URL `http://jenkins.jx.10.0.2.15.nip.io/github-webhook/ (all events)`.

- General DNS mapping so you don't have to have a custom domain
- Used by Jenkins X if you don't have a custom domain
- Maps <anything>[.-]<IP Address>.nip.io to corresponding <IP Address>
  - Dot notation: foo.127.0.0.1.nip.io maps to 127.0.0.1
  - Dash notation: foo-127-0-0-1.nip.io maps to 127.0.0.1
- For our case, <IP Address> corresponds to ip address of node
  - <http://jenkins.jx.10.0.2.15.nip.io> maps to 10.0.2.15
  - Minikube ip = 10.0.2.15
- However, our urls still aren't visible outside of virtual machine
- For that, we leverage a tool called "ultrahook"

- Simple command line tool that connects public webhook endpoints to private endpoints accessible from your system
- Solves problem of how to get webhook payloads locally to system that isn't exposed to outside web
- Allows for webhook requests to be bounced off of a name-spaced server and sent to your machine
- Installs ruby gem on system and run local command to have it detect requests to endpoint and send them on locally
- Can use different “subdomains” to direct different requests to
  - Example: <http://foo.jx1.ultrahook.com>, <http://bar.jx1.ultrahook.com>
  - For our purposes, subdomain = your GitHub userid

# Lab 3: Exploring Environments

# Quickstarts

- Basic applications you can use to start a project, instead of starting from nothing
- Command to create one
  - `jx create quickstart`
  - provides list of types to choose from
  - Can also filter with `-l` option (i.e. `jx create quickstart -l go`)
  - Can also filter with `-f http` on names
- Maintained at  
<https://github.com/jenkins-x-quickstarts>

```
android-quickstart
angular-io-quickstart
aspnet-app
dlang-http
golang-http
golang-http-from-jenkins-x-yml
jenkins-cwp-quickstart
jenkins-quickstart
node-http
node-http-watch-pipeline-activity
open-liberty
php-helloworld
python-http
rails-shopping-cart
react-quickstart
rollout-app
rust-http
scala-akka-http-quickstart
spring-boot-http-gradle
spring-boot-rest-prometheus
spring-boot-rest-prometheus-javall
spring-boot-watch-pipeline-activity
vertx-rest-prometheus
```

