A Guide to Efficient Canary Deployment Strategies on EKS with Argo CD

**Keywords:** AWS, EKS, ArgoCD, Deployment Strategy, Canary deployment

**Author:** [Mohan babu D](mailto:mohanbabu@cloudifyops.com)

**Reviewer:** [Vyshnav VM](mailto:vyshnav.vm@cloudifyops.com)

## **Introduction**

Canary deployment, a progressive release strategy, is seamlessly executed through Argo Rollouts—a Kubernetes controller. This method allows gradual exposure of a new version to a subset of users, minimizing risk. Argo Rollouts facilitates automated, fine-grained control over traffic shifting, enabling smooth transitions. Leveraging custom resource definitions, it ensures declarative management of rollout specifications. With features like automated analysis and rollback capabilities, Argo Rollouts empowers robust testing and monitoring during the canary process. This modern deployment approach enhances reliability and user experience, making it a valuable asset for Kubernetes-based applications.

## **Prerequisites**

* Amazon EKS Cluster: Ensure you have an EKS cluster set up.
* kubectl: Install kubectl to interact with your EKS cluster.
* Helm: Install Helm, as Argo CD is often deployed using Helm charts.

## **Argo Rollouts**

Argo Rollouts, a Kubernetes-native controller, elevates application deployment strategies, particularly in the context of canary deployments. Specializing in seamless software updates, Argo Rollouts introduces advanced features like canary analysis and progressive traffic shifting. By integrating Blue-Green methodologies, it streamlines the deployment process, offering a robust framework for controlled release of new application versions. With canary deployments, Argo Rollouts enables gradual exposure of updates to a subset of users, ensuring meticulous testing and monitoring.

The installation involves incorporating a set of Custom Resource Definitions (CRDs) into the Kubernetes Cluster, establishing a foundation for agile, reliable, and user-centric canary deployments.

| kubectl create namespace argo-rollouts |
| --- |

| Kubectl apply -n argo-rollouts -f https://github.com/argoproj/argo-rollouts/releases/latest/download/install.yaml |
| --- |

Check if we installed it successfully.

| kubectl get pod -n argo-rollouts |
| --- |

We can also install argo-rollouts through Helm.

| helm repo add argo https://argoproj.github.io/argo-helm  helm install my-release argo/argo-rollouts --set dashboard.enabled=true |
| --- |

This command installs Argo Rollouts, deploying its controller and a metrics server for performance data collection. Additionally, it enables the Argo Rollouts dashboard, offering a user-friendly interface to visualize and manage deployment rollouts.

## **Argo Rollouts Deployment Strategy**

Argo Rollouts provides a robust deployment strategy for Kubernetes environments. Leveraging techniques like Blue-Green, Canary, and AB testing, facilitates seamless application updates. With traffic shifting and automated rollback features, Argo Rollouts ensures controlled, reliable, and flexible deployment processes, enhancing the overall management of application versions in Kubernetes clusters.

The configuration of Argo Rollouts has a strategy property for us to choose the deployment strategy we want, with two values ​​of blue-green or canary.

| apiVersion: argoproj.io/v1alpha1  kind: Rollout  metadata:  name: rollouts-demo  spec:  replicas: 5  strategy:  canary:  steps:  - setWeight: 20  - pause: {}  - setWeight: 40  - pause: {duration: 10}  - setWeight: 60  - pause: {duration: 10}  - setWeight: 80  - pause: {duration: 10}  revisionHistoryLimit: 2  selector:  matchLabels:  app: rollouts-demo  template:  metadata:  labels:  app: rollouts-demo  spec:  containers:  - name: rollouts-demo  image: argoproj/rollouts-demo:yellow  ports:  - name: http  containerPort: 8080  protocol: TCP  resources:  requests:  memory: 32Mi  cpu: 5m |
| --- |

This Argo Rollout YAML defines a canary deployment for the rollouts-demo application with five replicas. The strategy involves incremental weight adjustments at 20%, 40%, 60%, and 80%, pausing for 10 seconds at each step ( 40%, 60%, and 80%). The canary deployment strategy involves an infinite pause at 20%, to promote to higher percentages, use the argo rollouts promote rollouts-demo command, which automatically progresses to the next weight defined in the canary strategy.

| kubectl apply -f canary-rollout.yaml  Kubectl get rollout |
| --- |

When we create a Rollout, the Argo Rollouts implicitly create a ReplicaSet for a current revision.

