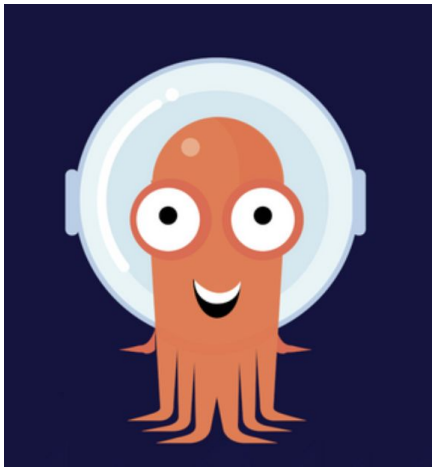
Progressive Delivery and Blue-Green Deployment using

Argo Rollouts



- **Ninad Desai (Sr. Site Reliability Engineer)**
  @InfraCloud(We're hiring!)

Argo Projects

https://github.com/argoproj/argoproj

- Argo Workflows - Container-native Workflow Engine
- Argo CD - Declarative GitOps Continuous Delivery
- Argo Events - Event-based Dependency Manager
- Argo Rollouts - Progressive Delivery with support for Canary and Blue Green deployment strategies

- Argoproj-labs - separate GitHub org that is setup for community contributions related to the Argoproj ecosystem

Agenda for Argo Rollout Series:

- Progressive Delivery
- Why Argo Rollouts?
- Intro of project
- How it works?
- Argo Rollout Installation
- Walkthrough of UI, use cases etc.
- Rollout Spec intro
- Rollout Spec - Blue Green
- Rollout Spec - Canary
- Rollout Spec - Analysis Template
- Rollout Spec - Canary+ Analysis
- (In all scenarios above - walk through YAML as well as some UI)
- Experiment Brief
- Migrating existing Deployments/Helm charts to Argo Rollout

Progressive Delivery

“Art of moving fast but with control”

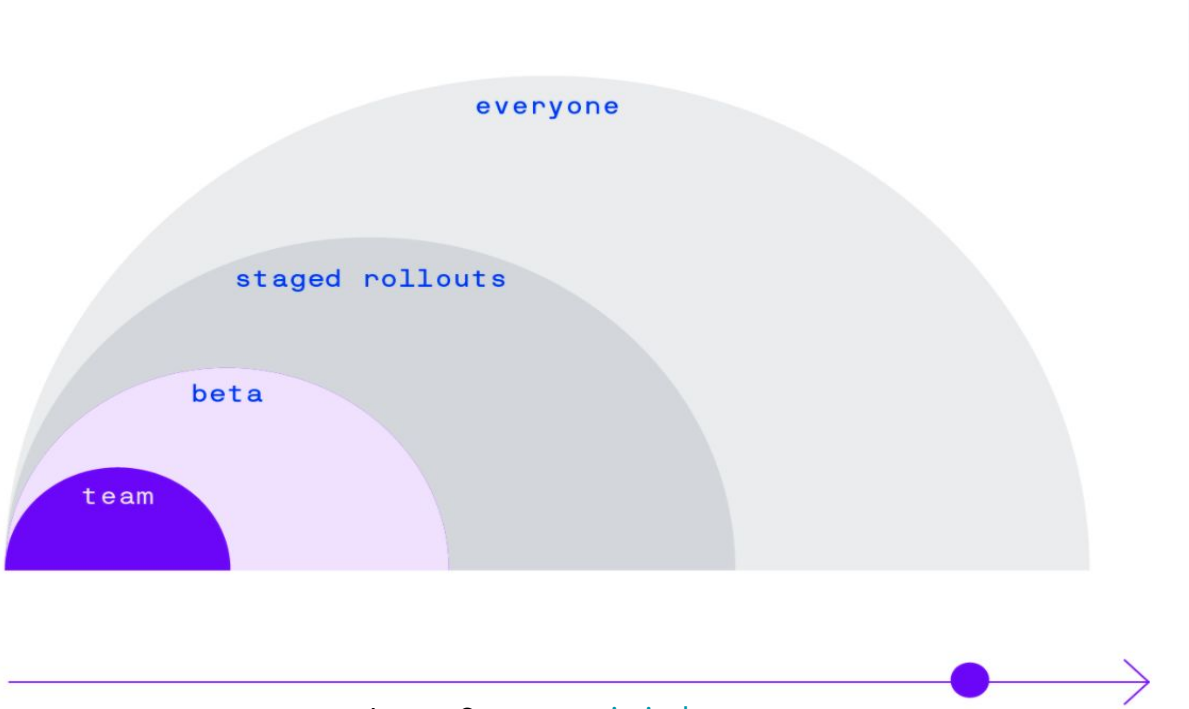
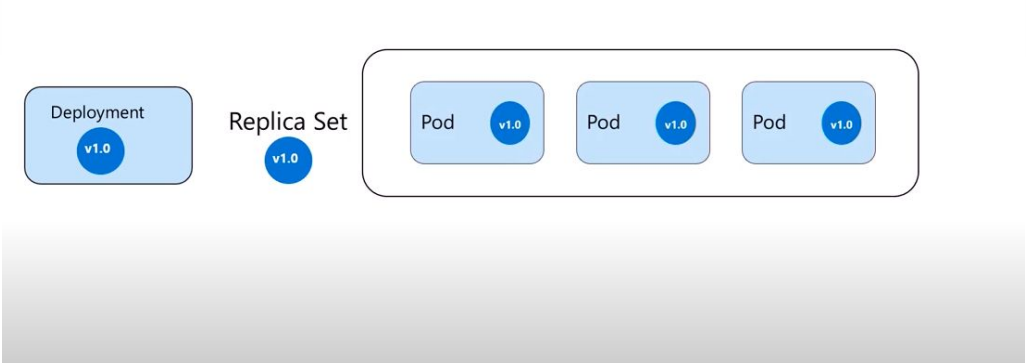


