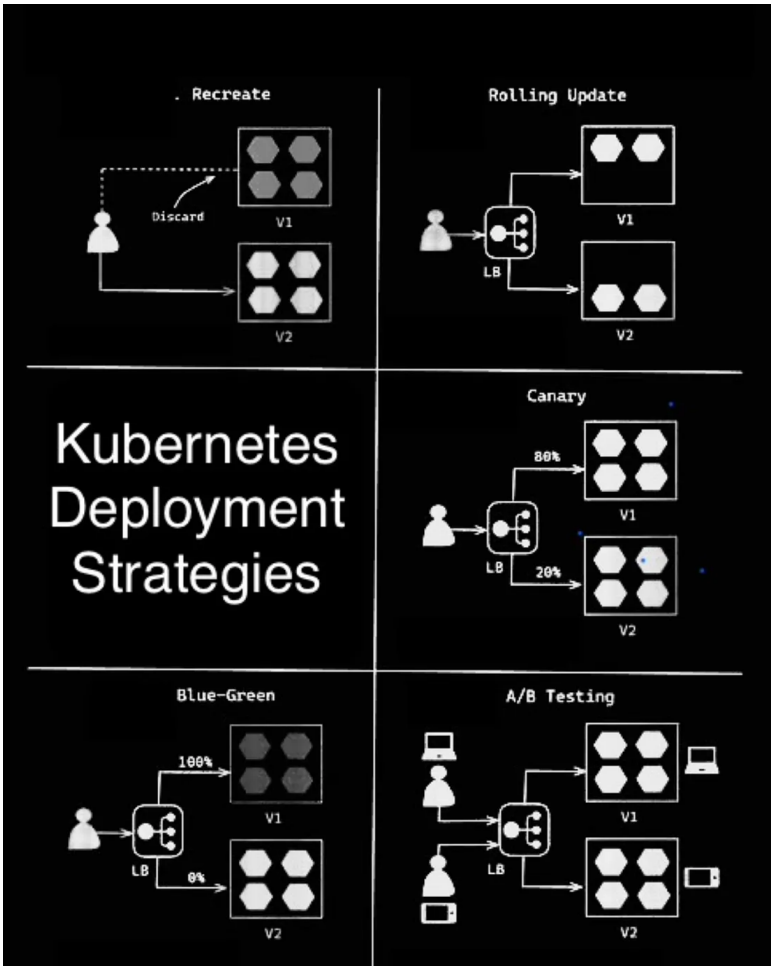
Deployment Strategies to definitely know!!!

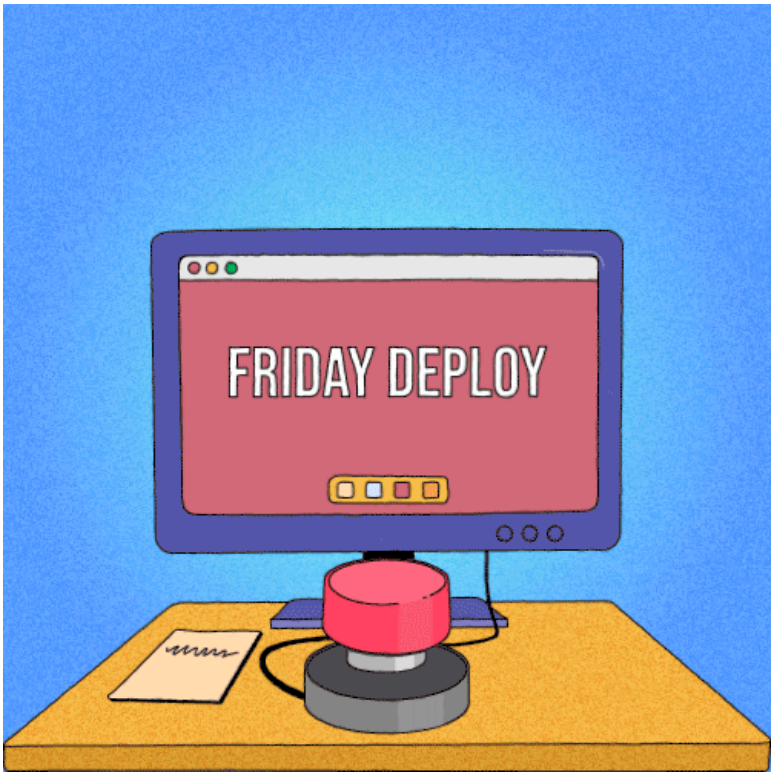
Video Tutorial:

https://youtu.be/AsXNDf5-cTs



Subscribe:

https://youtube.com/@cloudchamp?sub_confirmation=1



What is a Deployment? 🚀

Deployment means releasing or upgrading a software so that people can use it.

What is Deployment Strategy? 🤔

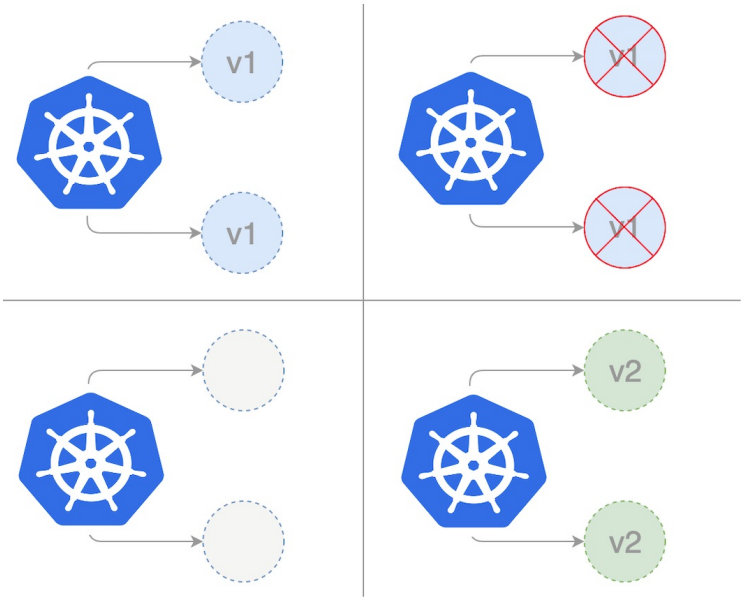
A deployment strategy is a plan for how to release and update software carefully to avoid problems for users.

Different Deployment Strategy:

1. **Recreate** 🔁
2. **Rolling Update (Default)** 🔄
3. **Blue Green** 🟦🟩
4. **Canary** 🐤
5. **A/B testing** 🅰️🅱️

1. Recreate:

> All old stuff is thrown away, then new stuff is put in its place.



Recreate Deployment strategy.

- **Definition:** In the Recreate strategy, the existing instances of the application are terminated all at once, and then new instances are created with the updated configuration.

- **Pros:**
  - Simple and straightforward.
  - Ensures all instances are running the same version.
- **Cons:**
  - Downtime during the transition.

- **Example:** If you have version 1.0 of your application deployed, updating to version 2.0 using the Recreate strategy would involve terminating all instances running version 1.0 and then deploying new instances running version 2.0.

- **When to use:** Recreate deployment strategy is suitable for applications where downtime during deployment is acceptable or when the application startup time is short.

> 💡 If your application is stateless and can afford to be temporarily unavailable during updates, you can use the Recreate strategy.