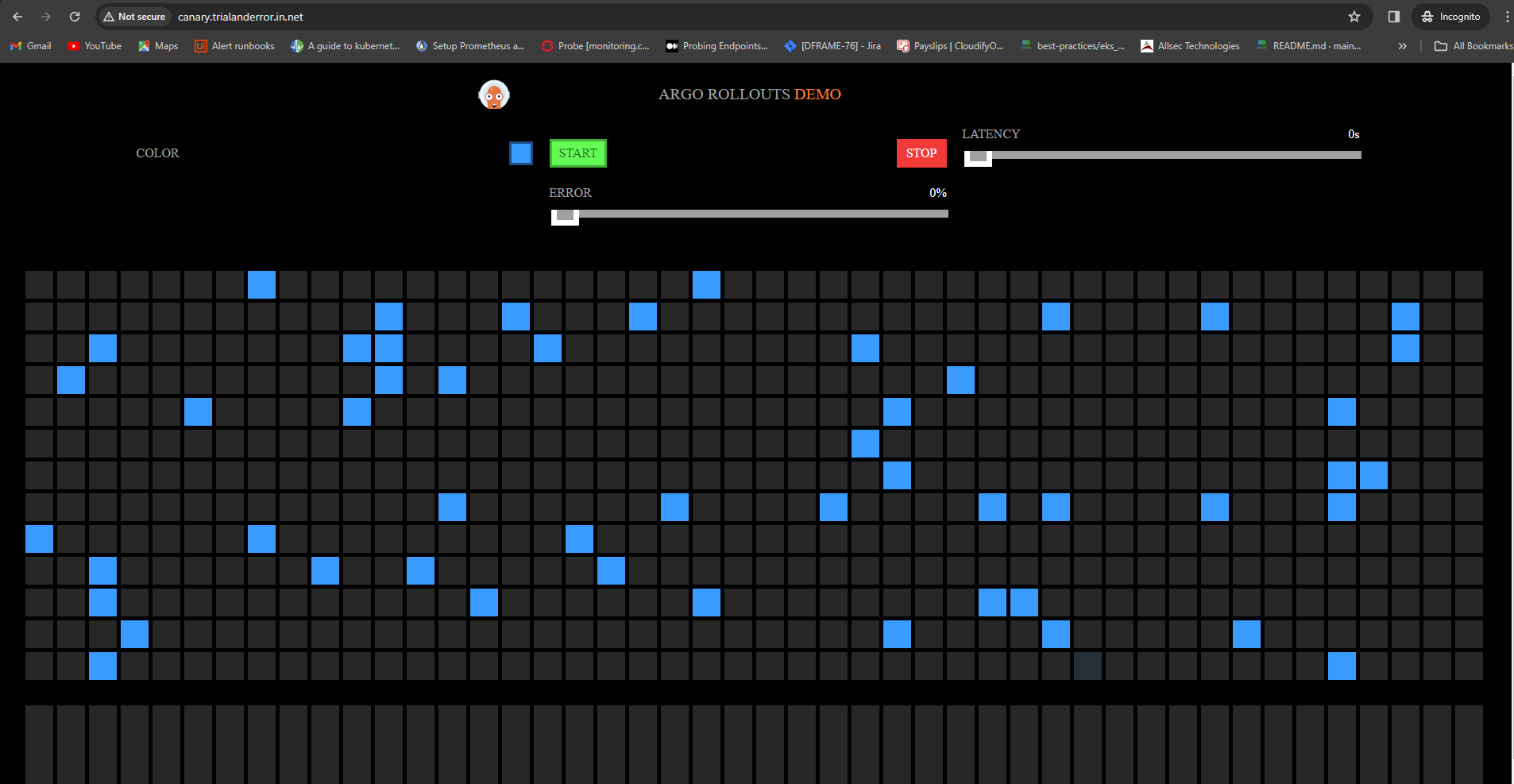
The two Services properties are the same except for the name property. Next, we create a Rollout Object.