# What happens when a quickstart is created?

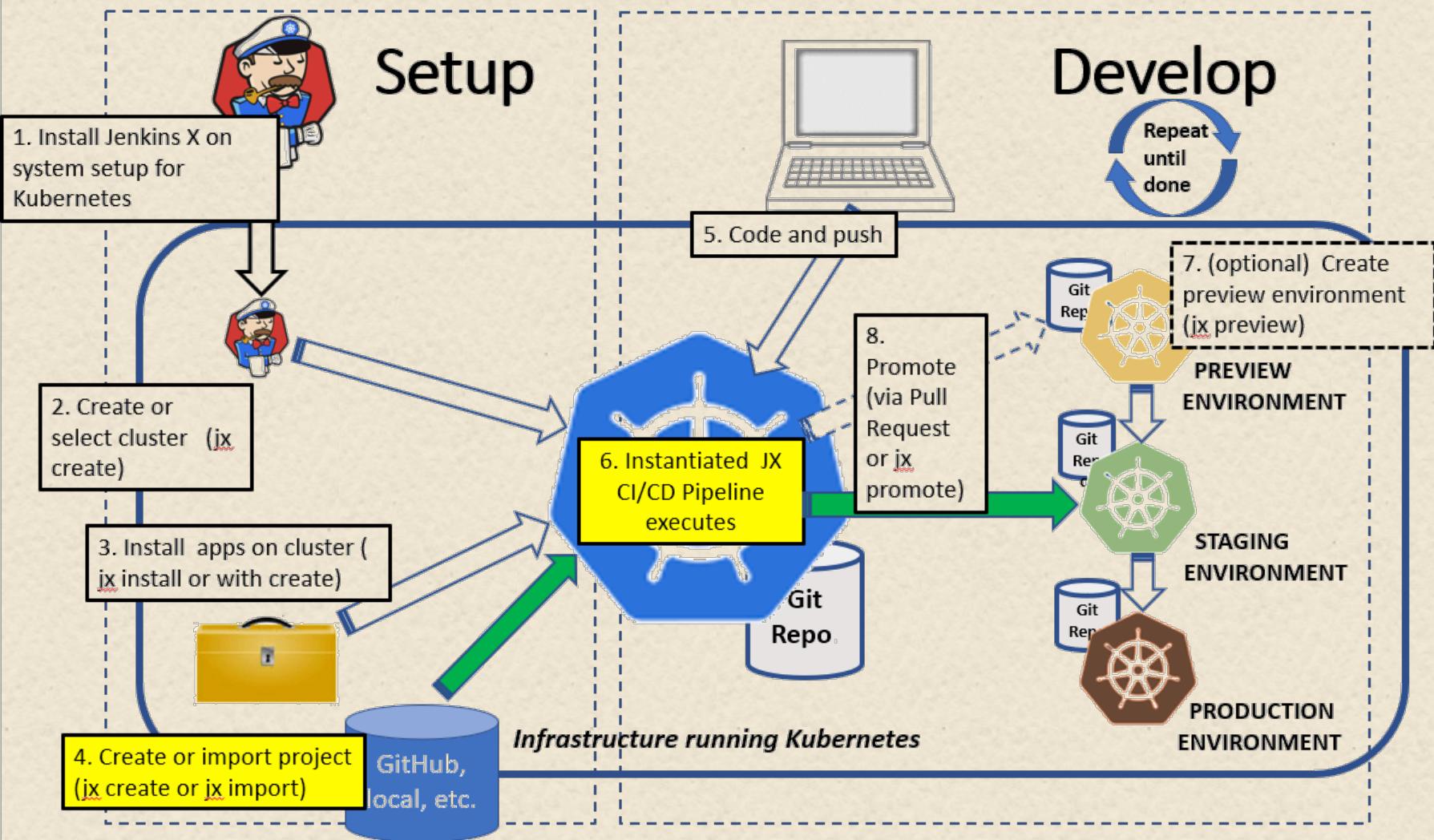
- User
  - Chooses the type of project to create
  - Provides a name
- Jenkins X automatically
  - Creates a new application from the quickstart project in a directory
  - Adds the source code into the local Git repository
  - Creates a remote Git repository (such as on GitHub)
  - Pushes the code to the remote Git repository
  - Adds necessary default versions of “infrastructure” files
    - » Dockerfile – to be able to build the application as a docker image
    - » Jenkinsfile – to implement the CI/CD pipeline
    - » Helm chart – to run the application in Kubernetes
  - Registers a webhook on the remote Git repository directed to your local Jenkins running in Kubernetes
  - Configures Jenkins to know about the Git repository
  - Triggers an initial pipeline run

# Quickstart and Environments

- By default, project from Quickstart doesn't belong to any permanent environments (is in dev context)
- Dev environment only contains services created by Jenkins X
- If successfully built through pipeline, and promotion policy is auto, will be moved to staging.
- To make project visible where promotion policy is manual (such as production), must promote it

# Lab 4: Creating a Quickstart project

# Overall Jenkins X Workflow



# Preview Environments

- “Extra” environment that Jenkins X can provide to allow users to get early/fast feedback on changes before they are merged into master branch
- Allows you to try changed versions of your work in your cluster first
  - Make sure they work as expected before promoting them
  - Deployed into cluster namespace/environment
- Jenkins X can create Preview environments for you
  - Automatically created for any Pull Requests
  - Additional ones can be created as needed using “`jx preview`” command
- When Preview environment is created
  - New environment of kind Preview is created
    - » New Kubernetes namespace is created
  - Visible via “`jx get environments`” and “`jx namespace`”
  - Pull Request is built as preview docker image
  - Image is deployed into preview environment via Helm chart
  - Comment is added to Pull Request to indicate preview application is ready for testing with link to the application

