Docker Deployment Guide

Blue-Green Deployment Model

3/31/2025

Contents

Blue-Green Deployment Model

> **Warning**
> The following content has been deprecated and is maintained for reference only.

Overview

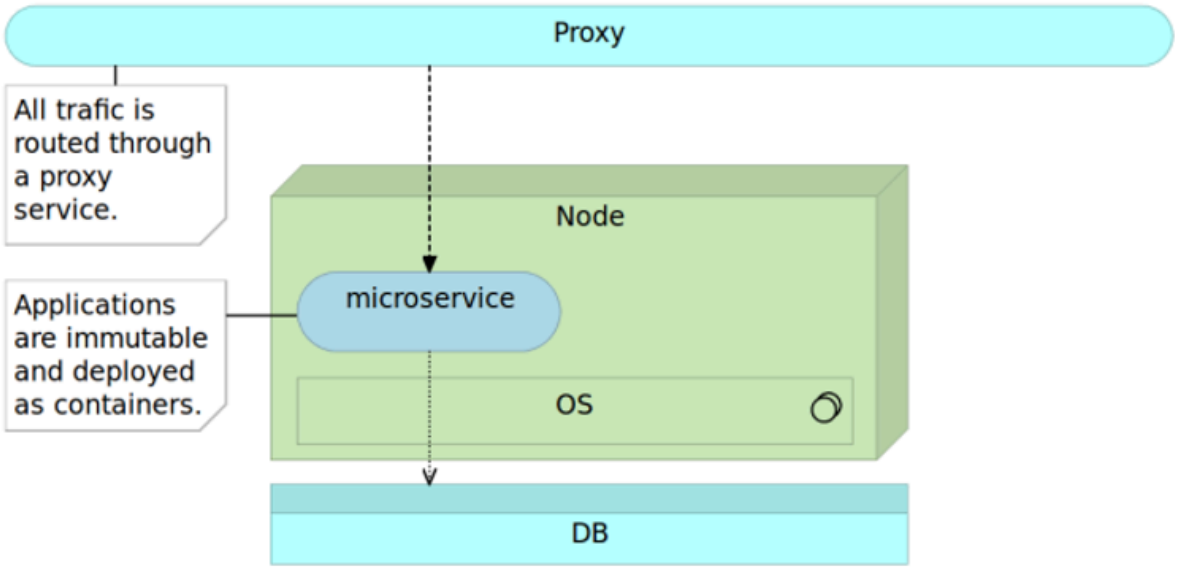
Typically, deploying a new release replaces the current one. You stop the previous release, and then replace it with the new release. The problem with this approach is the downtime that occurs from the moment the previous release is stopped till the new one is fully operational. The Blue-green process removes the deployment downtime and also reduces the risk that the deployment might introduce.

The Blue-Green Deployment Process

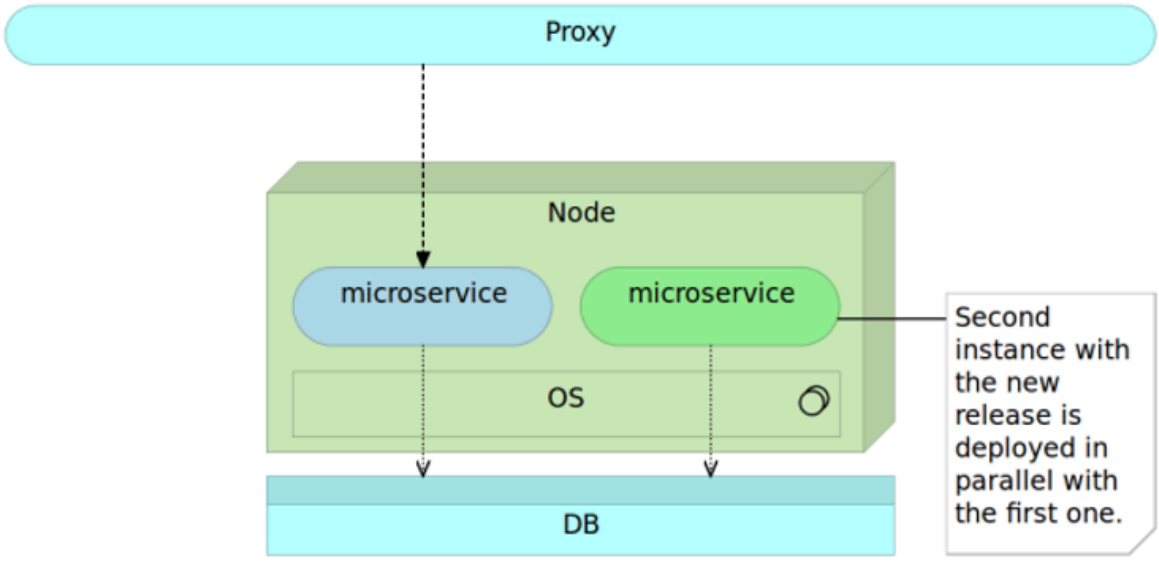
The Blue-Green deployment procedure, when applied to microservices packed as containers, is as follows.