| apiVersion: v1  kind: Service  metadata:  name: rollouts-demo  spec:  type: LoadBalancer  ports:  - port: 80  targetPort: http  protocol: TCP  name: http  selector:  app: rollouts-demo --- |
| --- |

Ensure that Replica Set and Pod are running, next, we create a Service.

| kubectl apply -f service.yaml |
| --- |

Open the browser, we will see the UI below.



Now, we change the image property of the Rollout Object.

| piVersion: argoproj.io/v1alpha1  kind: Rollout  metadata:  name: rollouts-demo  spec:  replicas: 5  strategy:  canary:  steps:  - setWeight: 20  - pause: {}  - setWeight: 40  - pause: {duration: 10}  - setWeight: 60  - pause: {duration: 10}  - setWeight: 80  - pause: {duration: 10}  revisionHistoryLimit: 2  selector:  matchLabels:  app: rollouts-demo  template:  metadata:  labels:  app: rollouts-demo  spec:  containers:  - name: rollouts-demo  image: argoproj/rollouts-demo:yellow  ports:  - name: http  containerPort: 8080  protocol: TCP  resources:  requests:  memory: 32Mi  cpu: 5m |
| --- |

Updating Rollout Object.

| kubectl apply -f canary-rollout.yaml |
| --- |

At this point, Argo Rollouts will create a new ReplicaSet for the new configuration.

| kubectl get rs |
| --- |

After we check the new ReplicaSet and see that all is well, next, we promote the new revision of ReplicaSet by updating the rollouts-demo service to point to it, we run the following command.

| kubectl argo rollouts promote rollouts-demo |
| --- |

Now, Argo Rollouts updates the rollouts-demo service to point to the new ReplicaSet, after waiting (default 30 seconds), the old ReplicaSet is scaled down.

Argo Rollout provides us with a dashboard, which we can enable using kubectl or using quay.io/argoproj/kubectl-argo-rollouts container image.

| kubectl argo rollouts dashboard Argo Rollouts Dashboard is now available at localhost 3100 |
| --- |