# Typical Change Process for App Update

- Happens after app has been created
- In local app directory, create new local branch
- In new branch, make code changes
- Commit changes locally and push to remote repository
- Create Pull Request from new branch to master
- Preview environment gets created for you with running new version of your app to “preview” functionality
- Merge pull request

# Continuous Integration with Pull Requests

61

- If can be merged automatically, Jenkins build will be kicked off
- Jenkins multibranch pipeline will show PR like branch
- If Jenkins build successful (and all checks successful), “Merge” button enabled

The screenshot displays three views related to a GitHub pull request and its Jenkins pipeline status:

- GitHub Pull Request View:** Shows a summary of CI status. It indicates "Some checks haven't completed yet" (1 pending and 1 successful) and "continuous-integration/jenkins/pr-merge" is pending. It also shows "continuous-integration/jenkins/branch" as successful. A note states "This branch has no conflicts with the base branch". A "Merge pull request" button is present.
- Jenkins Pipeline View:** Shows the Jenkins pipeline interface for the "lab4" pipeline. It lists a single pull request "PR-1" with details: Status (S) Success, Work (W) In Progress, Name (Name) PR-1, Last Success (Last Success) 7 hr 3 min - #1, Last Failure (Last Failure) N/A, Last Duration (Last Duration) 1 min 0 sec. Icons for S, M, L are shown. RSS feeds for all, failures, and latest builds are available.
- GitHub Pull Request View (Success):** Similar to the first view, but "continuous-integration/jenkins/pr-merge" is now successful. The "Merge pull request" button is now green and enabled.

# Completing the PR

- After checks have passed, you can click on “Merge” button and confirm (automated in Jenkins X)
- After a moment, results will show in GitHub (and be available to Jenkins X)
- In Jenkins, PR will no longer show

Add more commits by pushing to the `test` branch on `gwstudent2/filedump`.

Merge pull request #2 from `gwstudent2/test`

Test

Confirm merge Cancel

**gwstudent2** merged commit `234cccb` into `master` 2 minutes ago

2 checks passed

Pull request successfully merged and closed

You're all set—the `test` branch can be safely deleted.