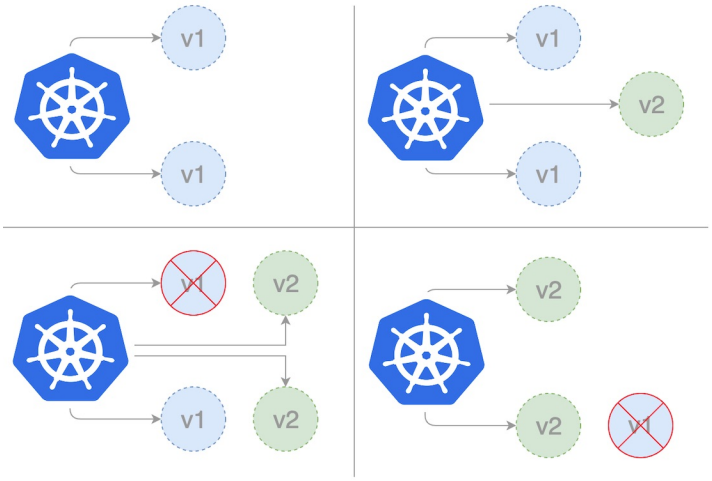
▼ Implementation

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: archetype-deployment
  labels:
    app: cloudchamp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: cloudchamp
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: cloudchamp
    spec:
      containers:
      - name: deploy
        image: cloudchamp/test:v2
```

```
        ports:
        - containerPort: 80
```

2. Rolling Update (Default):

> Changes are made bit by bit, so things keep working smoothly.



- **Definition:** In Rolling Update, new instances are gradually created while old instances are gradually terminated, ensuring a smooth transition without downtime.

- **Pros:**
  - Zero downtime deployment.
  - Easy to roll back in case of issues.
- **Cons:**
  - Longer deployment time compared to Recreate.

- **Example:** When updating from version 1.0 to version 2.0 using Rolling Update, Kubernetes will gradually replace instances running version 1.0 with instances running version 2.0, ensuring continuous availability.

- **When to use:** Rolling Update is suitable for applications that require zero downtime during deployment and can handle multiple versions running simultaneously.

> 💡 Rolling update can be achieved by simply updating the image tag in the Deployment manifest. Kubernetes will automatically perform rolling updates.

▼ Implementation

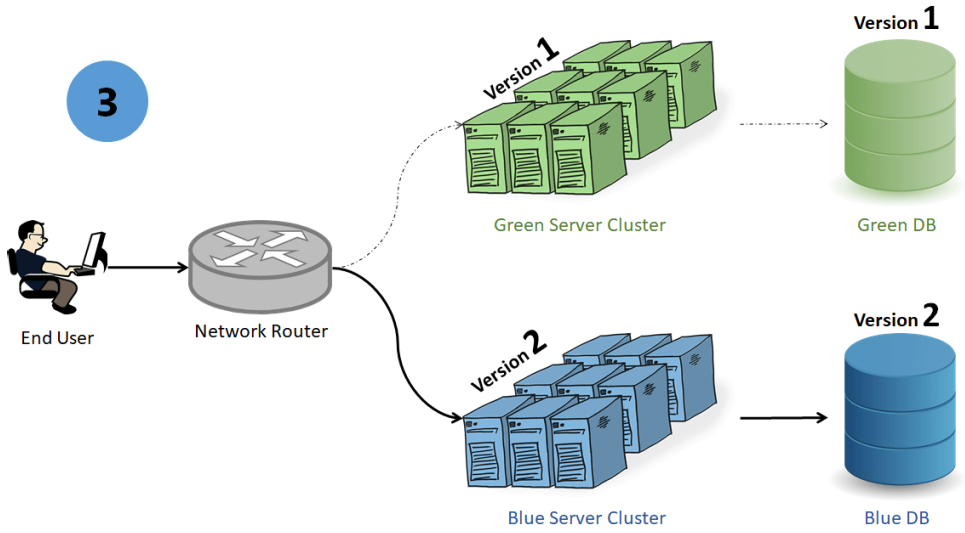
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:v2.0
```

```
Apply this manifest with kubectl apply -f rolling-update-depl
```

---

3. Blue Green:

> Users switch from the old version to the new one without noticing.



