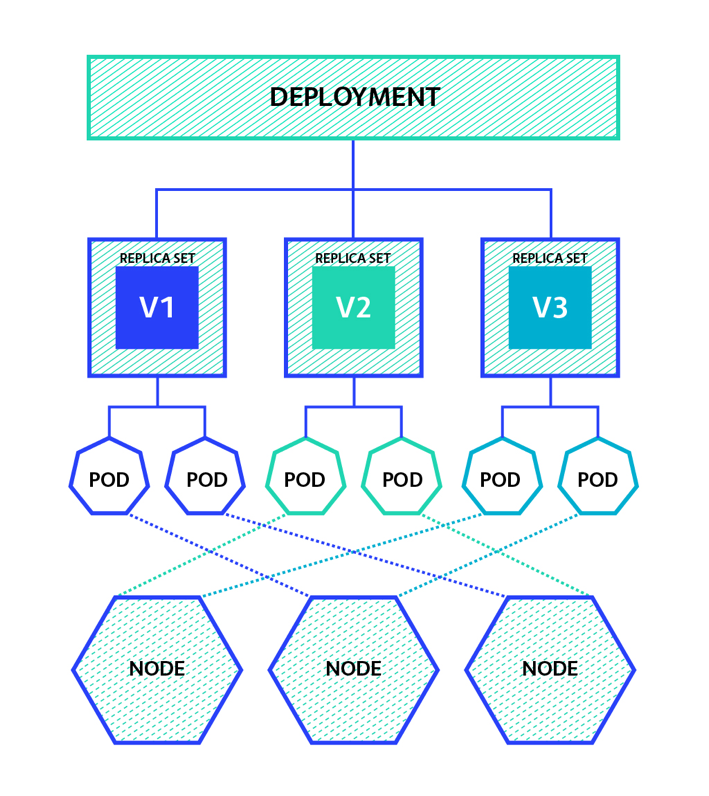
Kubernetes hands on series: Deployments: Live example of Rolling Out and Rolling Back updates with Zero downtime



<img alt="Image for post" class="s t u dt ai" src="https://miro.medium.com/max/1418/1\*wS9vaFCxZZYRpY9XsB9mkQ.png" width="709" height="796" srcSet="https://miro.medium.com/max/552/1\*wS9vaFCxZZYRpY9XsB9mkQ.png 276w, https://miro.medium.com/max/1104/1\*wS9vaFCxZZYRpY9XsB9mkQ.png 552w, https://miro.medium.com/max/1280/1\*wS9vaFCxZZYRpY9XsB9mkQ.png 640w, https://miro.medium.com/max/1400/1\*wS9vaFCxZZYRpY9XsB9mkQ.png 700w" sizes="700px"/>

For the purpose of this tutorial, we assume that you have a healthy 3-node Kubernetes cluster already been provisioned. Follow this tutorial to [spin up a Production ready 3 Node kubernetes cluster](https://medium.com/@rakeshjain_17559/production-ready-kubernetes-cluster-deployment-on-ubuntu-8cccdf952774).

**What is Deployment ?**

Its one of the Kubernetes object.

Deployments are upgraded and higher version of replication controller. They manage the deployment of replica sets which is also an upgraded version of the replication controller.

Its suggested to useDeployment instead of Replication Controller(rc) to perform a rolling update. Though, they are same in many ways, such as ensuring the homogeneous set of pods are always up/available and also they provide the ability to help the user to roll out the new images. However, Deployment provides more functionalities such as rollback support.

**Users expect applications to be available all the time and developers are expected to deploy new versions of them several times a day.**

In Kubernetes this is done with rolling updates. **Rolling updates** allow Deployments’ update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources.

