Open in app ↗

Medium Search Write

Curious Devs Corner

Member-only story

Argo CD Blue-Green Deployments in Kubernetes: Full Guide

How to roll out blue-green deployments with Argo CD and Kubernetes with Label Switching

Kirshi Yin Follow

9 min read · Nov 4, 2025

13

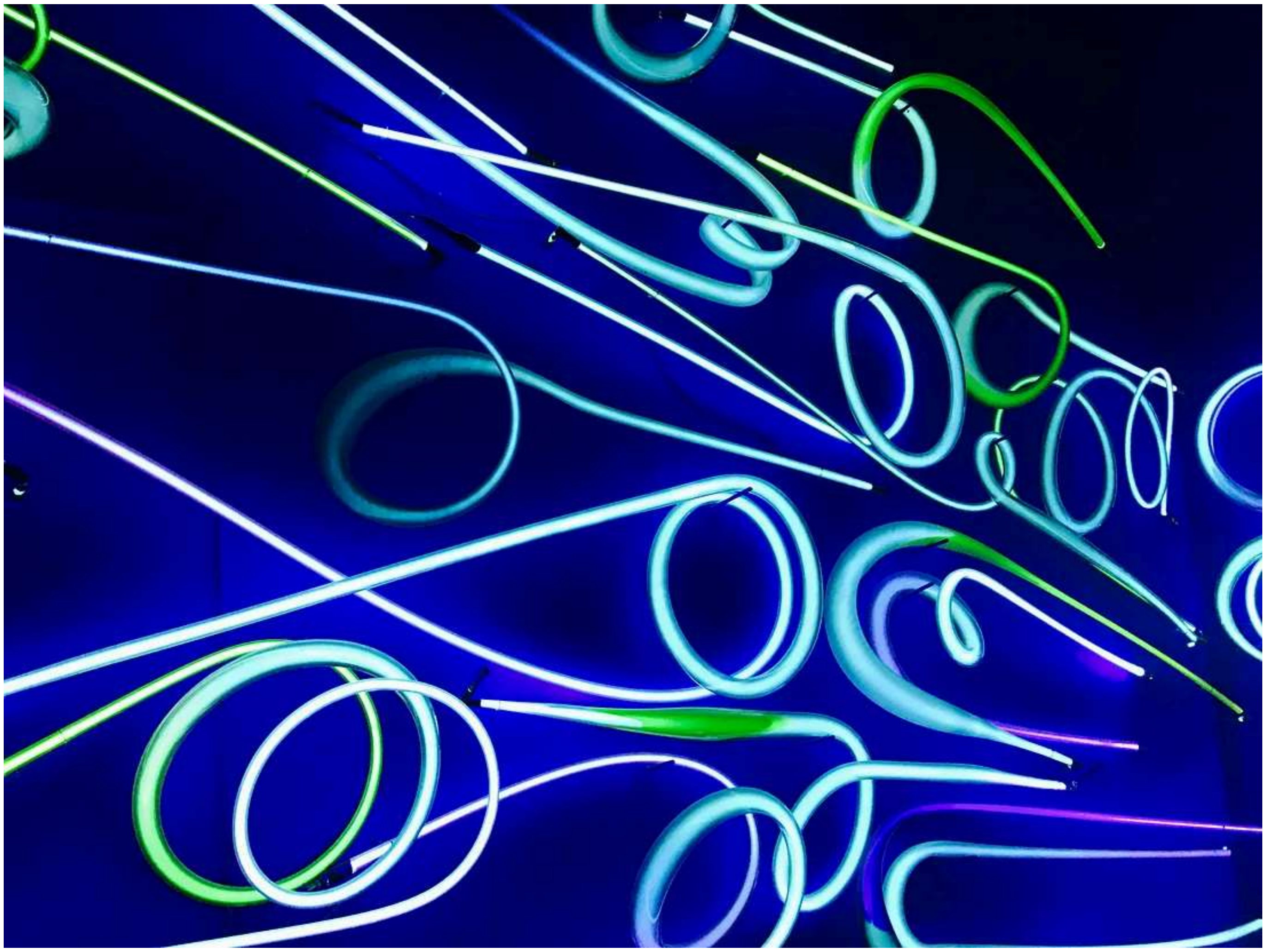


Photo by Lynn Kintziger on Unsplash

Introduction

Deployment strategies help teams release updates safely and reduce downtime. They define how new versions of an application replace old ones in a controlled way. Among these strategies, blue-green deployment is a popular choice because it keeps two separate environments — one active and one idle — allowing instant switchovers and quick rollback if needed.

In this article, you will learn how to implement blue-green deployments using only Argo CD and native Kubernetes resources. You will also explore alternative approaches, including using Argo Rollouts, the App of Apps pattern, and Ingress switching, so you can choose the method that fits your workflow best.

Prerequisites:

- Locally installed ArgoCD
- Local Kubernetes Cluster
- Basic ArgoCD and Kubernetes knowledge

*This article is an extract of my ebook, **Master GitOps with ArgoCD**. If you want to deepen your knowledge, you can check it out in my Gumroad store.*

If you want to learn about further Kubernetes deployment strategies, I have an article here about this topic.

