
WIZELINE
Cloud Native Serverless
#sre-emea

Your Questions during the session

Maybe in the session, we does not leave much time for questions.

Please post your questions to following Google doc

https://drive.google.com/file/d/15PvTVs3EBNVrUvA9TOfy37ImZ4-fQuCz/view?usp=sharing

OR let´s discuss in the team Slack channel **#sre-emea**

Repo → cloudnative-serverless-workshops

Agenda

- About the Cloud native Serverless workshop series.
  - Quick review about previous session.
  - New stuff in the repository
- Getting Context
- Deployment Strategies
  - Rolling Update
  - Canary Deployment
  - Blue Green Deployment
- Serverless Deployment using Knative Traffic management

Getting Context

Workshop Knative Traffic Management

Steps required.

```
> make help
hacking                install tooling required, Kind , Kubectl , make ..
create-cluster         Create Kind Cluster
delete-cluster         Delete Kind Cluster
knative-install        Install all the knative components in the cluster
knative-uninstall      UnInstall all the knative components in the cluster
knative-show           Knative configuration and resources
istio-install          Install the istio component in the cluster
kubeless-install       Install all the kubeless components in the cluster
kubeless-uninstall     UnInstall all the kubeless components in the cluster
monitoring-install     Install monitoring Operator Stacks (Prometheus, Grafana)
monitoring-uninstall   Install monitoring Operator Stacks (Prometheus, Grafana)
knative-serving-workshop-build Build Knative Helloworld-go servicing By Default
knative-serving-workshop-serve Run Knative Helloworld-go servicing By Default .
knative-bluegreen-workshop Run Knative bluegreen
```

https://github.com/erasmodominguezdc/cloudnative-serverless-workshops/tree/deployments

Deployment Considerations

As software Engineers, we must:

- Minimize impact to consumers/clients.
- Provide a rollback and recovery strategy.
- Be fast and robust in the Deployment process.
- Do not affect and impact the whole system.

Not Many years ago….

https://harness.io/blog/continuous-verification/blue-green-canary-deployment-strategies/

Kubernetes deployment

*A Kubernetes deployment is basically a **resource object** in Kubernetes that defines the **desired state** of your application.*

Kubernetes is basically an API with a Database that convert our yaml manifest into objects that represent a desired state

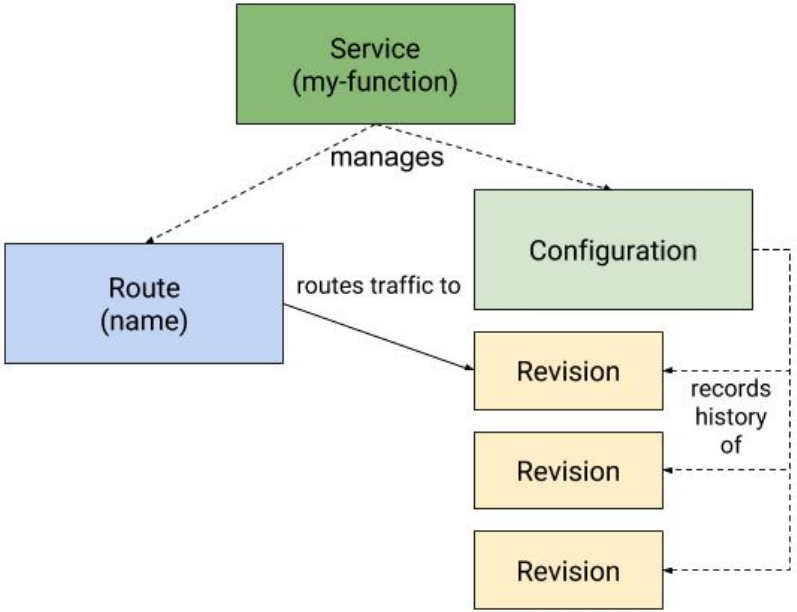
Kubernetes Deployment

```
apiVersion: apps/v1beta1  #Kubernetes API
kind: Deployment
metadata:
 name: hello-deploy
spec:
 replicas: 10
 selector:
   matchLabels:
     app: hello-world
 minReadySeconds: 10
 strategy:
   type: Recreate /RollingUpdate
   rollingUpdate:
     maxUnavailable: 1
     maxSurge: 1
```