**We will understand this with an very simple example:-**

We have a sample application image at: gcr.io/google-samples/hello-app:1.0

**Here is our Deployment definition file.**

|  |  |
| --- | --- |
|  | apiVersion: apps/v1 |
|  | kind: Deployment |
|  | metadata: |
|  | name: my-dep |
|  | namespace: default |
|  | spec: |
|  | replicas: 2 |
|  | selector: |
|  | matchLabels: |
|  | app: my-dep |
|  | template: |
|  | metadata: |
|  | labels: |
|  | app: my-dep |
|  | spec: |
|  | containers: |
|  | - image: gcr.io/google-samples/hello-app:2.0 |
|  | imagePullPolicy: Always |
|  | name: my-dep |
|  | ports: |
|  | - containerPort: 8080 |

[**view raw**](https://gist.github.com/TroubleshooteRJ/f15dc0278f4bf7a2ecc9d07a255b9175/raw/dcba441dc2f5163b4cf646e795378fd69ac02813/deployment-definition.yaml)

[**deployment-definition.yaml**](https://gist.github.com/TroubleshooteRJ/f15dc0278f4bf7a2ecc9d07a255b9175#file-deployment-definition-yaml) hosted with ❤ by [**GitHub**](https://github.com/)

<https://gist.github.com/TroubleshooteRJ/f15dc0278f4bf7a2ecc9d07a255b9175>

**Here is our Service definition file.**

|  |
| --- |
| apiVersion: v1 |
| kind: Service |
| metadata: |
| name: mydep-service |
| spec: |
| type: NodePort |
| ports: |
| - port: 8080 #service port |
| targetPort: 8080 #Pod Port |
| nodePort: 30088 #Node Port from the range - 30000-32767 |
|  |
| selector: |
| app: my-dep |

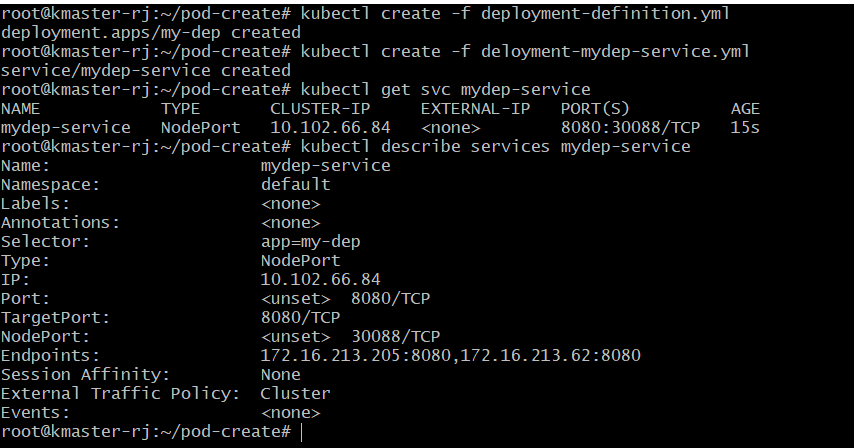
[**view raw**](https://gist.github.com/TroubleshooteRJ/90752cddc0dd3265f182aa9016147caa/raw/a1386d662f4befbac56e9735f2f72dc75f7103f2/deloyment-mydep-service.yml)

[**deloyment-mydep-service.yml**](https://gist.github.com/TroubleshooteRJ/90752cddc0dd3265f182aa9016147caa#file-deloyment-mydep-service-yml) hosted with ❤ by [**GitHub**](https://github.com/)

<https://gist.github.com/TroubleshooteRJ/90752cddc0dd3265f182aa9016147caa>

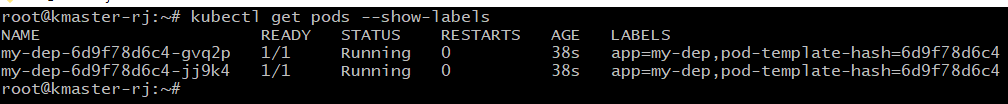
**Now Lets provision the deployment along with the service.**Follow the images to understand this.