> **Important**
> This example is limited only to microservices and not to the database.

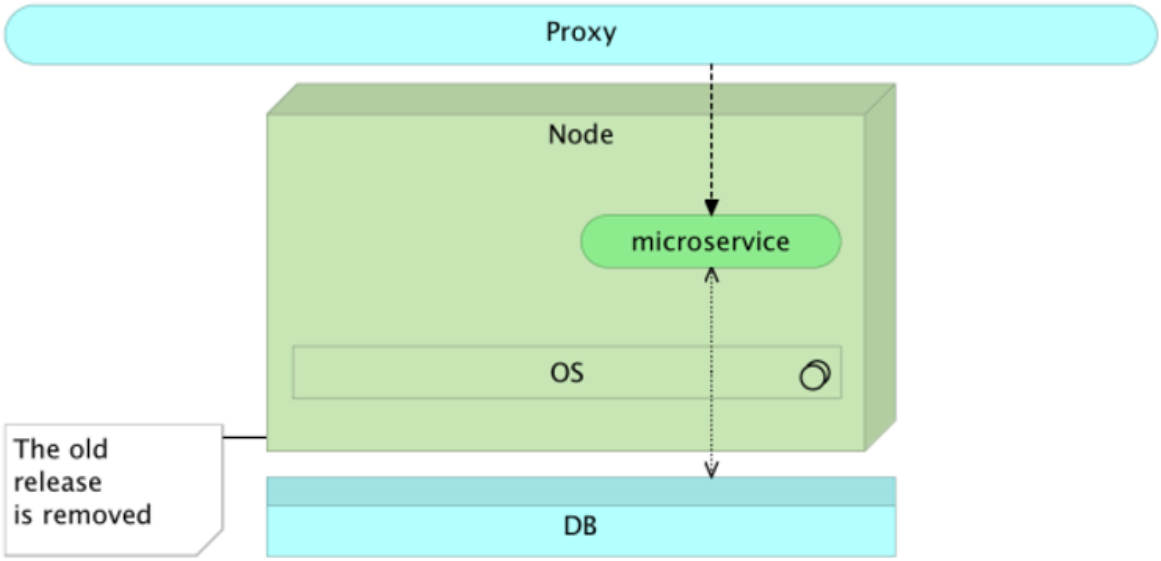
1. When the current release (for example, blue) is running on the server, route all traffic to that release through a proxy service. Microservices are immutable and deployed as containers.