Kubernetes deployment

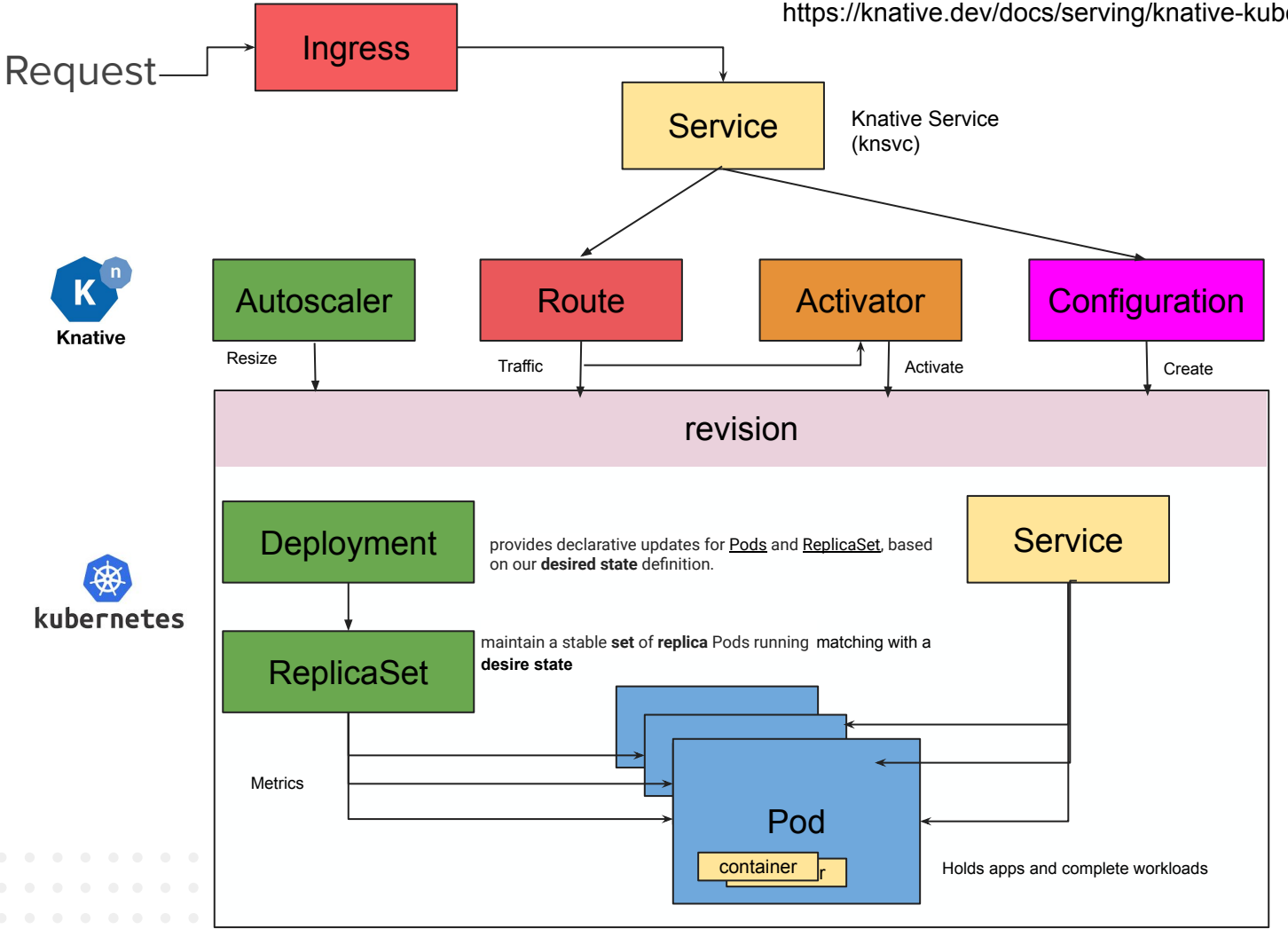
Ok!! Kubernetes is pretty cool , with some YAML hacks I can define my deployments as I need it, but….I just need to manage the traffic!!