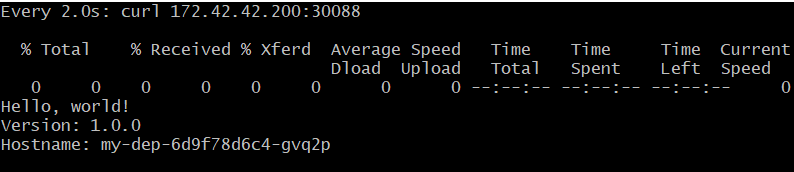


Here are the pods created and application access output using service NodePort (30088) on cmd line as well as from UI on our host machine .. (We will discuss more about service deployments in next tutorial)

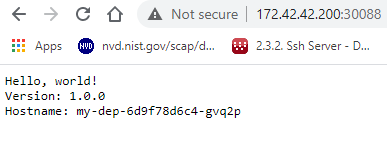




minikube service mydep-service –url







**Now its time to Roll Out a new update to our application.**

|  |  |
| --- | --- |
|  | apiVersion: apps/v1 |
|  | kind: Deployment |
|  | metadata: |
|  | name: my-dep |
|  | namespace: default |
|  | spec: |
|  | replicas: 2 |
|  | strategy: |
|  | type: RollingUpdate |
|  | rollingUpdate: |
|  | maxUnavailable: 25% |
|  | maxSurge: 1 |
|  | selector: |
|  | matchLabels: |
|  | app: my-dep |
|  | template: |
|  | metadata: |
|  | labels: |
|  | app: my-dep |
|  | spec: |
|  | containers: |
|  | - image: gcr.io/google-samples/hello-app:2.0 |
|  | imagePullPolicy: Always |
|  | name: my-dep |
|  | ports: |
|  | - containerPort: 8080 |
|  | readinessProbe: |
|  | httpGet: |
|  | path: / |
|  | port: 8080 |
|  | initialDelaySeconds: 5 |
|  | periodSeconds: 5 |
|  | successThreshold: 1 |

[**view raw**](https://gist.github.com/TroubleshooteRJ/52c481357324d3476f244c73ea2ee495/raw/0bf2f33f2c82afbdf678b12cebd497fcd8af4ff9/deployment-rollingupdate.yaml)

[**deployment-rollingupdate.yaml**](https://gist.github.com/TroubleshooteRJ/52c481357324d3476f244c73ea2ee495#file-deployment-rollingupdate-yaml) hosted with ❤ by [**GitHub**](https://github.com/)

<https://gist.github.com/TroubleshooteRJ/52c481357324d3476f244c73ea2ee495>

**Lets first understand this updated file -**

If you simply update your original deployment-definition.yaml file with the new image version and apply that then you will notice that there maybe a little downtime on your application because the old pods are getting terminated and the new ones are getting created.

**Why this happens and whats the solution ?**

This happens because Kubernetes doesn’t know when your new pod is ready to start accepting requests, so as soon as your new pod gets created, the old pod is terminated without waiting to see if all the necessary services, processes have started in the new pod which would then enable it to receive requests.

To do this, Kubernetes provide a config option in deployment called **Readiness Probe**. Readiness Probe makes sure that the new pods created are ready to take on requests before terminating the old pods. To enable this, first you need to have a route in whatever the application you want to run which would return a 200 on an HTTP GET request. (Note: you can have other HTTP request methods as well, but for this post, I’m sticking with GET method)

**Probes have a number of fields that you can use to more precisely control the behavior of liveness and readiness checks:**

* initialDelaySeconds: Number of seconds after the container has started before liveness or readiness probes are initiated. Defaults to 0 seconds. Minimum value is 0.
* periodSeconds: How often (in seconds) to perform the probe. Default to 10 seconds. Minimum value is 1.
* timeoutSeconds: Number of seconds after which the probe times out. Defaults to 1 second. Minimum value is 1.
* successThreshold: Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1. Must be 1 for liveness. Minimum value is 1.
* failureThreshold: When a probe fails, Kubernetes will try failureThreshold times before giving up. In case of readiness probe the Pod will be marked Unready. Defaults to 3. Minimum value is 1.