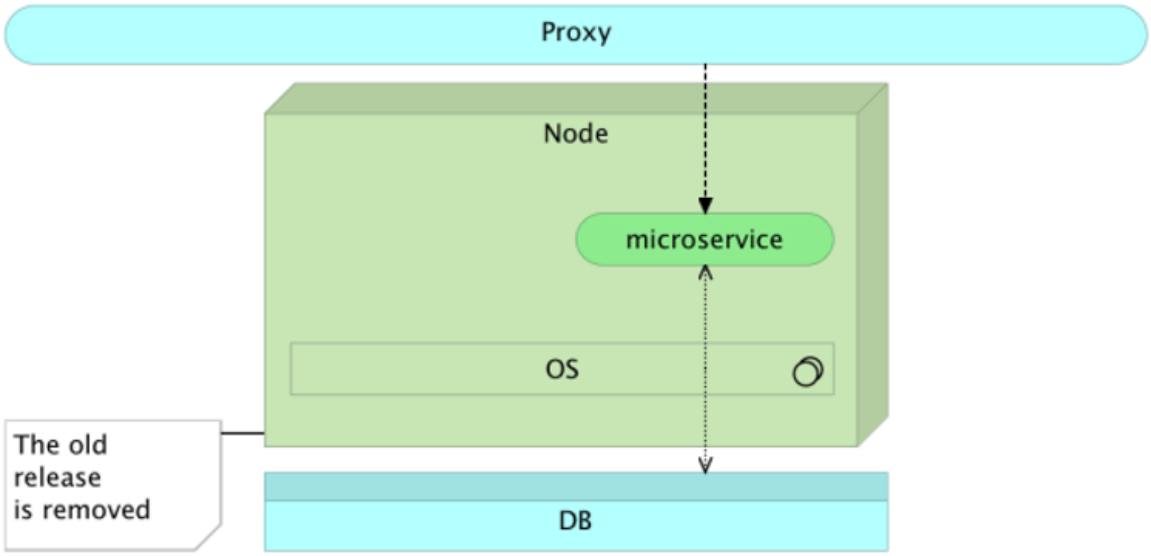
2. When a new release (for example, green) is ready to be deployed, run it in parallel with the current release. This way, you can test the new release without affecting the users since all the traffic continues to be sent to the current release.



3. Once the new release works as expected, change the proxy service configuration to redirect the traffic to the new release. Most of the proxy services will allow the existing requests to complete the execution using the previous proxy configuration to ensure there is no interruption.



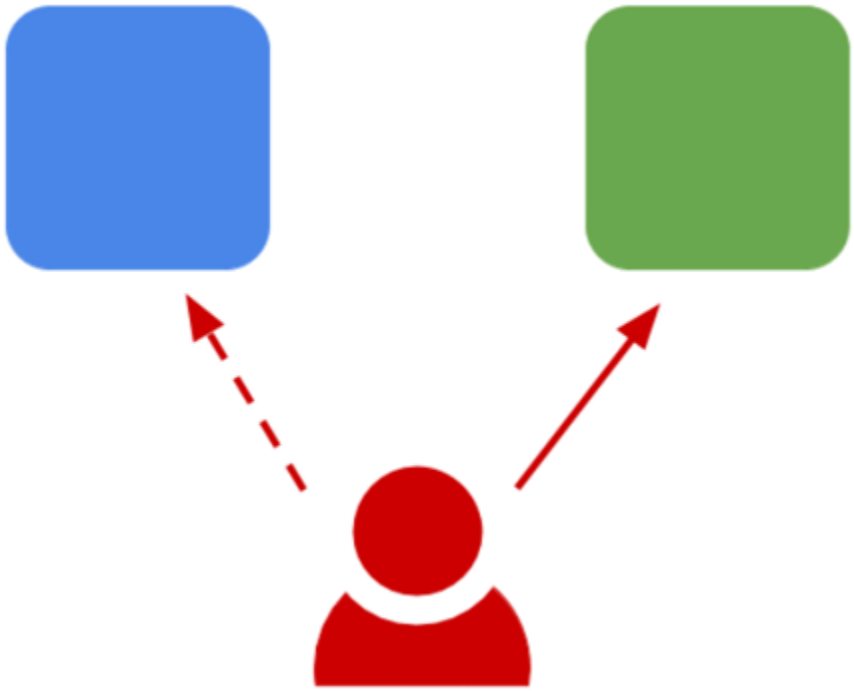
4. When all the requests sent to the previous release receive responses, you can remove or stop the previous version of a service. When you stop the previous version of a service from running, a rollback in case of a failure of the new release will be the instantaneous action so that you can back up the previous release.



Sample Blue-Green Deployment on Kubernetes

When blue-green deployments are performed, a new copy of the application (green) is deployed along with the existing version (blue). The ingress/router to the app is updated to switch to the new version (green). You must wait for the previous (blue) version to complete the requests sent to it.

However, for the most part, traffic to the app changes to the new version, instantly.



Kubernetes does not contain built-in support for blue-green deployments. Currently, the best way to support deployments is to create new deployment, and then update the service for the application to point to the new deployment. This section contains a sample blue-green deployment implemented on a Kubernetes cluster.

The Blue Deployment

A Kubernetes deployment specifies a group of instances of an application. At the back end, it creates a replicaset that is responsible for keeping the specified number of instances up and running.