Knative Serving Architecture



https://knative.dev/docs/serving/

https://knative.dev/docs/serving/knative-kubernetes-services/



Knative Traffic Management

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: foo
spec:
  template:
     # removed for brevity
  traffic: ❶
  - tag: v1 ❷
    revisionName: foo-v1 ❸
    percent: 50 ❹
  - tag: v2
    revisionName: foo-v2
    percent: 50
```

❶ The `traffic` block to specify the traffic distribution

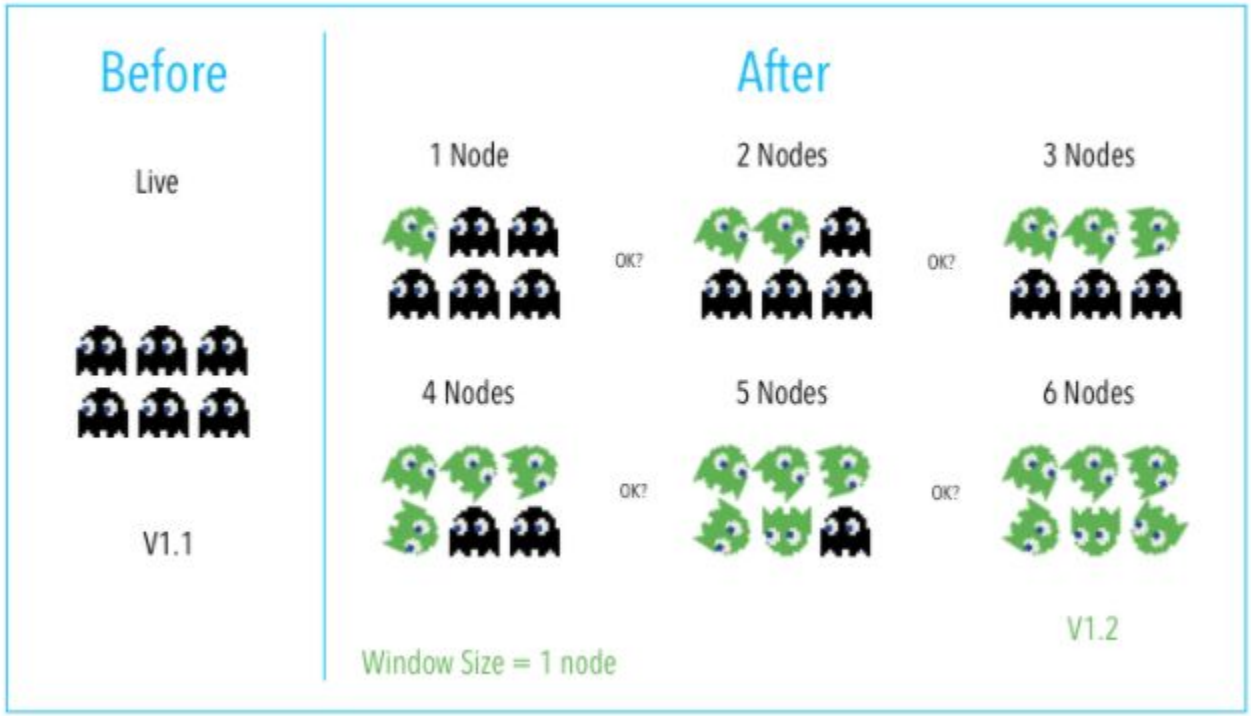
❷ The unique name for this traffic block list item

❸ The Knative Revision that will participate in the traffic distribution

❹ The amount of traffic that the revision will receive; it is a numerical value in percentage

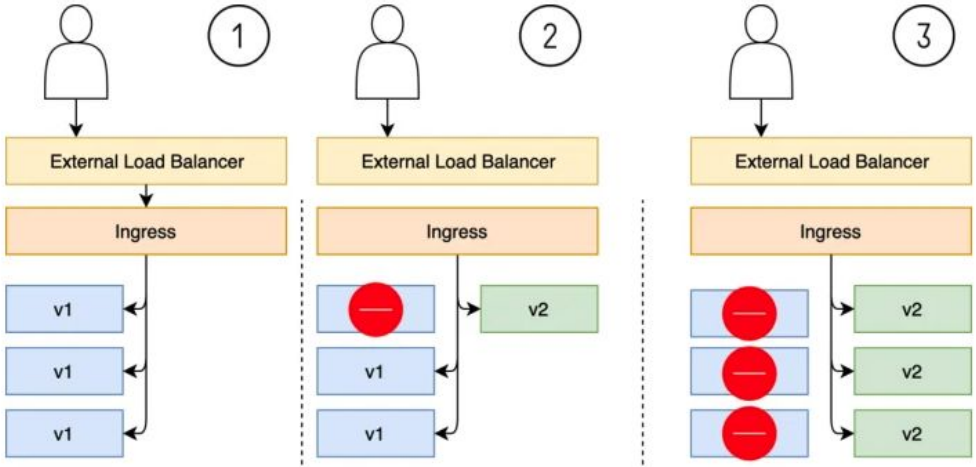
Deployment Strategies

Rolling Deployment (Kubernetes default method)



https://harness.io/blog/continuous-verification/blue-green-canary-deployment-strategies/

Rolling Updates Deployment Strategy



Requests are forwarded from the LB to the Ingress which, in turn, forwards them to all the replicas of the application.

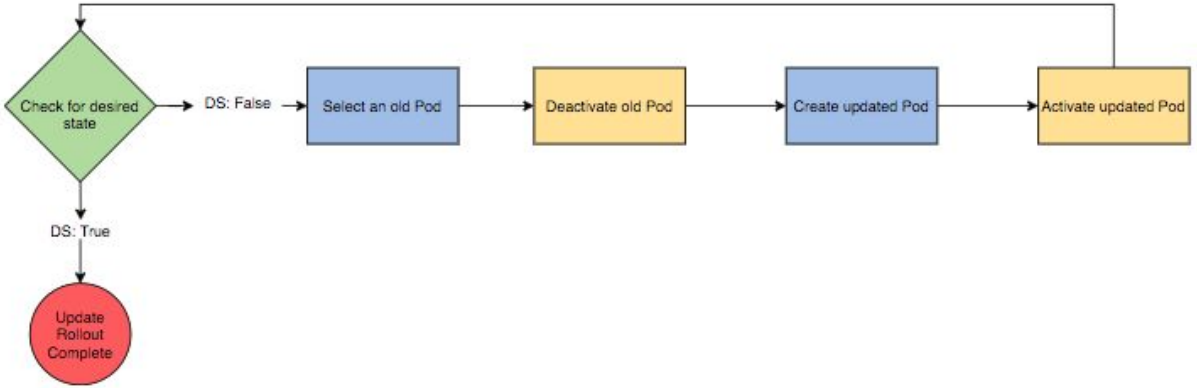
During the deployment of a new release, a replica of the old release is shut down and a replica of the new release is created. Ingress keeps distributing the traffic to all the replicas running (both old and new).

The process continues shutting down the replicas of the old release and creating those based on the new, until only those based on the new release are running.