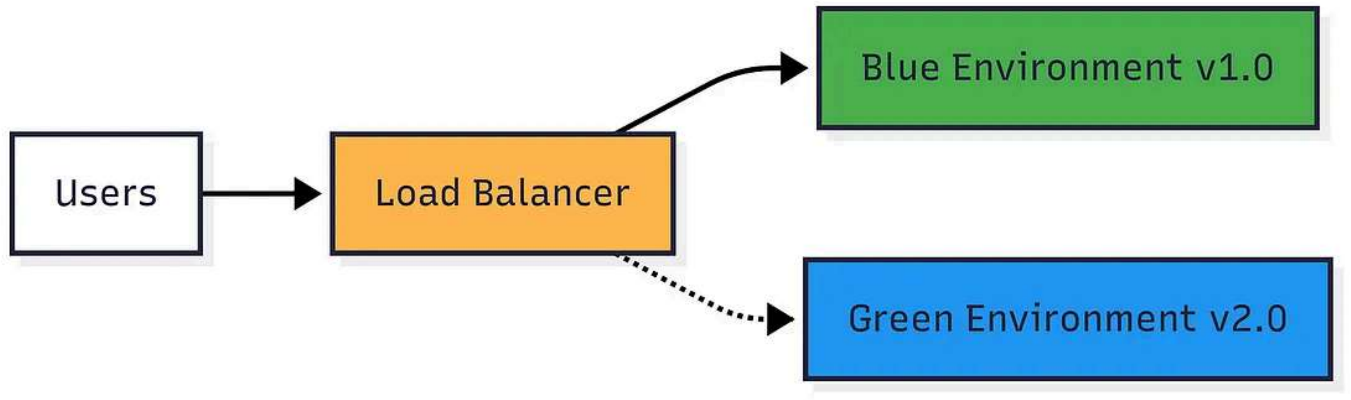
Let’s get started!

What is Blue-Green Deployment?

The Blue-green deployment strategy eliminates downtime during application updates. It uses two production environments:

- **Blue Environment**: Currently serves live user traffic
- **Green Environment**: Runs the new version for testing

When the green environment passes all tests, you switch traffic from blue to green. The old blue environment stays ready for instant rollback.



Argo CD and Blue-Green Deployments

Argo CD doesn't include built-in blue-green deployment features. However, we can implement it using native Kubernetes resources and GitOps principles.

You'll follow this strategy:

1. Deploy two separate Kubernetes Deployments (blue and green).
2. Use a single Service with labels pointing to the active deployment.
3. Switch traffic by updating Service selectors in Git.
4. ArgoCD syncs the changes automatically.

This approach uses a pure GitOps workflow. Every change goes through Git commits and Argo CD sync.

Hands-on: Blue-Green App Deployment Using Manual Service Selector Switching

In this exercise, you'll deploy a simple web application using **blue-green deployment**. We'll use **nginx** with two versions — blue and green — so you can visually see which environment serves traffic. This method uses **native**

Kubernetes resources and **Argo CD** to manage deployments and switch traffic safely.

1. Project Structure

Create a folder for the project and organize files like this:

```
├── blue-green-app
│   ├── app-service.yml
│   ├── blue-configmap.yml
│   ├── blue-deployment.yml
│   ├── green-configmap.yml
│   └── green-deployment.yml
```

- `ConfigMaps` hold the HTML content for each version.
- `Deployments` create pods for each environment.
- `Service` controls traffic routing.

2. Create ConfigMaps

This is the content of the blue ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-blue
  namespace: default
data:
  index.html: |
    <!DOCTYPE html>
    <html>
    <head>
        <title>Blue-Green Demo</title>
        <style>
            body {
                font-family: Arial;
                text-align: center;
                padding: 50px;
                background-color: #f0f0f0;
                color: #333;
                border: 3px solid #333;
            }
            .version-box {
                background-color: #e8e8e8;
                padding: 20px;
                margin: 20px;
                border-radius: 8px;
                border: 2px dashed #666;
            }
        </style>
    </head>
    <body>
        <h1> BLUE ENVIRONMENT</h1>
        <div class="version-box">
            <h2>Version 1.0.0</h2>
            <p><strong>Environment:</strong> BLUE</p>
            <p><strong>Release Date:</strong> January 2025</p>
            <p><strong>Status:</strong> PRODUCTION</p>
            <p><strong>Features:</strong> Basic functionality, stable release</p
        </div>
        <p><strong>Pod Label:</strong> version=blue</p>
        <p><strong>Instance ID:</strong> BLUE-PROD-001</p>
    </body>
    </html>
```

This is the content of the green ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config-green
  namespace: default
data:
  index.html: |
    <!DOCTYPE html>
    <html>
    <head>
```

```
        <title>Blue-Green Demo</title>
        <style>
            body {
                font-family: Arial;
                text-align: center;
                padding: 50px;
                background-color: #f0f0f0;
                color: #333;
                border: 3px solid #333;
            }
            .version-box {
                background-color: #e8e8e8;
                padding: 20px;
                margin: 20px;
                border-radius: 8px;
                border: 2px dashed #666;
            }
            .new-feature {
                background-color: #fffacd;
                padding: 10px;
                margin: 10px 0;
                border-left: 4px solid #ffd700;
            }
        </style>
    </head>
    <body>
        <h1> GREEN ENVIRONMENT</h1>
        <div class="version-box">
            <h2>Version 2.0.0</h2>
            <p><strong>Environment:</strong> GREEN</p>
            <p><strong>Release Date:</strong> March 2025</p>
            <p><strong>Status:</strong> TESTING</p>
            <p><strong>Features:</strong> Enhanced UI, new API endpoints</p>
        </div>
        <div class="new-feature">
            <h3>NEW FEATURES</h3>
            <p> Improved performance</p>
            <p>Better error handling</p>
            <p>Enhanced security</p>
        </div>
        <p><strong>Pod Label:</strong> version=green</p>
        <p><strong>Instance ID:</strong> GREEN-TEST-002</p>
    </body>
    </html>
```