Image Source: optimizely.com

Rolling update strategy in Kubernetes Deployments

default ***RollingUpdate*** Strategy of Kubernetes deployments

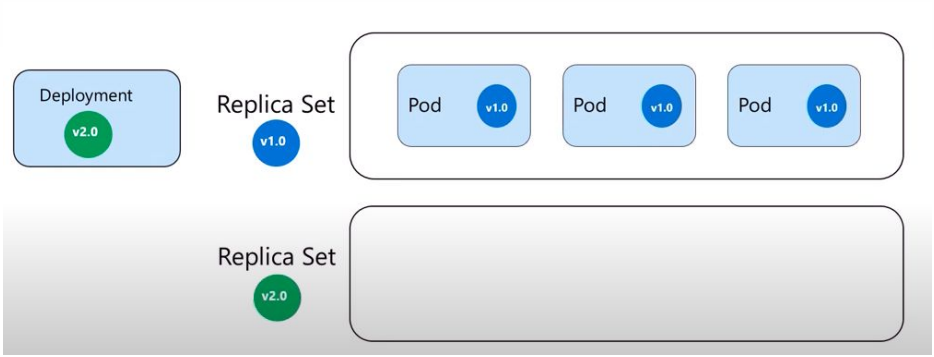
- **Need of rolling update strategy**
  - To implement Reliable Zero Downtime Upgrades with Gradual update
- **The way rolling update strategy works**



Rolling update strategy in Kubernetes Deployments

default ***RollingUpdate*** Strategy of Kubernetes deployments

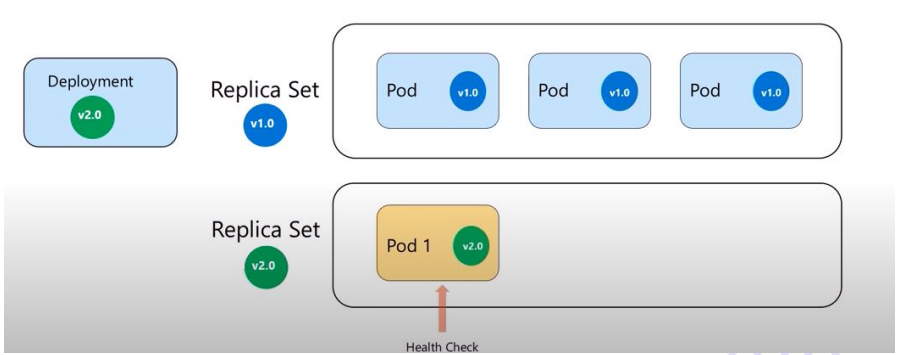
- **Need of rolling update strategy**
  - To implement Reliable Zero Downtime Upgrades with Gradual update
- **The way rolling update strategy works**