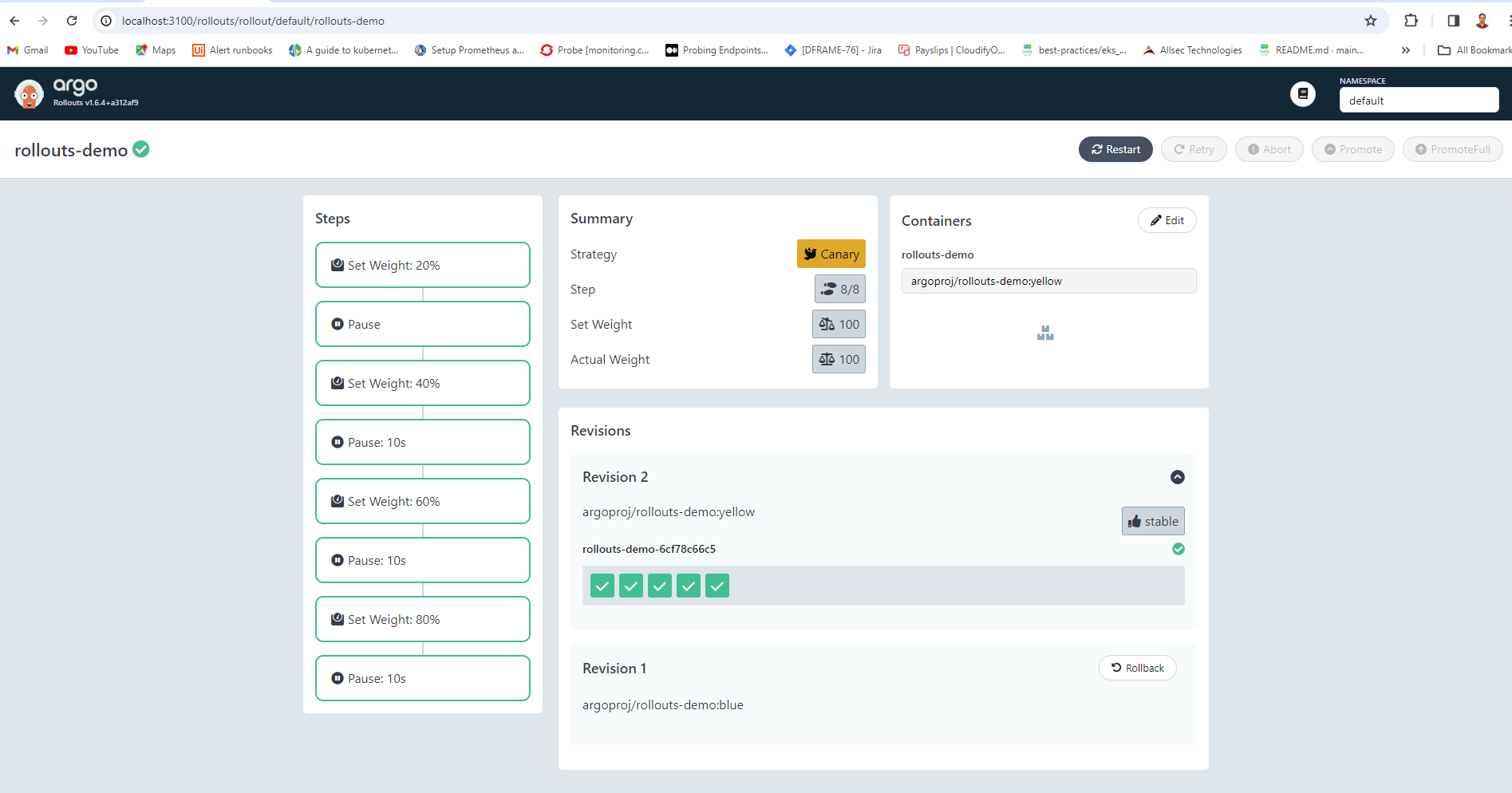
Navigate to localhost:3100 and we will see the dashboard of Argo Rollouts. Or one can use the following ingress setup for dashboard.

To set up an Ingress for dashboard, deploy the Ingress controller, map the DNS records in the domain name using the DNS name of the loadbalancer. Once it is configured, set up an ingress rule for the dashboard Service.