These ConfigMaps store static HTML pages. Each environment has a unique visual style to make it easy to see which one is serving traffic. The blue and green backgrounds are intentionally swapped to highlight the switch.

3. Create Deployments

Create the `blue-deployment.yaml`:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-blue
  namespace: default
  labels:
    app: demo-app
    version: blue
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo-app
      version: blue
  template:
    metadata:
      labels:
        app: demo-app
        version: blue
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
        volumeMounts:
        - name: app-content
          mountPath: /usr/share/nginx/html
        resources:
          requests:
            memory: "64Mi"
            cpu: "50m"
          limits:
            memory: "128Mi"
            cpu: "100m"
      volumes:
```

```
      - name: app-content
        configMap:
          name: app-config-blue
```

Explanation:

- Runs 3 pods for high availability.
- Mounts the blue ConfigMap as HTML content.
- Labels identify this deployment as the blue environment.

Create the `green-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-green
  namespace: default
  labels:
    app: demo-app
    version: green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo-app
      version: green
  template:
    metadata:
      labels:
        app: demo-app
        version: green
    spec:
      containers:
      - name: nginx
        image: nginx:1.21
        ports:
        - containerPort: 80
        volumeMounts:
          - name: app-content
            mountPath: /usr/share/nginx/html
          resources:
            requests:
              memory: "64Mi"
              cpu: "50m"
            limits:
              memory: "128Mi"
              cpu: "100m"
        volumes:
        - name: app-content
          configMap:
            name: app-config-green
```

Explanation:

- Green deployment mirrors blue for consistency.
- Labels indicate the green environment.
- Ready for testing without affecting the live blue environment.

4. Create the Service

Create the `app-service.yaml`:

```
apiVersion: v1
kind: Service
metadata:
  name: app-service
  namespace: default
spec:
  selector:
    app: demo-app
    version: blue  # Initially points to blue
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  type: LoadBalancer
```

Explanation:

- Routes traffic to pods matching the selector.
- Initially directs traffic to the blue environment.
- Updating `version` to `green` will instantly switch traffic.

5. Push to Git and Create Argo CD App

Create a Git repo for this project and push the files:

```
# Initialize Git repository
git init
git add .
git commit -m "Initial blue-green deployment setup"

# Push to your remote repository
git remote add origin https://github.com/your-username/blue-green-demo
git push -u origin main
```

Create an Argo CD application manifest:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: blue-green-demo
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/your-user/argocd-blue-green-deployment
    path: blue-green-app
    targetRevision: HEAD
  destination:
    server: https://kubernetes.default.svc
    namespace: default
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

