**Blue Green Deployment Strategy**

**Use this deployment strategy to test the new version of the application by a set of people in our organization in Production environment before it is made available to public on internet.**

Icon

Description automatically generated

**Step1:**

1. Existing Deployment (Blue)

2. Existing Service (Blue)

apiVersion: apps/v1

kind: Deployment

metadata:

  name: deployment-blue

spec:

  replicas: 10

  selector:

    matchLabels:

      app: app-blue

  template:

    metadata:

      name: nginx-pod

      labels:

        app: app-blue

    spec:

      containers:

      - name: mynginx-con

        image: nginx:1.18.0

        ports:

        - containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

  name: my-service-blue

spec:

  type: LoadBalancer

  selector:

    app: app-blue

  ports:

    - protocol: TCP

      port: 8080

      targetPort: 80

3. Test the Blue version with the URL: http://localhost:8080

**Step2:**

3. New Deployment (Green)

Clone the old deployment YAML and change Deployment Name, Label and Image

4. New Service (Green)

Clone the old service and change Name, Selector Label and Port

apiVersion: apps/v1

kind: Deployment

metadata:

  name: deployment-green

spec:

  replicas: 10

  selector:

    matchLabels:

      app: app-green

  template:

    metadata:

      name: httpd-pod

      labels:

        app: app-green

    spec:

      containers:

      - name: httpd-con

        image: httpd

        ports:

        - containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

  name: my-service-green

spec:

  type: LoadBalancer

  selector:

    app: app-green

  ports:

    - protocol: TCP

      port: 8081

      targetPort: 80

Step3: Test of New/Green Deployment using New/Green Service (**http://localhost:8081)**

Step4: Move Traffic from Blue(Old) to Green(New) Deployment

1. Change **Selector** of existing blue service to New Deployment Label (app: app-green)

apiVersion: v1

kind: Service

metadata:

  name: my-service-blue

spec:

  type: LoadBalancer

  selector:

    app: app-green

  ports:

    - protocol: TCP

      port: 8080

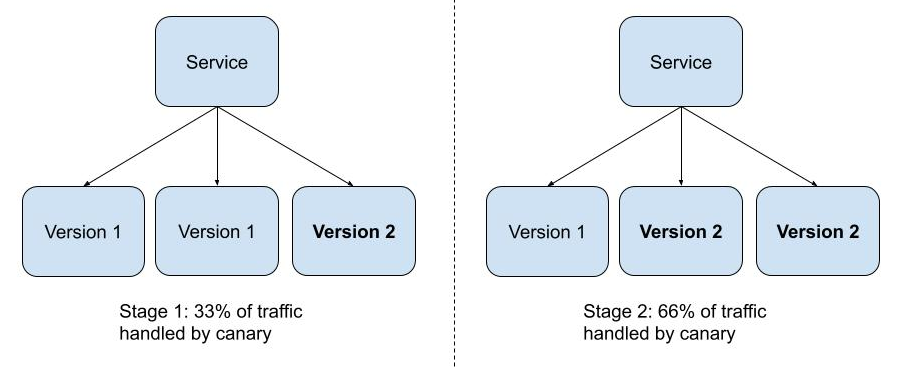
      targetPort: 80

1. **Test using** [**http://localhost:8080**](http://localhost:8080) **– It now generates the output from New/Green version.**
2. Delete Blue Deployment (Old version) and Green Service (new service)

**Canary Deployment**

**Gradually** move load from existing deployment to New Deployment.

1. Create New Deployment with **Same Label** as Existing Deployment.
2. Gradually increase replicas of new and decrease replicas of existing.



apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 10

  selector:

    matchLabels:

      app: nginx-app

  template:

    metadata:

      name: nginx-pod

      labels:

        app: nginx-app

    spec:

      containers:

      - name: mynginx-con

        image: nginx:1.16.0

        ports:

        - containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

  name: my-service-lb

spec:

  type: LoadBalancer

  selector:

    app: nginx-app

  ports:

    - protocol: TCP

      port: 8080

      targetPort: 80

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment-canary

spec:

  replicas: 2

  selector:

    matchLabels:

      app: nginx-app

  template:

    metadata:

      name: nginx-pod

      labels:

        app: nginx-app

    spec:

      containers:

      - name: mynginx-con

        image: httpd

        ports:

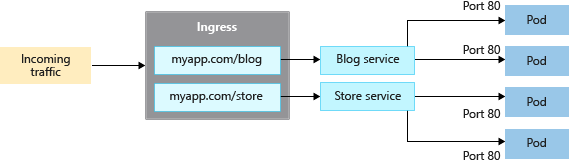
        - containerPort: 80

With time reduce the replicas of Main Deployment and increase the replica of Canary Deployment.