You can create your blue deployment by saving the following `yaml` to a `blue.yaml` file.

```
apiVersion: extensions/v1beta1

kind: Deployment

metadata:

  name: nginx-1.10

spec:

  replicas: 3
```

```
  template:
   metadata:
    labels:
     name: nginx
     version: "1.10"
   spec:
    containers:
     - name: nginx
       image: nginx:1.10
       ports:
        - name: http
          containerPort: 80
```

Create the deployment using the `kubectl` command:

```
$ kubectl apply -f blue.yaml
```

After the deployment, you can provide a way to access the instances of the deployment by creating a Service. Services are decoupled from deployments, which means that you do not explicitly point a service at a deployment. Instead, you specify a label selector that is used to list the pods that make up the service. When using deployments, this is typically set up so that it matches the pods for a deployment.

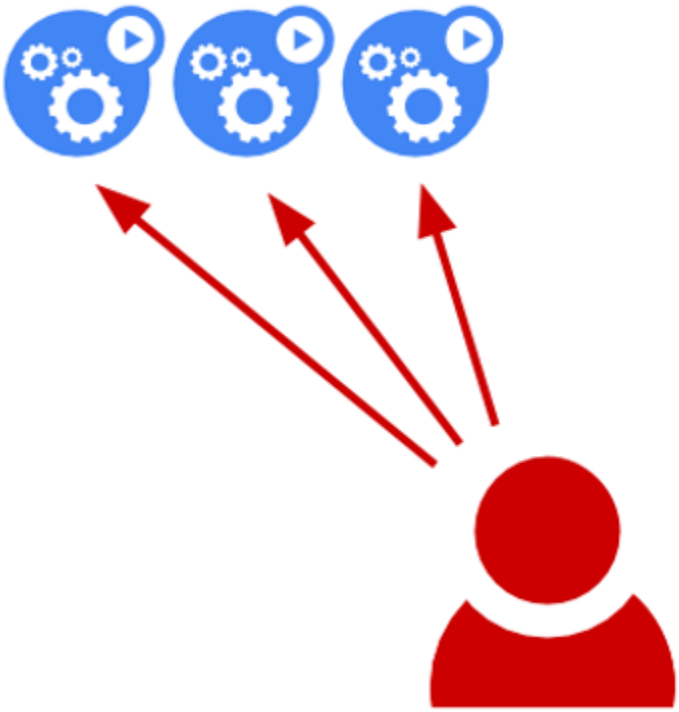
In this case, you have two labels, `name=nginx` and `version=1.10`. You will set them as the label selector for the following service. Save this to `service.yaml`.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels
```

```
    name: nginx
  specs:
    ports
    - name: http
      Port: 80
      targetPort: 80
    selector
    name: nginx
    version: "1.10"
    type: LoadBalancer
```

Creating the service creates a load balancer that is accessible outside the cluster.

```
$ kubectl apply -f service.yaml
```



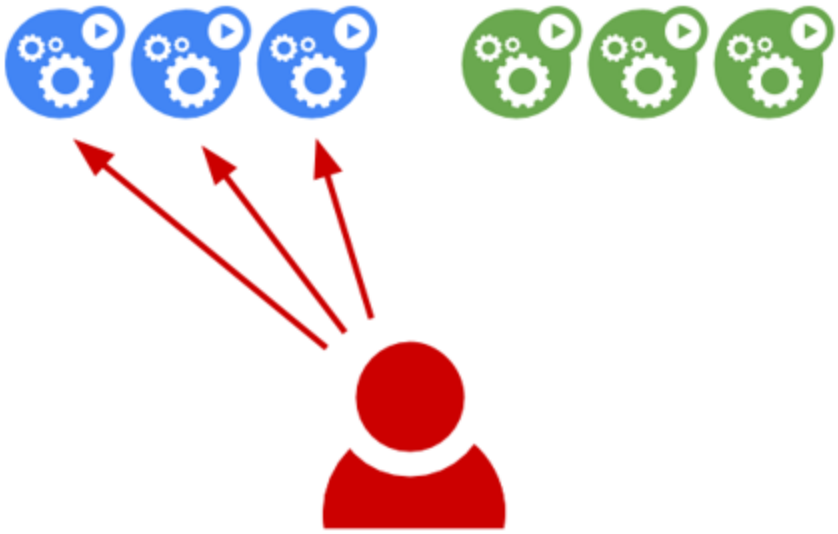
The Green Deployment

For the green deployment, you will perform a new deployment in parallel with the "blue" deployment. The following service is saved as `green.yaml`.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-1.11
spec:
  replicas: 3
  template:
    metadata:
     labels:
      name: nginx
      version: "1.11"
    spec:
     containers:
      - name: nginx
      image: nginx:1.11
      ports:
       - name: http
        containerPort: 80

$ kubectl apply -f green.yaml
```