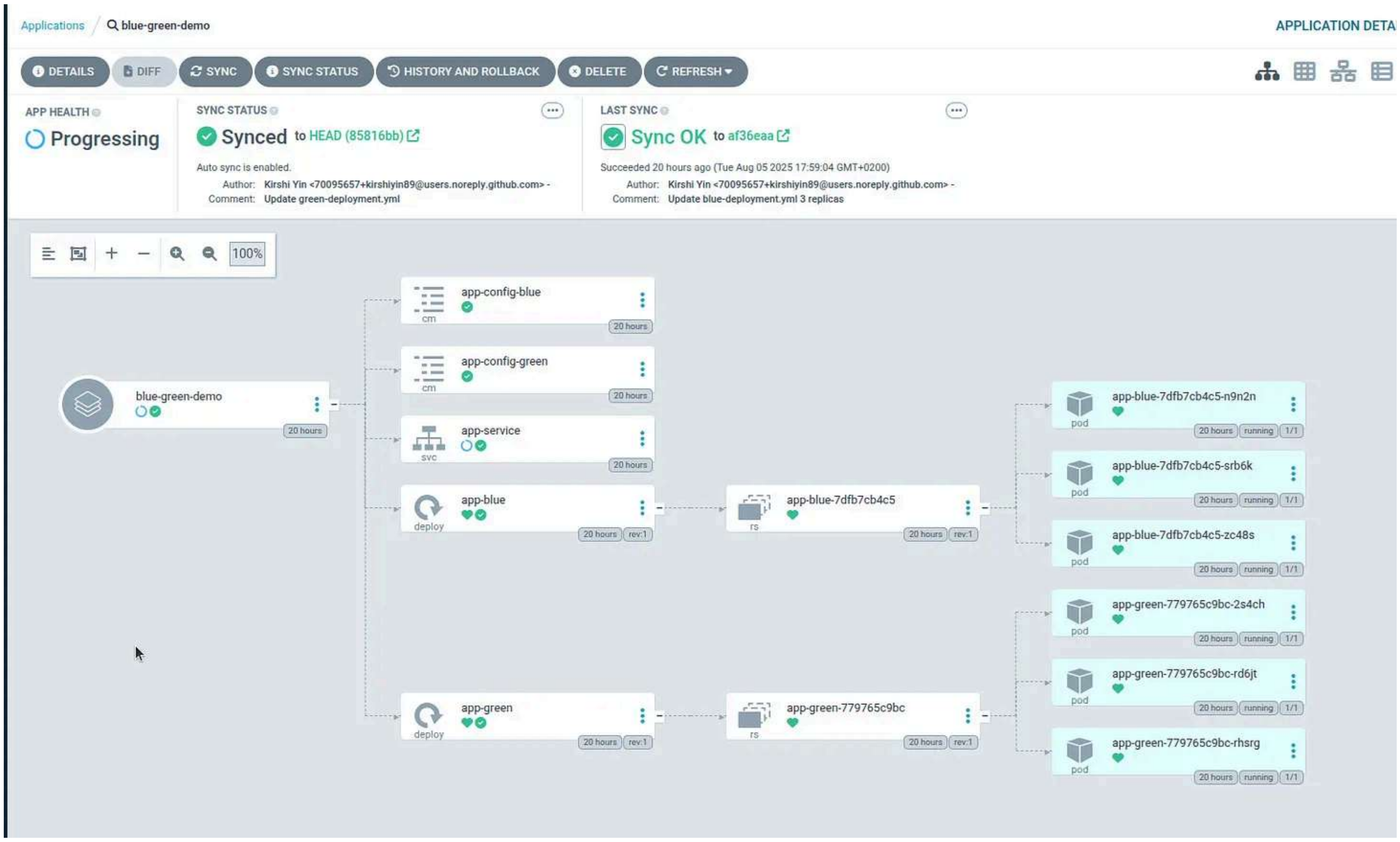
Deploy the app:

```
kubectl apply -f argocd-app.yaml
```

Explanation:

- Argo CD automatically syncs the repository with the cluster.
- Both blue and green deployments exist, but only blue serves traffic initially.

Check the status in the Argo CD UI:



APP HEALTH
Progressing

SYNC STATUS
Synced to HEAD (85816bb)
Auto sync is enabled.
Author: Kirshi Yin <70095657+kirshiyin89@users.noreply.github.com> -
Comment: Update green-deployment.yml

LAST SYNC
Sync OK to af36eaa
Succeeded 20 hours ago (Tue Aug 05 2025 17:59:04 GMT+0200)
Author: Kirshi Yin <70095657+kirshiyin89@users.noreply.github.com> -
Comment: Update blue-deployment.yml 3 replicas

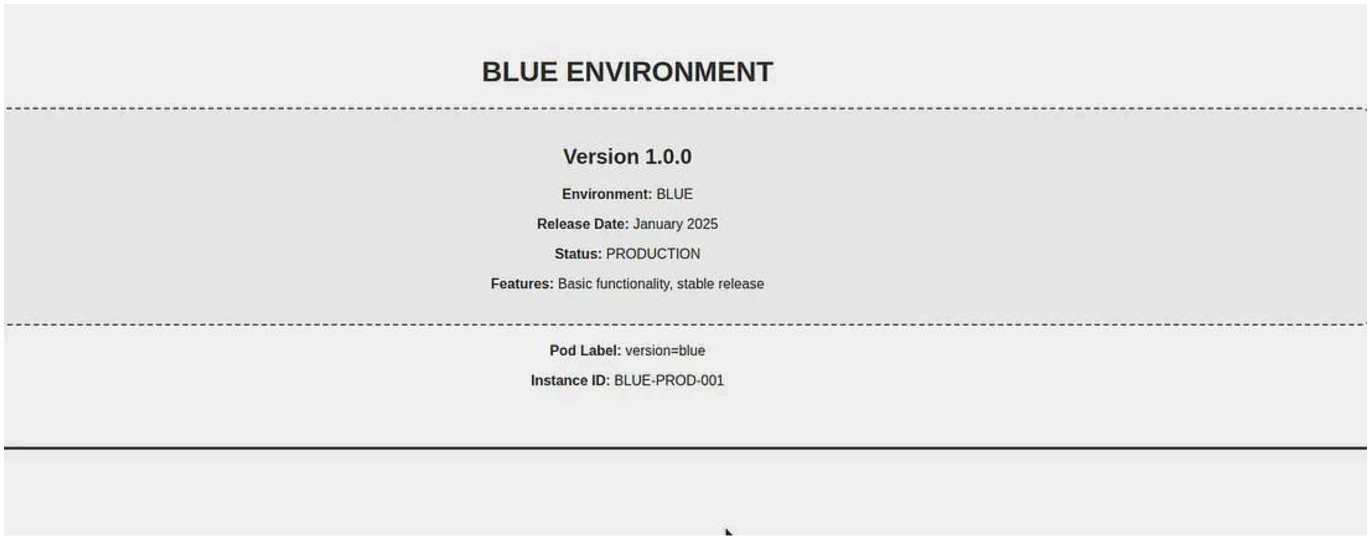
You should see both blue and green deployments running, but only the blue one receives traffic.