**Another thing we should add is something called RollingUpdate strategy and it can be configured as follows.**

strategy:

type: RollingUpdate

rollingUpdate:

maxUnavailable: 25%

maxSurge: 1

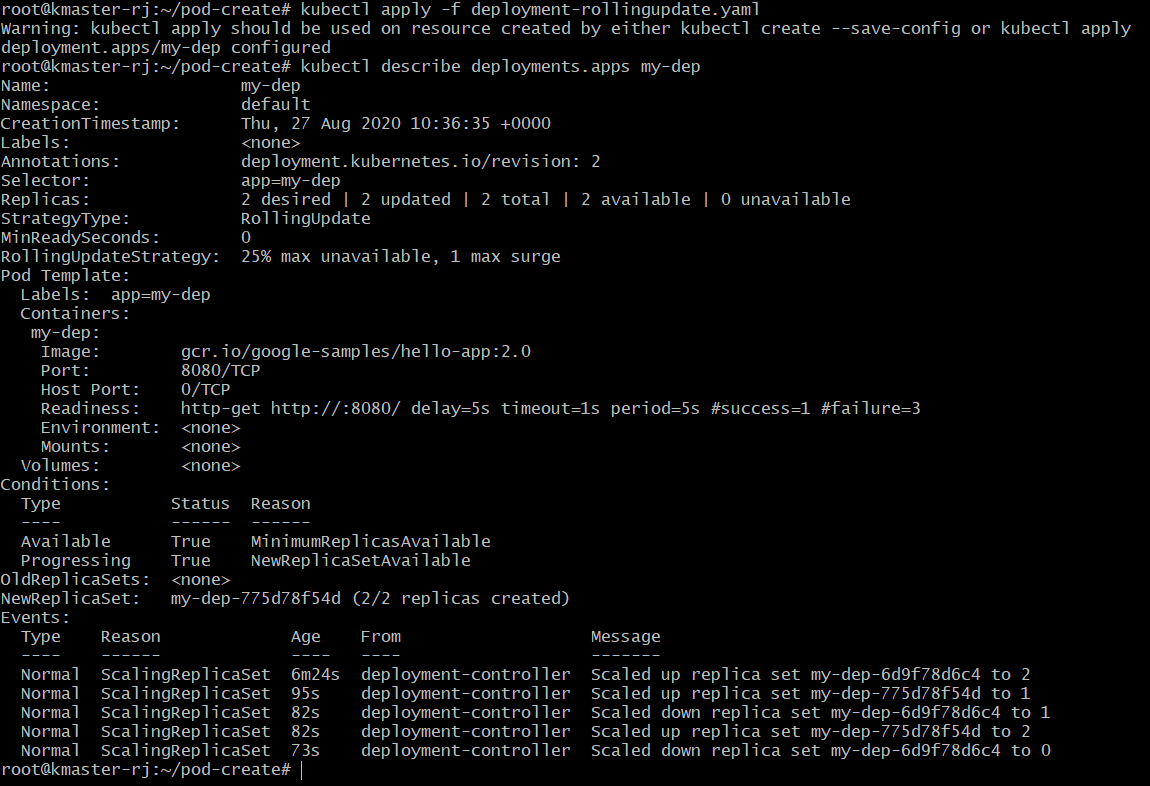
The above specifies the strategy used to replace old Pods by new ones. The type can be “Recreate” or “RollingUpdate”. “RollingUpdate” is the default value. **So it should be configured along with rediness/liveness probe settings because the Kubernetes doesnt know at what time your pod is ready, so you might have a downtime due to that.**

**maxUnavailable** is an optional field that specifies the maximum number of Pods that can be unavailable during the update process. The value can be an absolute number (for example, 5) or a percentage of desired Pods (for example, 10%). The absolute number is calculated from percentage by rounding down. The value cannot be 0 if **maxSurge** is 0. The default value is 25%.