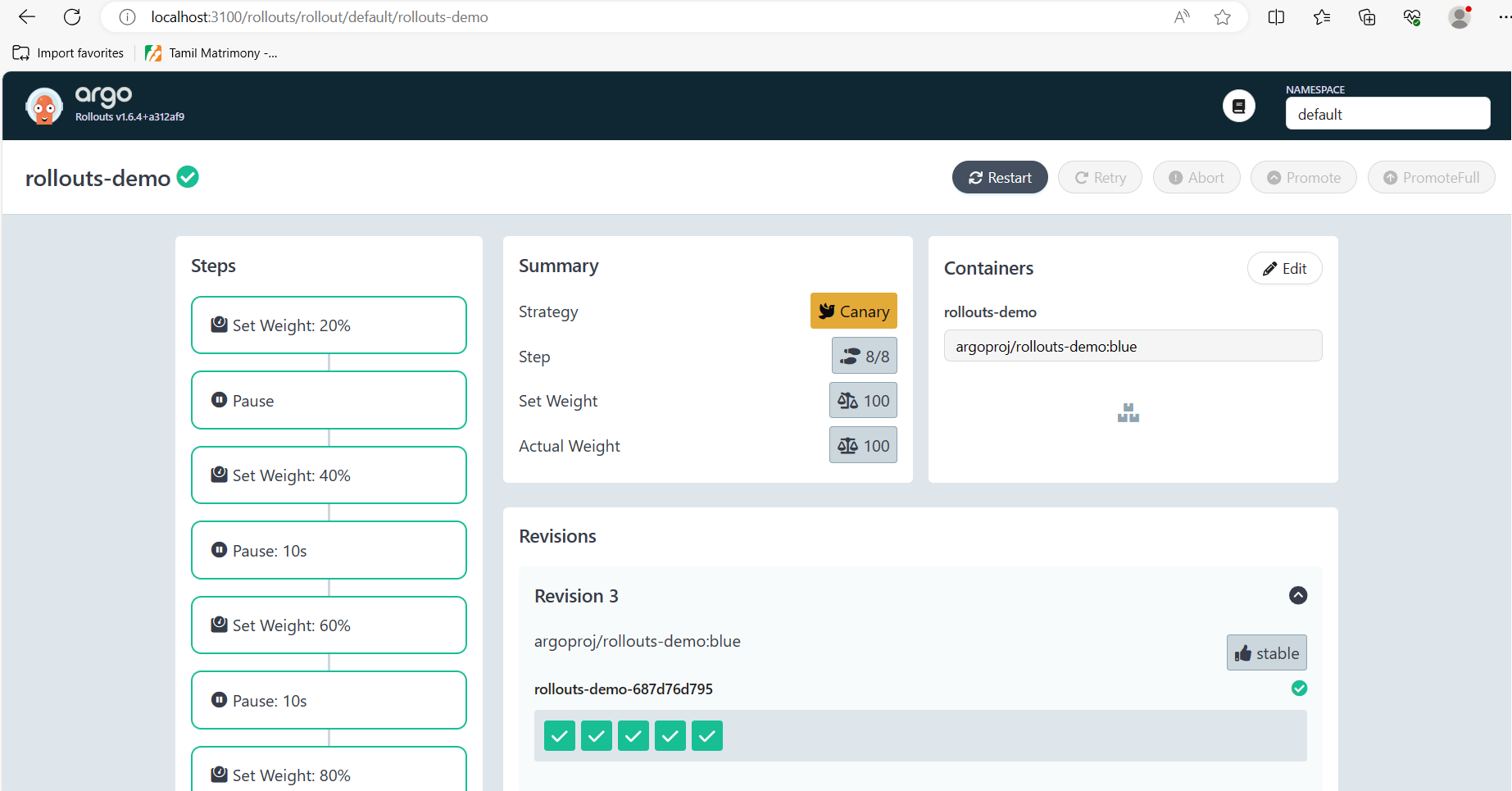
For example, the following can be configured as an ingress rule

| apiVersion: networking.k8s.io/v1 kind: Ingress metadata:  name:argo-rollouts-dashboard  namespace: default  annotations:  nginx.ingress.kubernetes.io/rewrite-target: / spec:  ingressClassName: nginx  rules:  - host: < DNS name >  http:  paths:  - path: /dashboard  pathType: Prefix  backend:  service:  name:argo-rollouts-dashboard  port:  number: 8080 |
| --- |