Now there are two deployments, but the service still points to the "blue" one.



The Cut-Over

To cut over to the "green" deployment, you must update the selector for the service. Edit `service.yaml` and then change the selector version to "1.11". This matches the pods on the "green" deployment.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
   name: nginx
spec:
  ports:
   - name: http
     port: 80
     pargetPort: 80
  selector:
```

```
    name: nginx
    version: "1.11"
    type: LoadBalancer
```

The following **apply** command will update the existing `nginx` service.

```
$ kubectl apply -f service.yaml
```

Now the service appears as follows:

Updating the selector for the service is applied immediately. Therefore, you can see that the new version of nginx will be serving the traffic.

```
$ EXTERNAL_IP=$(kubectl get svc nginx -o
jsonpath="{.status.loadBalancer.ingress[*].ip}")
$ curl -s http://$EXTERNAL_IP/version | grep nginx
```

Terminating and Deleting Blue Deployment

The Blue deployment will stop receiving further requests because the service points to the Green deployment. However, it is necessary that the Blue deployment is terminated gracefully after it terminates the current serving connections.

This can be done using one of the following methods depending on the scenario:

- Terminate with grace-period
- Use preStop hook

Terminate with grace-period can be done by specifying a grace period while deleting the deployment:

```
kubectl delete deployment blue --grace-period=120
```

The grace-period setting can also be specified at the pod spec level as follows:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: test
spec:
  replicas: 1
  template:
   spec:
    containers:
     - name: test
      image: ...
  terminationGracePeriodSeconds: 60
```

The **preStop hook** is configured at the container level and allows execution of a custom command before SIGTERM (signal sent to a running process to end the process) is sent. The termination grace period countdown starts before invoking the preStop hook and not after the SIGTERM signal is sent.

The following example, shows how to configure a preStop command:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx/tt>
spec:
```

```
    template:
     metadata:
      labels:
       app: nginx
     spec:
      containers:
      - name: nginx
       image: nginx
       ports:
       - containerPort: 80
       lifecycle:
        preStop:
         exec:
          # SIGTERM triggers a quick exit; gracefully terminate
   instead
          command: ["/usr/sbin/nginx","-s","quit"]
```

Automating

You can automate the blue/green deployment to some extent with scripting. The following script uses the name of the service, version you want to deploy, and path to the green deployment's **yaml** file. Then, it runs through a full blue/green deployment process using **kubectl** to send a raw JSON as the output from the API and parsing it with the **jq** script. It waits for the green deployment to become ready by inspecting **status.conditions** on the deployment object before updating the service definition.

```
#!/bin/bash

# bg-deploy.sh <servicename> <version> <green-deployment.yaml> # Deployment name
should be <service>-<version>

DEPLOYMENTNAME=$1-$2 SERVICE=$1 VERSION=$2 DEPLOYMENTFILE=$3

kubectl apply -f $DEPLOYMENTFILE
```

```
# Wait until the Deployment is ready by checking the MinimumReplicasAvailable
condition. READY=$(kubectl get deploy $DEPLOYMENTNAME -o json | jq
'.status.conditions[] | select(.reason == "MinimumReplicasAvailable") | .status' | tr
-d '"') while [[ "$READY" != "True" ]]; do

    READY=$(kubectl get deploy $DEPLOYMENTNAME -o json | jq
  '.status.conditions[] | select(.reason ==
  "MinimumReplicasAvailable") | .status' | tr -d '"')

    sleep 5

done

# Update the service selector with the new version kubectl patch svc $SERVICE -p
"{\"spec\":{\"selector\": {\"name\": \"${SERVICE}\", \"version\": \"${VERSION}\"}}}"

echo "Done."
```