https://www.cloudbees.com/blog/exploring-deployment-strategies-kubernetes

Rolling update strategy

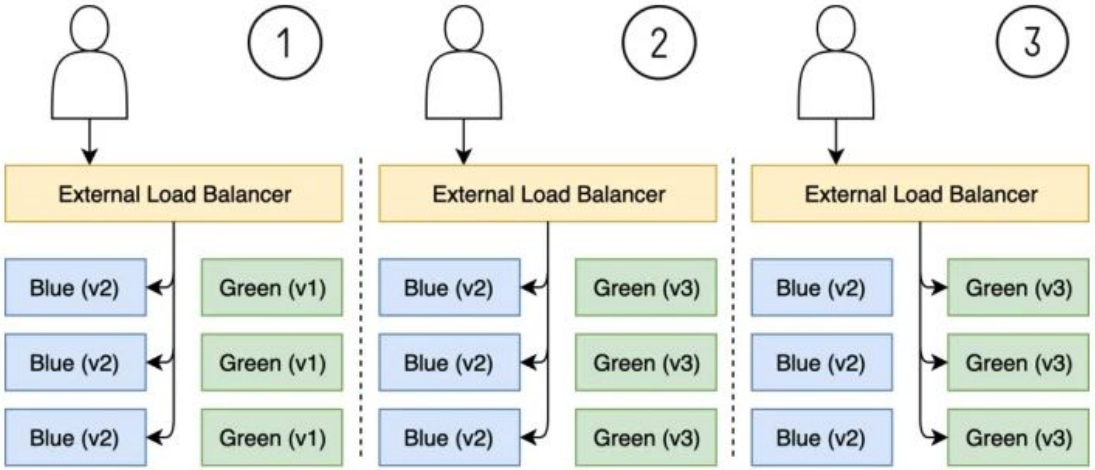
The rolling update strategy is a gradual process that allows you to update your Kubernetes system with only a minor effect on performance and no downtime.



Blue Green Deployment

https://harness.io/blog/continuous-verification/blue-green-canary-deployment-strategies/

Blue-Green Deployments



All the requests are forwarded to the current release (v2, blue). The one before it (v1, green) lays dormant in case we need to revert to it.

A new release is deployed on top of the inactive (unused) release (v1 > v3, green).

The LB is reconfigured to forward all the requests to the new release.

https://www.cloudbees.com/blog/exploring-deployment-strategies-kubernetes

Blue Green Knative Deployment

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: greeter
spec:
  template:
    metadata:
      name: greeter-v2
    spec:
      containers:
        - image: quay.io/rhdevelopers/knative-tutorial-greeter:quarkus
          env:
            - name: MESSAGE_PREFIX
              value: Namaste
          livenessProbe:
            httpGet:
              path: /healthz
          readinessProbe:
            httpGet:
              path: /healthz
  traffic:
    - tag: v1
      revisionName: greeter-v1
      percent: 100
    - tag: v2
      revisionName: greeter-v2
      percent: 0
    - tag: latest
      latestRevision: true
      percent: 0
```