- **Definition:** Blue Green deployment involves running two identical production environments (Blue and Green), with only one active at any time. The inactive environment can be updated without affecting users, and then traffic is switched to the updated environment.
- **Pros:**
  - Zero downtime deployment.
  - Easy rollback by switching traffic back to the previous environment.
- **Cons:**
  - Requires double resources to maintain two identical environments.

- **Example:** If version 1.0 of your application is running in the Blue environment, deploying version 2.0 to the Green environment allows you to test the update before switching user traffic.
- **When to use:** Blue Green deployment is useful when you need to ensure minimal downtime and easy rollback capabilities.

Implementing Blue Green deployment involves maintaining two identical environments

▼ Implementation

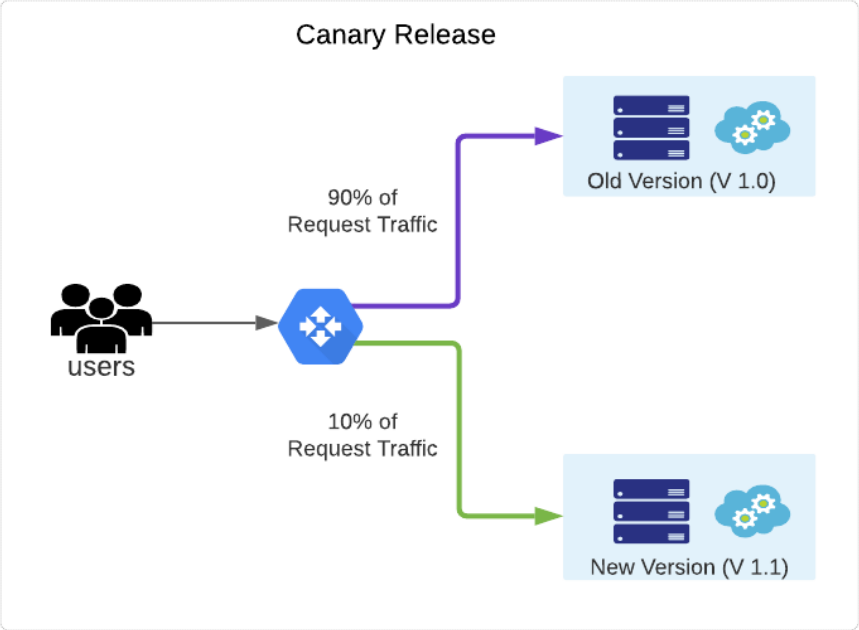
```
apiVersion: v1
kind: Service
metadata:
  name: myapp
spec:
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-blue
spec:
```

```
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
        color: blue
    spec:
      containers:
      - name: myapp
        image: myapp:v1.0

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
        color: green
    spec:
      containers:
      - name: myapp
        image: myapp:v2.0
```

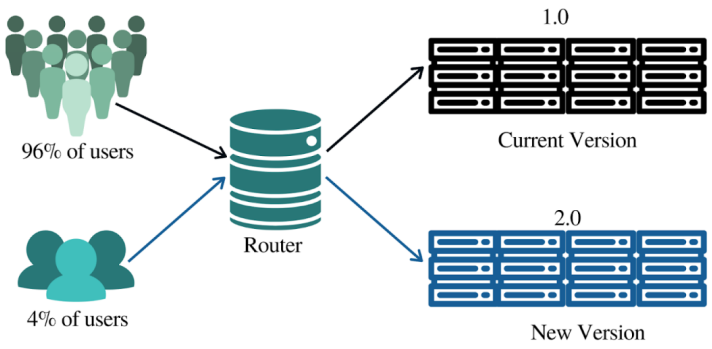
Apply this manifest with `kubectl apply -f blue-green-deployment.yaml`.

4. Canary:

> A few users try out the new version first to see if it works well.



- **Definition:** Canary deployment involves rolling out updates to a small subset of users or servers before rolling it out to the larger group, allowing for testing in a real-world scenario before full deployment.