Delete branch

Very simple repo to use in Intro to CI course

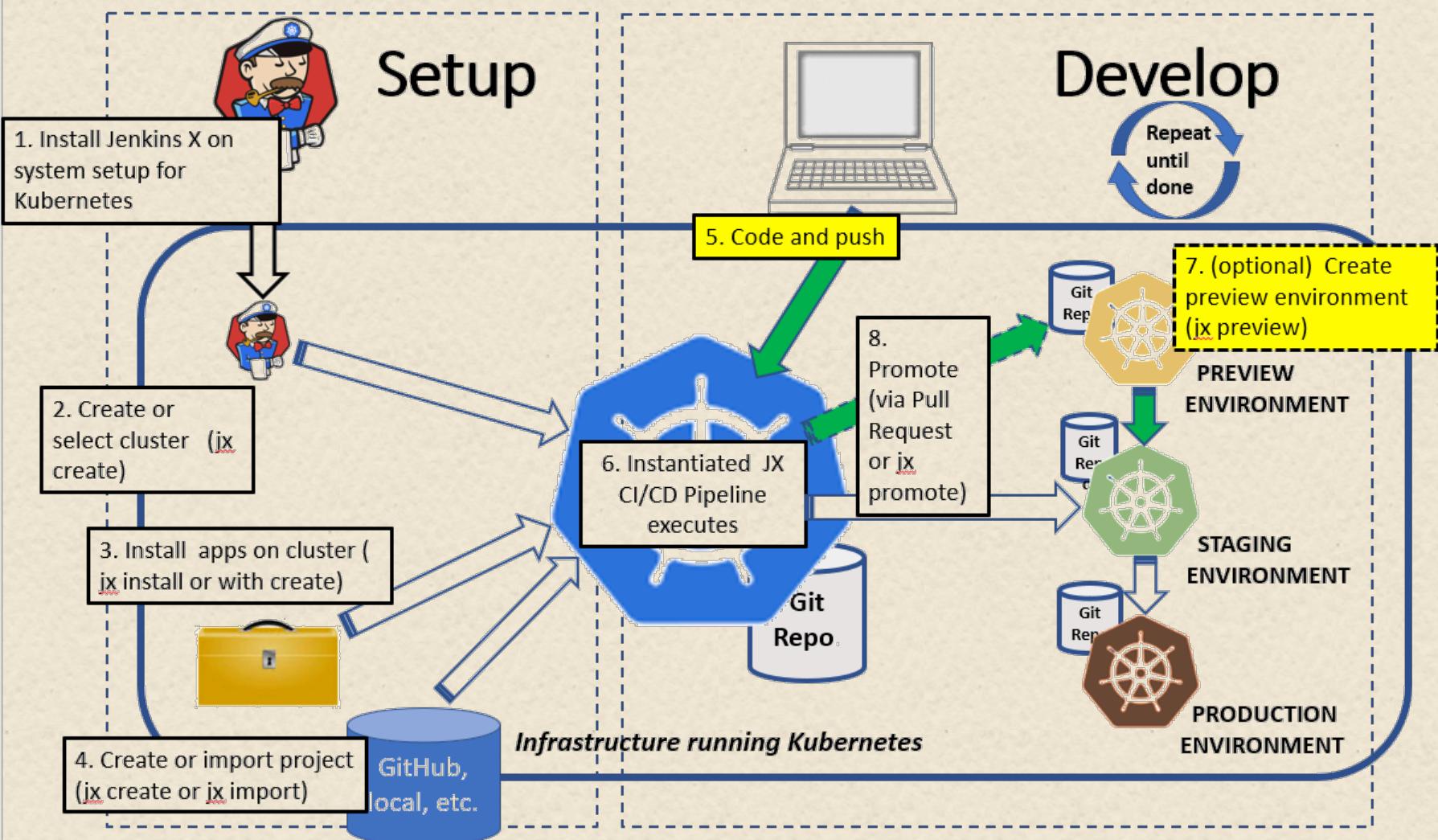
Branches (2)		Pull Requests (0)	
S	W	Name	Last Success
		<a href="#">master</a>	11 hr - #1
		<a href="#">test</a>	N/A

Icon: S M L

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

# Lab 5: Creating a Preview environment

# Jenkins X Overall Workflow



# Typical Project Promotion Workflow (Auto) 65

Create a branch off of master

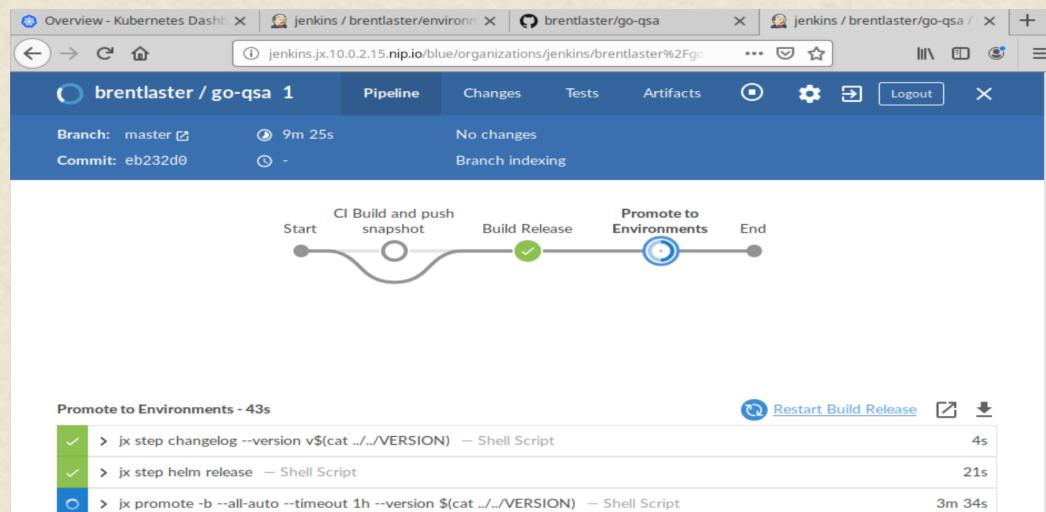
Make a change and push it to remote Git repository