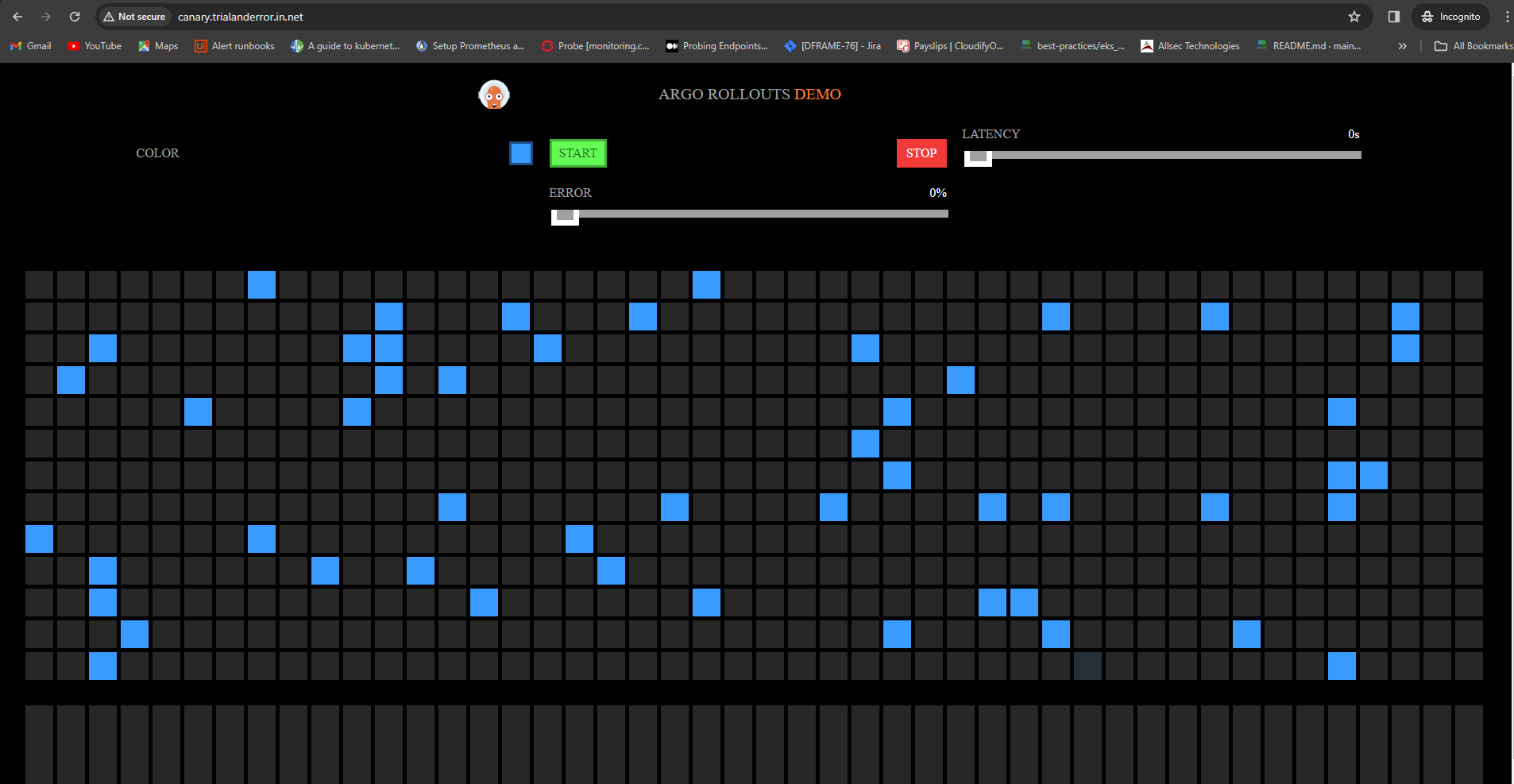
---



The following image illustrates 100% of the rollout in the argo rollouts dashboard:



The following image illustrates blue version of the app in the DNS name in the browser:



## **Conclusion**

In conclusion, Canary deployment with Argo Rollouts emerges as a pivotal strategy for enhancing application releases in Kubernetes environments. By seamlessly integrating advanced features such as canary analysis and traffic shifting, Argo Rollouts empowers developers to ensure a smooth transition to new software versions. The automated rollback capabilities and integration with monitoring tools further solidify its position as a reliable and flexible deployment solution. This approach not only minimizes risks associated with updates but also optimizes user experience by allowing careful testing and monitoring of changes. With Argo Rollouts, teams can embrace a modern, Kubernetes-native deployment paradigm, fostering agility, reliability, and continuous improvement in their software delivery pipelines.