- **Pros:**
  - Risk mitigation by testing with a small subset of users.
  - Quick detection of issues before affecting all users.
- **Cons:**
  - Requires careful monitoring and management of traffic routing.

- **Example:** Deploying version 2.0 of your application to 10% of your user base initially to observe performance and gather feedback before fully deploying to all users.

- **When to use:** Canary deployment is ideal when you want to test new features or changes with a small subset of users before rolling out to everyone.

> 💡 Deploying a canary release can be done by gradually routing a percentage of traffic to the new version.

▼ Implementation

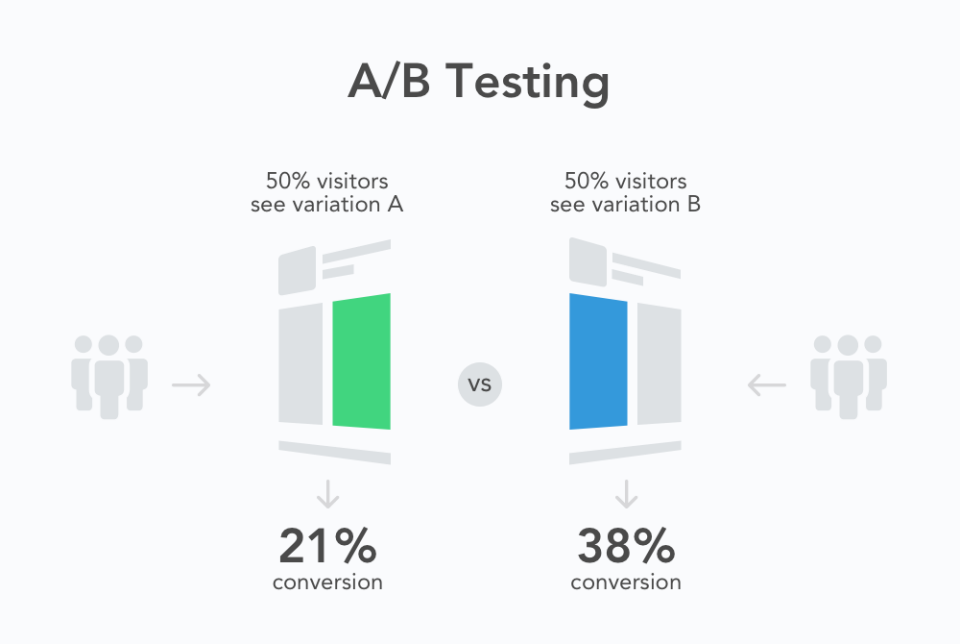
```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: myapp
spec:
  hosts:
  - myapp.example.com
  http:
  - route:
    - destination:
        host: myapp
        subset: v2
      weight: 10
    - destination:
        host: myapp
        subset: v1
      weight: 90
```

Apply this manifest with `kubectl apply -f canary-deployment.yaml`.

---

5. A/B Testing:

> Some users see one version, while others see a different version, to see which one they like better.



- **Definition:** A/B testing involves deploying multiple versions of an application simultaneously and directing different users or requests to each version to compare performance or features.
- **Pros:**
  - Allows for comparison of different versions in real-world scenarios.
  - Enables data-driven decision-making based on user behavior.
- **Cons:**
  - Complexity in managing multiple versions simultaneously.

- **Example:** Deploying version A and version B of a feature to different user segments to measure which version performs better in terms of user engagement or conversion rates.

When to use: A/B testing is appropriate when you want to compare the performance or effectiveness of two different versions of your application.

▼ Implementation

A/B testing is usually implemented within the application's codebase rather than in Kubernetes manifests. Here's a simplified example in Python:

```
if feature_enabled('new_version'):
    # Use new version code
else:
    # Use old version code
```

You would deploy both versions of the code and control which version is executed based on a feature flag or some other condition.

Connect with me :
LinkedIn:
https://www.linkedin.com/in/nasiullha-chaudhari/
Youtube:
https://youtube.com/@cloudchamp?sub_confirmation=1