Since remote Git repository has webhook to local Jenkins, kicks off, multibranch pipeline is kicked off

If promotion attribute for environment is set to “auto”, project is automatically built, PR created and merged and product promoted

App runs in container in our staging namespace in cluster

```
diyuser3@training1:~$ jx get env
NAME      LABEL      KIND      PROMOTE   NAMESPACE      ORDER CLUSTER SOURCE
REF PR
dev       Development Development Never    jx                0
staging   Staging     Permanent   Auto     jx-staging     100      https://github.com/brentlaster/environment-divemirror-staging.git
production Production Permanent Manual   jx-production 200      https://github.com/brentlaster/environment-divemirror-production.git
```



chore: qs3 to 0.0.3 #1  
Merged brentlaster merged 1 commit into bclasterorg:master from brentlaster:promote-qs3-0.0.3 13 hours ago  
Conversation 0 Commits 1 Checks 0 Files changed 2  
brentlaster commented 13 hours ago  
chore: Promote qs3 to version 0.0.3  
chore: Promote qs3 to version 0.0.3 07faf98  
brentlaster merged commit 49749cb into bclasterorg:master 13 hours ago 1 check passed

```
BUILT RELEASE          17m55s  1m13s Succeeded
Promote to Environments 16m58s  16m55s Succeeded
Promote: staging        16m13s  15m58s Succeeded
PullRequest             16m13s  3m7s Succeeded PullRequest: https://github.com/bclasterorg/environment-slaveroan-staging/pull/1 Merge SHA: 49749cb78011baa2268ae0c16ce4494bc7798204
Update                  13m6s   12m51s Succeeded Status: Success at: http://jenkins.jx.10.0.2.15.nip.io/job/bclasterorg/job/environment-slaveroan-staging/job/master/2/display/redirect
Promoted                 13m6s   12m51s Succeeded Application is at: http://qs3.jx-staging.10.0.2.15.nip.io
```

# Typical Project Promotion Workflow (Manual)

- Create a branch off of master
- Make a change and push it to remote Git repository. You are presented with a link to open a PR.
- Go to Git and create PR.
- Jenkins X runs a build to make sure it is ok to merge, creates a Preview Environment and informs you in PR about it.
- Jenkins will show build for Preview Environment.
- Can see Preview Environment via jx preview.
- Promote via “jx promote” command

```
diyuser3@training1:~$ jx get env
NAME      LABEL     KIND      PROMOTE   NAMESPACE    ORDER CLUSTER SOURCE
REF PR
dev       Development Development Never    jx           0
staging   Staging    Permanent  Auto     jx-staging   100   https://github.com/brentlaster/environment-divemirror-staging.git
production Production Permanent  Manual   jx-production 200
it         it         Permanent  Manual   jx-production 200   https://github.com/brentlaster/environment-divemirror-production.git
```