6. Verify and Switch Traffic

Use port-forwarding to access the blue app:

```
kubectl port-forward svc/app-service 8080:80
```

Visit http://localhost:8080. You should see the blue environment page.



BLUE ENVIRONMENT

Version 1.0.0

Environment: BLUE

Release Date: January 2025

Status: PRODUCTION

Features: Basic functionality, stable release

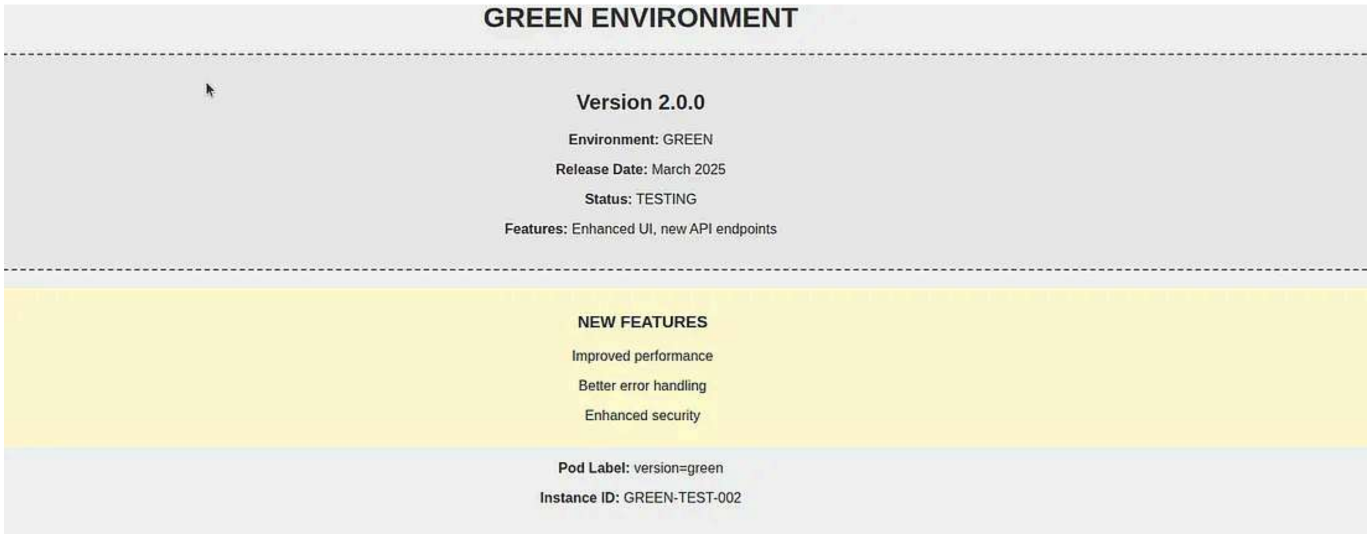
Pod Label: version=blue

Instance ID: BLUE-PROD-001

Now, let's test the green environment by port-forwarding the traffic:

```
kubectl port-forward deployment/app-green 8081:80
```

Check the page shows the green version:



GREEN ENVIRONMENT

Version 2.0.0

Environment: GREEN

Release Date: March 2025

Status: TESTING

Features: Enhanced UI, new API endpoints

NEW FEATURES

Improved performance

Better error handling

Enhanced security

Pod Label: version=green

Instance ID: GREEN-TEST-002

We verified that it's working fine. We can now switch the traffic to green.

Update the Service selector to point to green. Edit the `app-service.yml` and update the label to green:

```
apiVersion: v1
kind: Service
metadata:
  name: app-service
  namespace: default
spec:
  selector:
    app: demo-app
    version: green  # Changed from blue to green
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  type: LoadBalancer
```

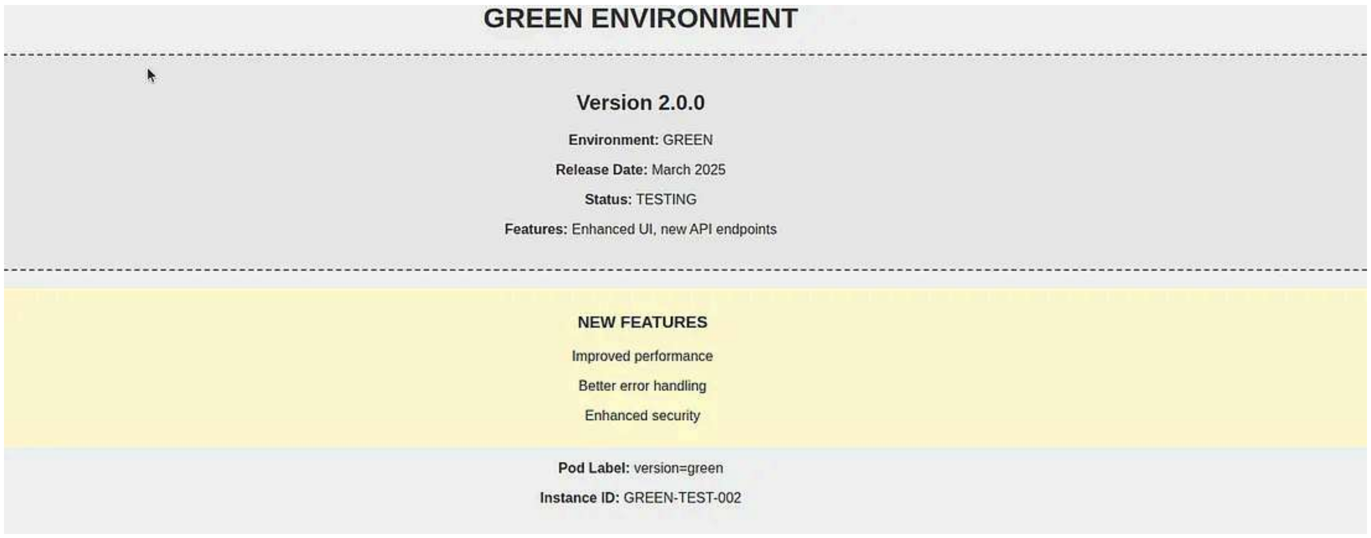
Commit and push the change. Argo CD will pick it automatically.