Workshop Knative Traffic Management

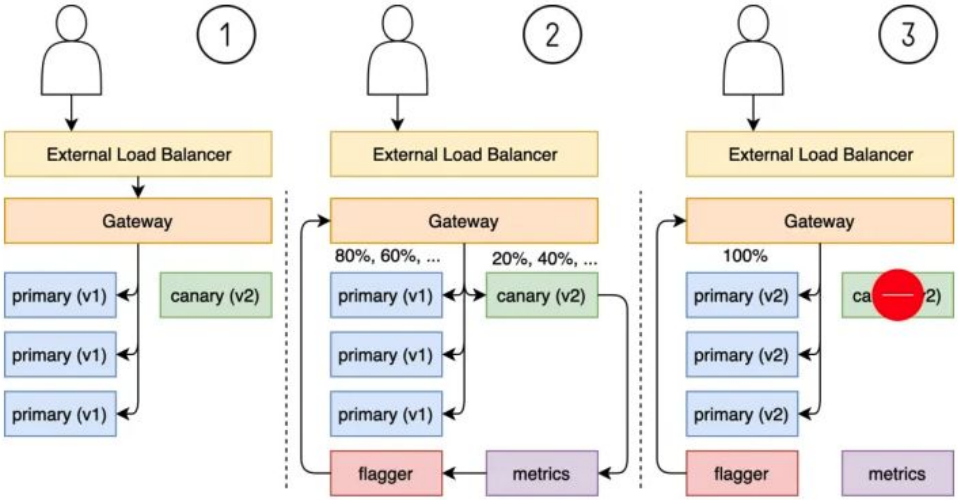
Steps required.

```
kn service update blue-green-canary \
   --image=quay.io/rhdevelopers/blue-green-canary \
   --env BLUE_GREEN_CANARY_COLOR="#5bbf45" \
   --env BLUE_GREEN_CANARY_MESSAGE="Namaste"
```

Canary Deployment

https://harness.io/blog/continuous-verification/blue-green-canary-deployment-strategies/

Canary deployment rollout



Requests are forwarded from the LB to the gateway which, in turn, forwards them to one of the Pods (e.g., round-robin).

When a new release (canary) .is deployed, flagger configures the gateway to send portion of the requests to the new (canary) release and continue sending the rest to the old (primary). Metrics are collected in a database and Flagger uses them to decide whether to proceed with the deployment by increasing the percentage sent to canary and decreasing those that are sent to primary.

When the progressive rollout iterations as finished, primary is updated to the new release and canary is shut down. Flagger reconfigures the gateway so that all the traffic is forwarded to primary replicas.

https://www.cloudbees.com/blog/exploring-deployment-strategies-kubernetes

Canary Knative Deployment

```yaml
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: greeter
spec:
  template:
    metadata:
      name: greeter-v2
    spec:
      containers:
        - image: quay.io/rhdevelopers/knative-tutorial-greeter:quarkus
          env:
            - name: MESSAGE_PREFIX
              value: Namaste
          livenessProbe:
            httpGet:
              path: /healthz
          readinessProbe:
            httpGet:
              path: /healthz
  traffic:
    - tag: v1
      revisionName: greeter-v1
      percent: 80
    - tag: v2
      revisionName: greeter-v2
      percent: 20
    - tag: latest
      latestRevision: true
      percent: 0
```

Workshop Knative Traffic Management

And for more examples →

https://redhat-developer-demos.github.io/knative-tutorial/knative-tutorial/serving/traffic-distribution.html