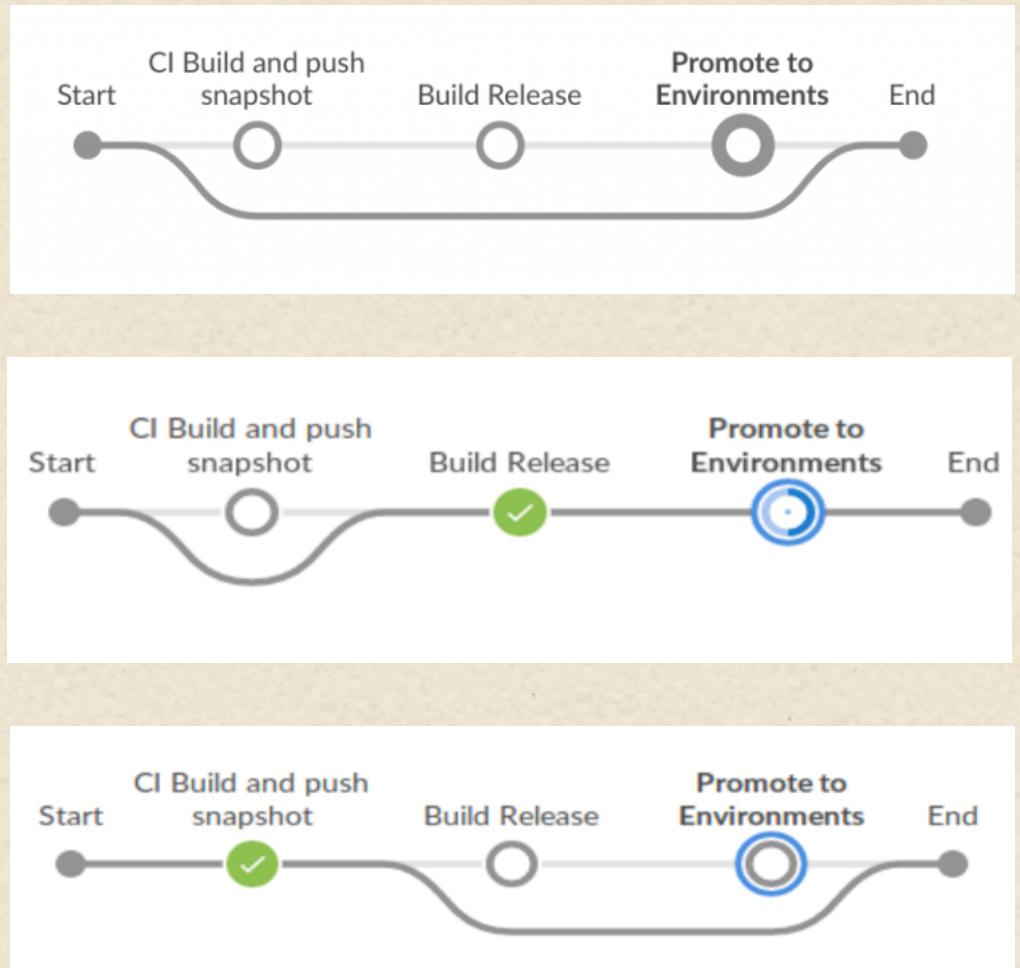
The screenshots illustrate the Jenkins X manual promotion workflow:

- Terminal output showing environment details (NAME, LABEL, KIND, PROMOTE, NAMESPACE, ORDER, CLUSTER, SOURCE) for dev, staging, production, and it environments.
- Github pull request page for 'main.go' comparing 'base: master' and 'compare: pr2'. It shows a 'Table to merge' button and a note that the branches can be automatically merged.
- Jenkins X pipeline status page for PR-1. It shows the 'Pipeline' tab with a green 'CI Build and push snapshot' step completed. The commit hash is 8d36f31, and it was pushed 42 minutes ago.
- Jenkins X preview environment page for 'main.go'. It shows a green 'All checks have passed' status and a note that this branch has no conflicts with the base branch.
- Github pull request page showing the PR has been merged into 'master'. It includes a comment from brentlaster and a note that the PR built and is available in a preview environment.
- Jenkins X pipeline status page for PR-1 showing the 'Promote to Environments' step completed. It includes a note that the branch can be safely deleted.

```
diyuser3@training1:~$ jx get preview
PULL REQUEST          NAMESPACE          APPLICATION
https://github.com/brentlaster/go-qsa/pull/1 jx-brentlaster-go-qsa-pr-1 http://go-qsa.jx-brentlaster-go-qsa-pr-1.10.0.2.15.nip.io
```