Check the service selector:

```
kubectl get service app-service -o yaml | grep -A2 selector
```

```
{"apiVersion":"v1","kind":"Service","metadata":{"annotations":{"argocd.argoproj.
  creationTimestamp: "2025-08-05T15:55:31Z"
  name: app-service
--
  selector:
    app: demo-app
    version: green
```

When you refresh the page on http://localhost:8080, you should see the green version:


GREEN ENVIRONMENT

Version 2.0.0

Environment: GREEN

Release Date: March 2025

Status: TESTING

Features: Enhanced UI, new API endpoints

NEW FEATURES

Improved performance

Better error handling

Enhanced security

Pod Label: version=green

Instance ID: GREEN-TEST-002

The switch happened instantly with zero downtime.

7. Scale Down Old Environment

Once the green environment works as expected, scale down blue:

Edit `blue-deployment.yaml` :

```
spec:
 replicas: 0
```

Commit and push. Argo CD will scale down the old pods automatically.

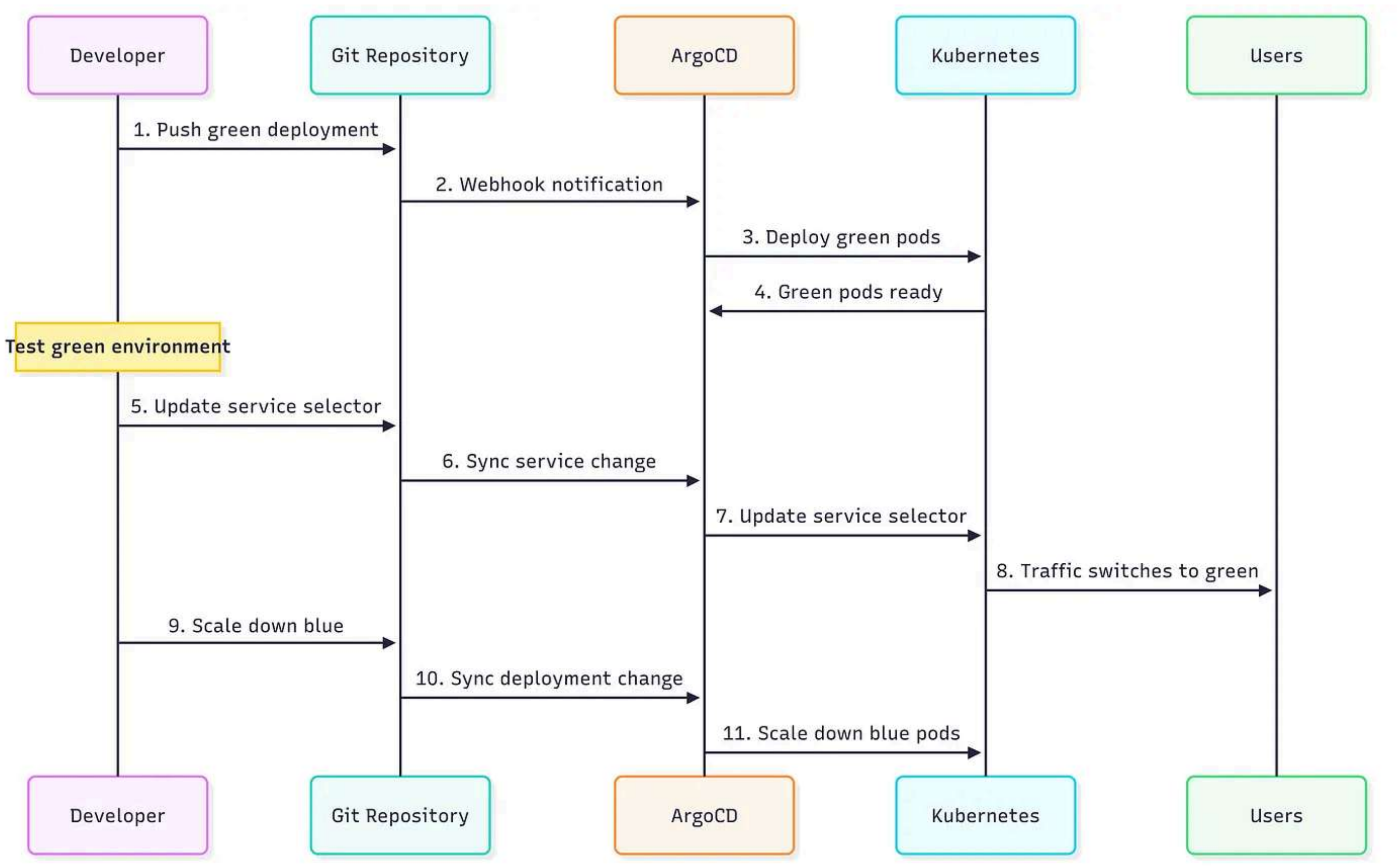
Explanation:

- Frees cluster resources.