Rolling update strategy in Kubernetes Deployments

default ***RollingUpdate*** Strategy of Kubernetes deployments

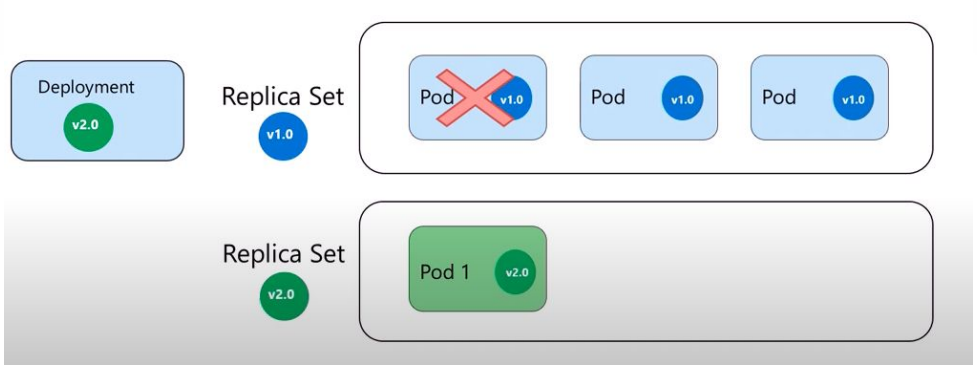
- **Need of rolling update strategy**
  - To implement Reliable Zero Downtime Upgrades with Gradual update
- **The way rolling update strategy works**



Rolling update strategy in Kubernetes Deployments

default ***RollingUpdate*** Strategy of Kubernetes deployments

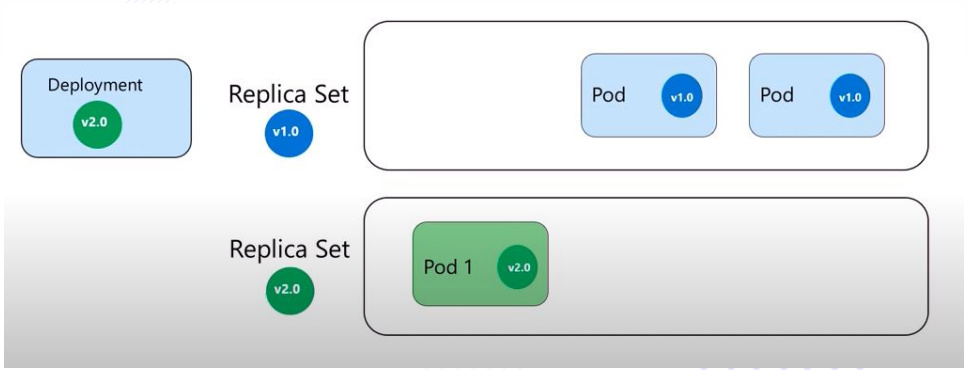
- **Need of rolling update strategy**
  - To implement Reliable Zero Downtime Upgrades with Gradual update
- **The way rolling update strategy works**



Rolling update strategy in Kubernetes Deployments

default ***RollingUpdate*** Strategy of Kubernetes deployments

- **Need of rolling update strategy**
  - To implement Reliable Zero Downtime Upgrades with Gradual update
- **The way rolling update strategy works**



Rolling update strategy in Kubernetes Deployments

default ***RollingUpdate*** Strategy of Kubernetes deployments

- **Need of rolling update strategy**
  - To implement Reliable Zero Downtime Upgrades with Gradual update
- **The way rolling update strategy works**

Why Argo Rollouts?