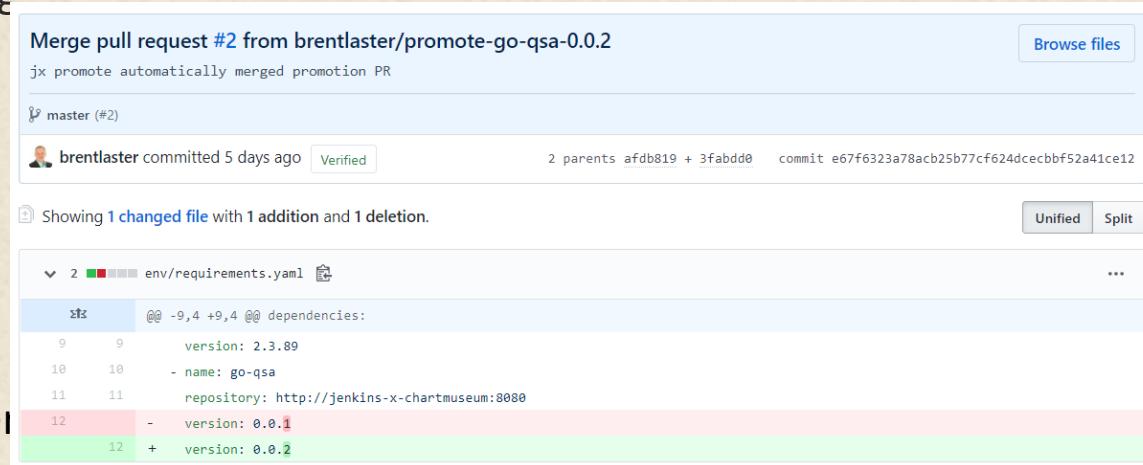
# Stages in PR Pipeline

- CI Build and push snapshot
- Build Release
  - Runs tests
  - Creates Docker image
  - Pushes it to Docker registry
- Promote to Environments
  - Creates release information in repository
    - » Archived source code
    - » Pull request history
  - Creates Helm chart and pushes it to Chartmuseum
  - Accessible to Monocular for pulling from Helm



# Jenkins X versioning of apps

- Jenkins X uses semantic versioning
  - Format - **Major.Minor.Patch-<label>**
  - Increment Major version for incompatible API changes.
  - Increment Minor version for adding functionality in a backwards-compatible manner
  - Increment Patch version for backwards-compatible bug fixes.
  - Labels for pre-release or build metadata can be added as an extension to the version number (in “<label>”).
- Stored in env/requirements.yaml
- Accessible via “jx get version”



A screenshot of a GitHub pull request titled "Merge pull request #2 from brentlaster/promote-go-qsa-0.0.2". The commit message is "jx promote automatically merged promotion PR". The author is brentlaster, committed 5 days ago, and the commit hash is e67f6323a78acb25b77cf624dcecbbf52a41ce12. The pull request has 2 parents: afdb819 + 3fabdd0. The file "env/requirements.yaml" is shown with a diff:

```
diff --git a/env/requirements.yaml b/env/requirements.yaml
@@ -9,4 +9,4 @@ dependencies:
 9   9     version: 2.3.89
 10  10   - name: go-qsa
 11  11     repository: http://jenkins-x-chartmuseum:8080
 12  12   - version: 0.0.1
 12  12   + version: 0.0.2
```

- Jenkins X uses git tags to calculate the next release version
- Will automatically create a released version on every merge to master