- Keeps only the active environment running.

Blue-Green Deployment Flow

The diagram below helps you summarize the steps you took to switch from blue to green:



Rollback Strategy

If issues occur after switching to green, rollback is simple:

Quick Rollback (Service Switch):

```
# Revert service to blue
git revert <commit-hash-of-service-change>
git push origin main
```

Full Rollback (Scale Up Blue)

```
# Scale up blue deployment
# Edit blue-deployment.yaml: replicas: 0 → replicas: 3
git add blue-deployment.yaml
git commit -m "Emergency rollback: scale up blue environment"
git push origin main
```

ArgoCD syncs the changes within seconds. Traffic switches back to the stable blue environment.

Alternative Blue-Green Deployment Approaches

The method we just used is simple and beginner-friendly. It requires no extra tools and relies entirely on Kubernetes and Argo CD. You control traffic with a single Service, and the concepts — Deployments, Services, and labels — are ones you already know.

Still, as your applications grow in complexity, other blue-green approaches can offer more flexibility and control. Here are some alternatives worth knowing.

Dual Service with Ingress Switching

One common approach is to deploy both blue and green environments with separate Services, then use an Ingress controller or LoadBalancer to manage traffic. Switching environments is as simple as updating the Ingress rules or LoadBalancer configuration. Argo CD can manage these changes automatically.

Benefits:

- Enables more advanced traffic management.
- Allows weighted traffic splitting or A/B testing.
- Suitable for complex routing scenarios across multiple services.

Drawbacks:

- Requires additional setup and configuration.
- Needs an Ingress controller or external LoadBalancer.
- Slightly higher resource usage since both Services run continuously.

This method is useful when you want fine-grained control over how traffic flows between environments, especially in multi-service applications or testing new features with a subset of users.

Argo CD App-of-Apps Pattern

Another approach leverages Argo CD's **App-of-Apps** pattern. Here, you create separate Argo CD applications for the blue and green environments. You can enable or disable each application to control which environment is active. Using Argo CD sync waves, you can orchestrate deployments across multiple services in a controlled order.

Benefits:

- Makes full use of Argo CD’s application management features.
- Ideal for multi-service or microservice architectures.
- Provides granular control over sync policies and health checks.
- Supports progressive delivery patterns.

Drawbacks:

- Requires a deeper understanding of Argo CD concepts.
- It can become complex with many applications and microservices.
- Less intuitive for beginners.

This approach is best if you already manage multiple services with Argo CD and want to incorporate blue-green deployment as part of your GitOps workflow.

Argo Rollouts

For teams looking for a purpose-built solution, **Argo Rollouts** extends Kubernetes with advanced deployment strategies. You install the Rollouts controller alongside Argo CD and use Rollout resources instead of standard Deployments. Rollouts can automate blue-green switching, health checks, canary releases, and even metrics-based promotion.

Benefits:

- Designed for advanced deployment strategies.

- Automates promotion and rollback of environments.
- Integrates with monitoring and metrics for safer releases.
- Production-ready with enterprise features.

Drawbacks:

- Adds another tool to install and maintain.
- Has a learning curve for Rollouts concepts.
- It can be overkill for simple applications.

Conclusion

In this tutorial, you learned how to perform blue-green deployment with ArgoCD and Kubernetes.

You also got an overview of alternative approaches, so you can make the right choice. I encourage you to try the other methods at your leisure.

Experimenting with different strategies will deepen your understanding and help you handle more complex release scenarios confidently.

References

The code for this code is in my GutHub repository:

GitHub - kirshiyin89/argocd-blue-green-deployment: Argocd blue green deployment with kubernetes...

Argocd blue green deployment with kubernetes native resources - kirshiyin89/argocd-blue-green-deployment

github.com

For more information on **DevOps** topics, check you my **Gumroad** store, or my **Medium** publication.

- https://curiousdevscorner.gumroad.com/
- https://medium.com/curious-devs-corner

I also wrote an article about the 8 Kubernetes deployment strategy types. You can read it here.

Thanks for reading, and see you next time!

Kubernetes Argo Cd Gitops DevOps Software Engineering

Published in Curious Devs Corner

Follow

52 followers · Last published 2 days ago

Welcome to Curious Devs Corner! Here, you'll find easy-to-follow tutorials and hands-on guides on the latest in Cloud Computing, DevOps, AI, Spring Boot, technology tips, and many more. Discover our ebooks on Gumroad: https://curiousdevscorner.gumroad.com/

Written by Kirshi Yin

Follow

2.2K followers · 125 following

Software Developer | Tech Blogger I write hands-on guides on SpringBoot, DevOps, AI and more. Explore my published books: https://curiousdevscorner.gumroad.com/

No responses yet

David B Chase

What are your thoughts?