Limitations with default ***RollingUpdate*** Strategy of Kubernetes deployments

- Few controls over the speed of the rollout
- Inability to control traffic flow to the new version
- Readiness probes are unsuitable for deeper, stress, or one-time checks
- No ability to query external metrics to verify an update
- Can halt the progression, but unable to automatically abort and rollback the update

Argo Rollouts

https://github.com/argoproj/argo-rollouts

- Argo Rollouts is a Kubernetes controller and set of CRDs
- Provide deployment capabilities such as blue-green, canary, canary analysis, experimentation, and progressive delivery features to Kubernetes
- Drop in replacement for the Kubernetes Deployment

Some use cases of Argo Rollouts

- To run last-minute functional tests on the new version before it starts to serve production traffic
- To determine if the new version is performant compared to the old version
- To control the percentages of traffic, user want the new version to receive and the amount of time to wait between percentages, before directing whole traffic to the new version

How it works?

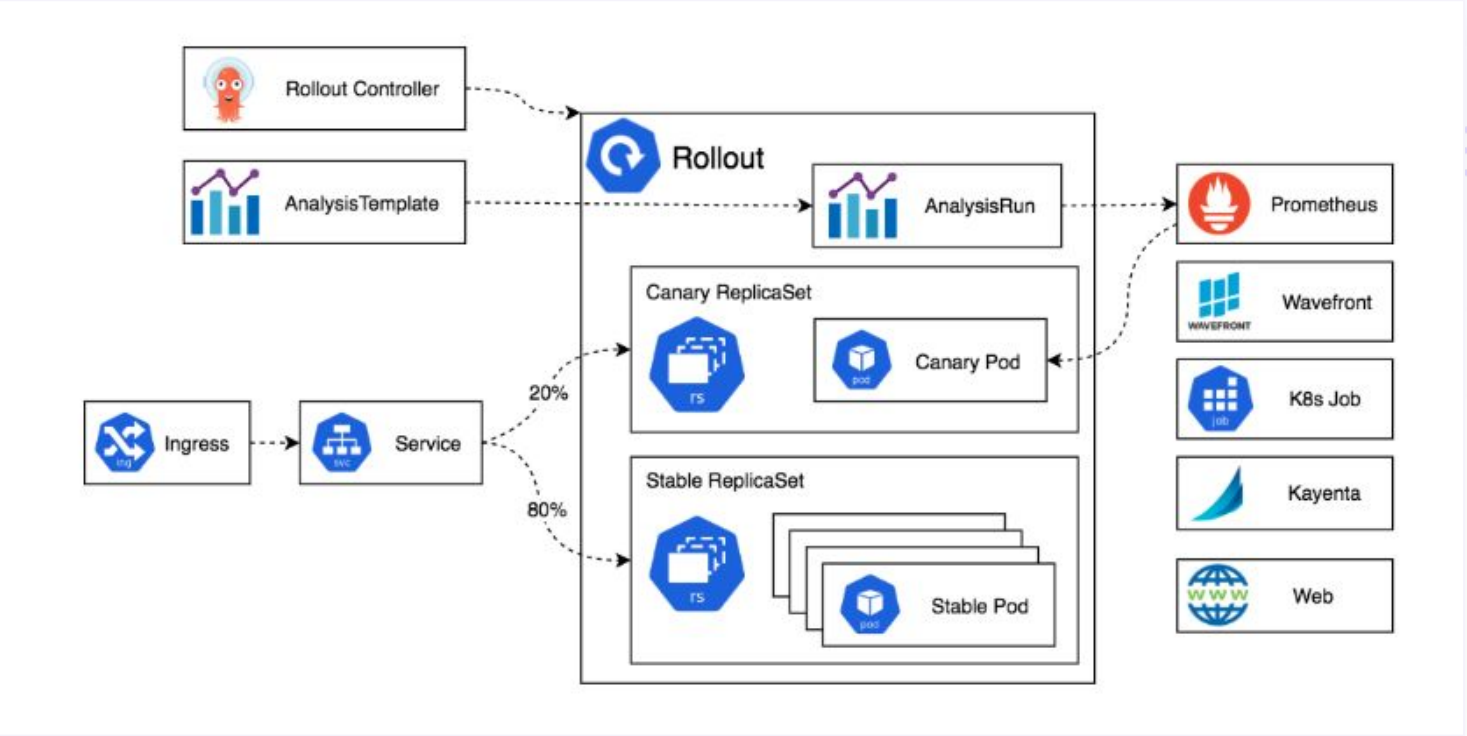


Image Source https://argoproj.github.io/argo-rollouts/architecture/

Argo Rollout Installation

```
kubectl create namespace argo-rollouts
```

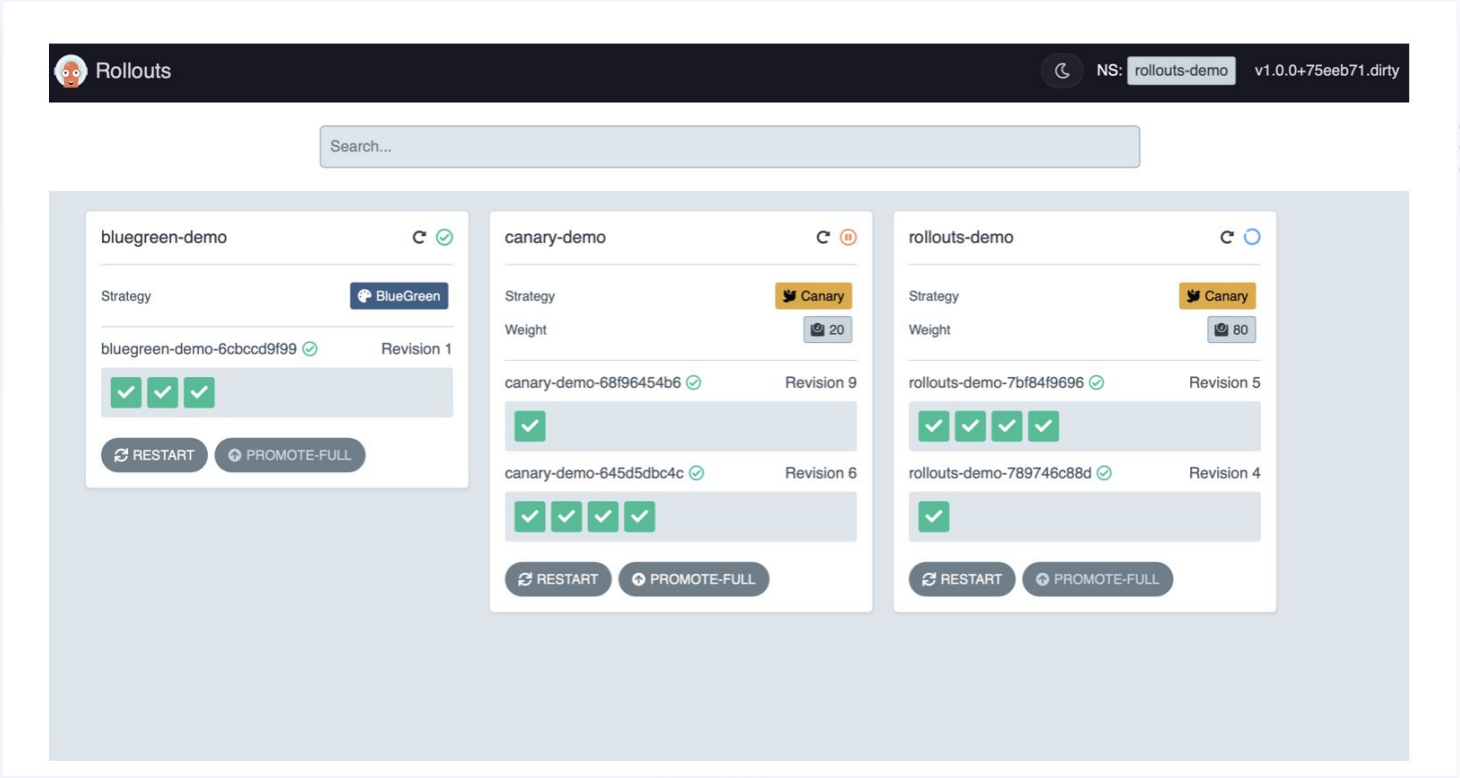
```
kubectl apply -n argo-rollouts -f https://github.com/argoproj/argo-rollouts/releases/latest/download/install.yaml
```