# Other jx commands

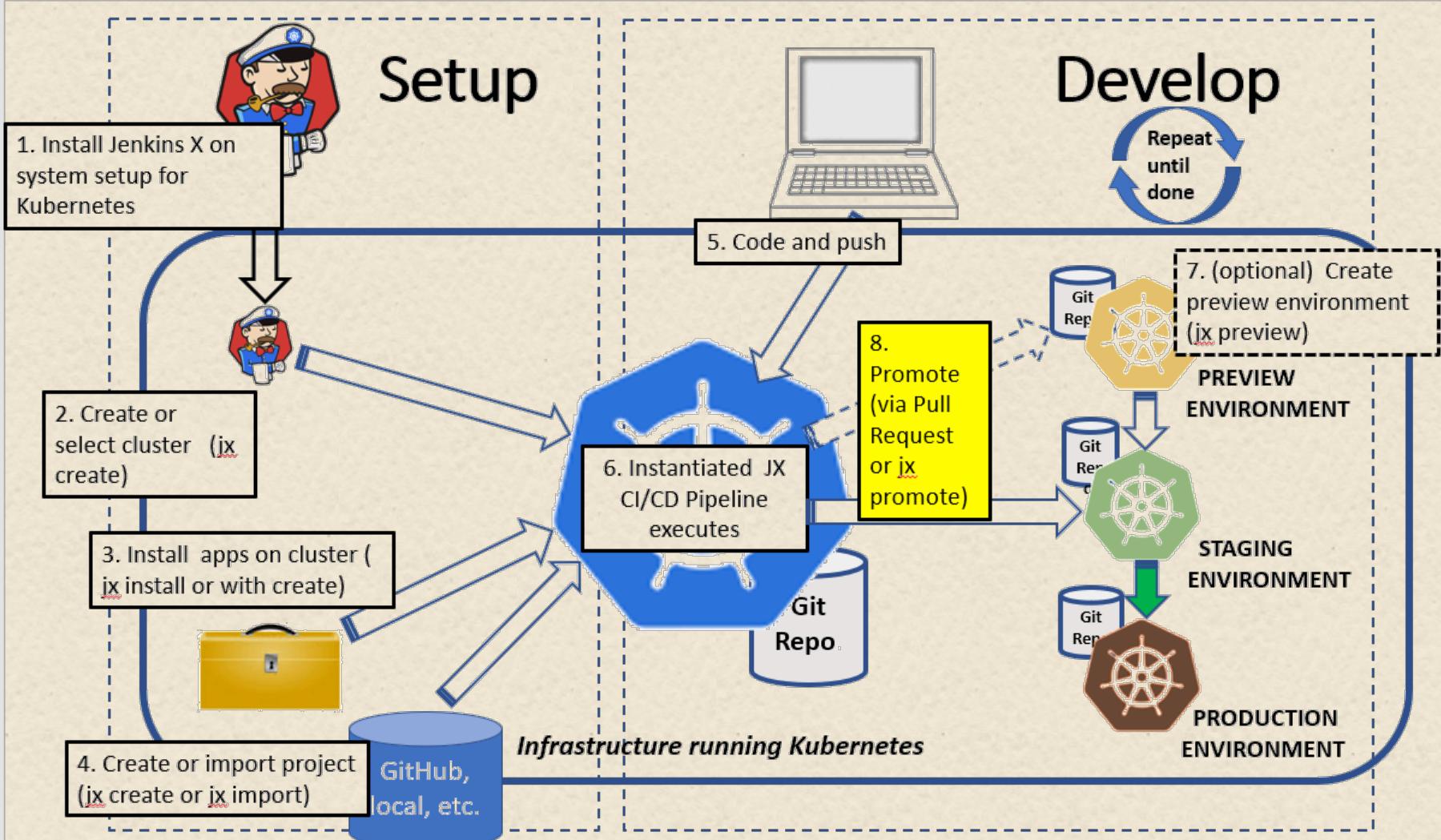
- `jx open <app name> -e <level>`
- `jx promote <app name> -e <level>`

Or

```
jx promote --version < version number > -e <level>
```

# Lab 6: Promoting to Prod

# Jenkins X Overall Workflow



# Other Topics to Explore

- `jx boot` – new consistent way to install, configure, upgrade Jenkins X using GitOps
- Devpods – pods that Jenkins X can create for developers to use with terminals, shells, IDEs with same tools as CI/CD pipeline – for use before committing to Git
- Serverless Jenkins X – Jenkins X without Jenkins!

# That's all - thanks!

## Professional Git 1st Edition

by Brent Laster (Author)

★★★★★ 3 customer reviews

[Look inside](#)