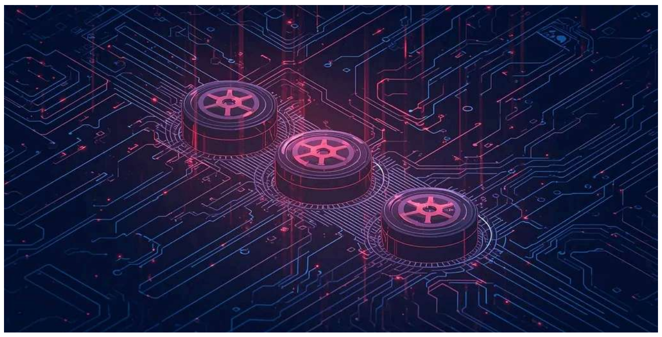
More from Kirshi Yin and Curious Devs Corner

In Curious Devs Corner by Kirshi Yin

Visualize Your Kubernetes Cluster with Headlamp: A Modern...

Deploy Headlamp Kubernetes Dashboard In-Cluster with Helm | Complete Guide for Kind,...

Oct 8 62 1



In Curious Devs Corner by Kirshi Yin

Kubernetes Deployment Strategy Types Explained: A Complete...

Learn when to use Rolling Updates, Blue/Green, Canary, and other deployment...

Oct 21 1

In Curious Devs Corner by Kirshi Yin

Helm vs Kustomize: Comparison, Use Cases & How to Combine Them

Learn the key differences between Helm and Kustomize and how to combine them for...

Jul 22 3

In Curious Devs Corner by Kirshi Yin

Kubernetes kubectl Commands Cheat Sheet for Everyday Use

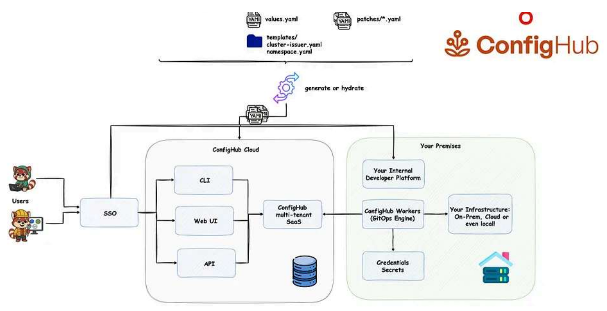
A curated list of kubectl commands you'll need

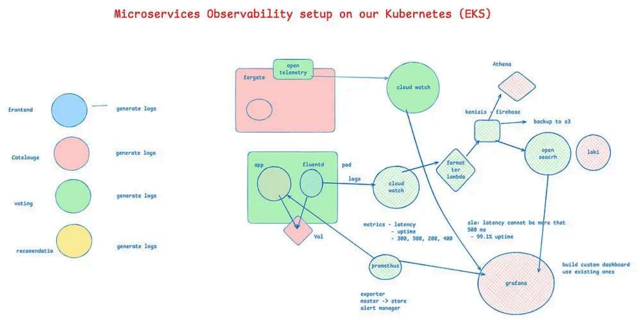
Oct 1 7

See all from Kirshi Yin

See all from Curious Devs Corner

Recommended from Medium





In ITNEXT by Artem Lajko

ConfigHub: Why Your Internal Developer Platform Needs It

See why GitOps often feels like a sprawl of configs, discover how to manage...

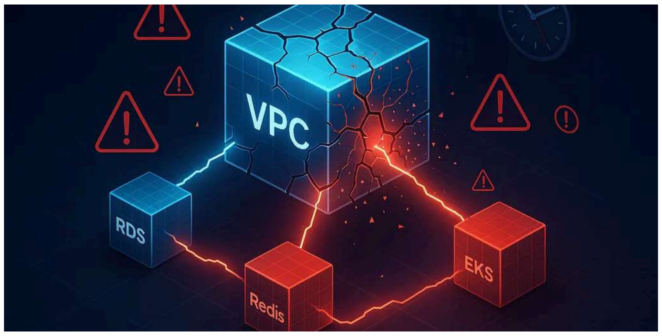
5d ago 64 1

In Towards AWS by Akhilesh Mishra

You're Not Ready for Kubernetes Observability Until You Understan...

The Real Story of Kubernetes Observability (That Actually Works in Production)

3d ago 101 1



Bhavyansh

The CI/CD Setup That Cut Our Deployment Time from 40 Minute...

How we optimized our deployment pipeline and got back 32 minutes of our lives (per...

Oct 27 65 3

Heinancabouly

How Crossplane's Auto-Upgrades Took Down Our Hospital...

When a broken patch release, dangerous defaults, and auto-upgrades collided with...

5d ago 6 1

DevOps — But Nobody
Told Your Team Yet
Why insternal developer platforms are redefining productivity, reliability, and ownership in 2025
</>

Freddie A

Platform Engineering Is the New DevOps—But Nobody Told Your...

Why internal developer platforms are redefining productivity, reliability, and...

Zudonu Osomudeya

The Real DevOps Tools Nobody Talks About

What 100+ engineers actually use daily (hint: it's not Kubernetes)

Nov 6 11

Nov 3 85 5

See more recommendations