Check whether Argo rollout is up and running

```
kubectl get pods -n argo-rollouts
```

Argo Rollout UI

(run kubectl argo rollouts dashboard and then visit: localhost:3100)



Rollout Spec details

(https://argoproj.github.io/argo-rollouts/features/specification/)

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: example-rollout-canary
spec:
  replicas: 5   # Number of desired pods. # Defaults to 1.
  analysis:
    # limits the number of successful analysis runs and experiments to be stored in a history
    # Defaults to 5.
    successfulRunHistoryLimit: 10
    # limits the number of unsuccessful analysis runs and experiments to be stored in a history.
    # Stages for unsuccessful: "Error", "Failed", "Inconclusive"# Defaults to 5.
    unsuccessfulRunHistoryLimit: 10
  strategy:
    # Blue-green update strategy
    blueGreen:
      # Reference to service that the rollout modifies as the active service Required.
      activeService: active-service
      # Name of the service that the rollout modifies as the preview service.
      # +optional
      previewService: preview-service
      # Pre-promotion analysis run which performs analysis before the service
      # cutover. +optional
      prePromotionAnalysis:
        templates:
        - templateName: success-rate
        args:
        - name: service-name
          value: guestbook-svc.default.svc.cluster.local
      # Anti Affinity configuration between desired and previous ReplicaSet. # Only one must be s
      antiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution: {}
        preferredDuringSchedulingIgnoredDuringExecution:
          weight: 1 # Between 1 - 100
```

```yaml
canary:
  # Reference to a service which the controller will update to selectcanary pods. Required for t
  canaryService: canary-service
  # Reference to a service which the controller will update to select stable pods. Required for
  stableService: stable-service
  # Background analysis to run during a rollout update. Skipped upon
  # initial deploy of a rollout. +optional
  analysis:
    templates:
    - templateName: success-rate
    args:
    - name: service-name
      value: guestbook-svc.default.svc.cluster.local
  # Steps define sequence of steps to take during an update of the canary. Skipped upon initial
  steps:
  # Sets the ratio of canary ReplicaSet to 20%
  - setWeight: 20
  # Pauses the rollout for an hour. Supported units: s, m, h
  - pause:
      duration: 1h
  # Pauses indefinitely until manually resumed
  - pause: {}
  # an inline analysis step
  - analysis:
      templates:
      - templateName: success-rate
  trafficRouting:
    # NGINX Ingress Controller routing configuration
    nginx:
      stableIngress: primary-ingress  # required
      annotationPrefix: customingress.nginx.ingress.kubernetes.io # optional
      additionalIngressAnnotations:   # optional
        canary-by-header: X-Canary
        canary-by-header-value: iwantsit
```

Blue-Green Rollout concept

Image Source https://argoproj.github.io/argo-rollouts/concepts

Rollout Spec - Blue Green

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: rollout-bluegreen
spec:
  replicas: 2
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: rollout-bluegreen
  template:
    metadata:
      labels:
        app: rollout-bluegreen
    spec:
      containers:
      - name: rollouts-demo
        image: argoproj/rollouts-demo:blue
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
  strategy:
    blueGreen:
      # activeService specifies the service to update with the new template hash at time of promotion
      # This field is mandatory for the blueGreen update strategy.
      activeService: rollout-bluegreen-active
      # previewService specifies the service to update with the new template hash before promotion.
      # This allows the preview stack to be reachable without serving production traffic.
      # This field is optional.
      previewService: rollout-bluegreen-preview
      # autoPromotionEnabled disables automated promotion of the new stack by pausing the rollout
      # immediately before the promotion. If omitted, the default behavior is to promote the new
      # stack as soon as the ReplicaSet are completely ready/available.
      # Rollouts can be resumed using: `kubectl argo rollouts promote ROLLOUT`
      autoPromotionEnabled: false
```

Hands-On lab for Blue-Green Roll out strategy

Pro’s and Con’s - Blue Green deployment strategy

Pro’s:

- Clients get API consistency
- Preview stack can be tested before receiving production traffic
- Rollbacks/Aborts are immediate

Con’s:

- 2x resource costs during updates
- Cannot canary

argo

Thank you!