Do this as long as the users are not complaining about the new deployment and old deployment replica changes to "0" and all traffic is shifted to Canary Deployment which now should become Main deployment.

**Ingress Controller**

* Ingress exposes **HTTP and HTTPS** routes from **outside the cluster** to services within the cluster. Traffic routing is controlled by **rules** defined on the Ingress resource.
* Ingress actually acts as a reverse proxy to bring traffic into the cluster, then uses internal service routing to get the traffic where it is going.
* Ingress may be configured to give Services externally reachable URLs, load balance traffic, terminate SSL / TLS, and offer name-based virtual hosting.
* **Ingress controllers** work **at layer 7** (unlike LoadBalancer Service which works at layer 4), and can use more intelligent rules to distribute application traffic.



A screenshot of a computer

Description automatically generated with low confidence

In order for the Ingress resource to work, the cluster must have an ingress controller running.

(List of Ingress Controllers: <https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/>)

**Ingress with NGINX Ingress Controller** as a reverse proxy and load balancer **(**[**https://kubernetes.github.io/ingress-nginx/deploy/**](https://kubernetes.github.io/ingress-nginx/deploy/)**).**

**Step1:**

For Docker Desktop - Install NGINX Ingress controller

kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.8.0/deploy/static/provider/cloud/deploy.yaml>

OR  
For Minikube: **minikube addons enable ingress**

**Step2:** Create a YAMLs file as below

**nginx.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: mynginx

spec:

replicas: 1

selector:

matchLabels:

app: mynginx

template:

metadata:

labels:

app: mynginx

spec:

containers:

- image: nginx

name: mynginx

---

apiVersion: v1

kind: Service

metadata:

name: mynginx-cip

spec:

type: ClusterIP

ports:

- port: 8090

protocol: TCP

targetPort: 80

selector:

app: mynginx

**kubectl apply -f nginx.yaml**

**httpd.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: myhttpd

name: myhttpd

spec:

replicas: 1

selector:

matchLabels:

app: myhttpd

template:

metadata:

labels:

app: myhttpd

spec:

containers:

- image: **httpd**

name: myhttpd

---

apiVersion: v1

kind: Service

metadata:

name: myhttpd-cip

spec:

ports:

- port: 8090

protocol: TCP

targetPort: 80

selector:

app: myhttpd

**kubectl apply -f httpd.yaml**

**Ingress.yaml**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: my-ingress

  annotations:

    nginx.ingress.kubernetes.io/rewrite-target: /

spec:

  ingressClassName: nginx

  defaultBackend:

    service:

      name: mynginx-cip

      port:

        number: 8090

  rules:

  - http:

      paths:

      - path: /nginx

        pathType: Prefix

        backend:

          service:

            name: mynginx-cip

            port:

              number: 8090

      - path: /httpd

        pathType: Prefix

        backend:

          service:

            name: myhttpd-cip

            port:

              number: 8090

**kubectl apply -f Ingress.yaml**

**Rules based on Domain name:**

rules:

**- host: nginx.mydomain.com**

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: mynginx-cip

port:

number: 8090

**- host: httpd.mydomain.com**

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: myhttpd-cip

port:

number: 8090

Note: Stop IIS or Apache or Tomcat or any other service running on your machine on Port 80.

**minikube ip**

curl --header 'Host: demo.mydomain.com' http://localhost/nginx

curl --header 'Host: demo.mydomain.com' http://localhost/httpd

curl --header 'Host: demo.mydomain.com' http://192.168.49.2:80

**Note: In the above URL: 192.168.49.2 is the Minikube IP Address.**

OR

**Step3:** Edit as administrator:

**Windows**: c:\windows\system32\drivers\etc\hosts

**Mac/Linux**: /etc/hosts

**sudo vi /etc/hosts**

Edit file as below and save.

192.168.49.2 demo.mydomain.com #Minikube

127.0.0.1 demo.mydomain.com #Docker Desktop

curl <http://demo.mydomain.com>

curl <http://demo.mydomain.com/nginx>

curl <http://demo.mydomain.com/httpd>

**Step4:** Open in browser following URL's (**may require to wait for couple of minutes**). Also stop other WebService Services running on Port 80.

**To View the Logs and the POD to which the traffic is forward:**

kubectl get pods **-n ingress-nginx**

kubectl logs **-n ingress-nginx** pod/ingress-nginx-controller-XXXXXXX-xxxx