**maxSurge** is an optional field that specifies the maximum number of Pods that can be created over the desired number of Pods. The value can be an absolute number (for example, 5) or a percentage of desired Pods (for example, 10%). The value cannot be 0 if **MaxUnavailable** is 0. The absolute number is calculated from the percentage by rounding up. The default value is 25%.

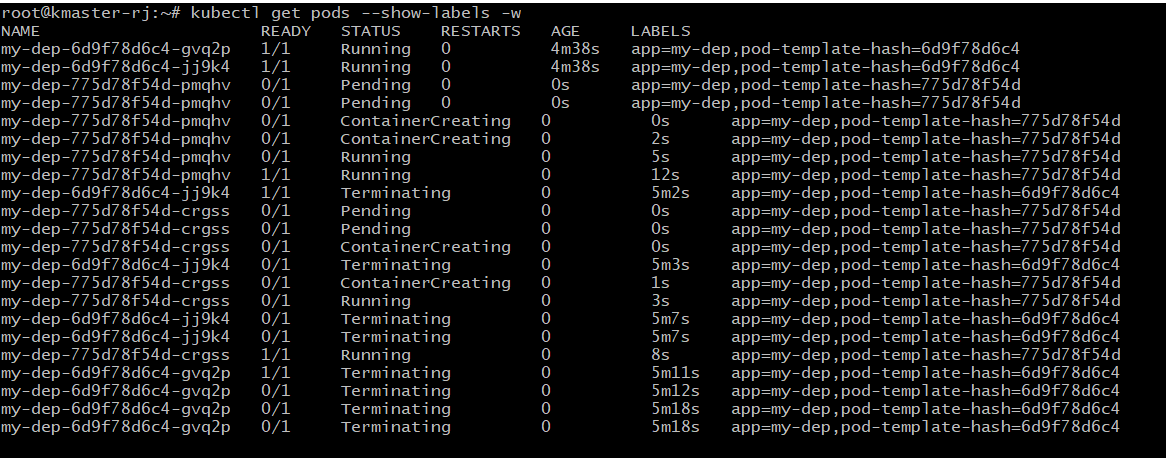
**So we have understood our updated file to rollout a new update to our application. Lets apply it now.**





**Lets look at existing pods going to termination state while new pods are getting live ..**

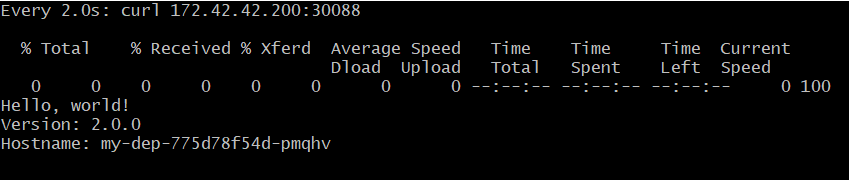




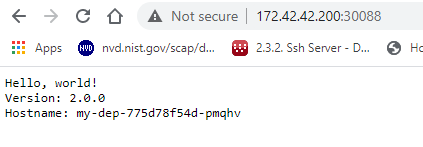
**Now verify if our application is updated to newer version or not –**

minikube service --url mydep-service







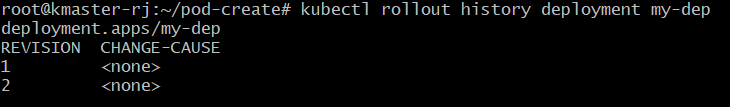


<img alt="Image for post" class="s t u dt ai" src="https://miro.medium.com/max/846/1\*mzAylAQA8xzzIuXJQKiWtA.png" width="423" height="147" srcSet="https://miro.medium.com/max/552/1\*mzAylAQA8xzzIuXJQKiWtA.png 276w, https://miro.medium.com/max/846/1\*mzAylAQA8xzzIuXJQKiWtA.png 423w" sizes="423px"/>

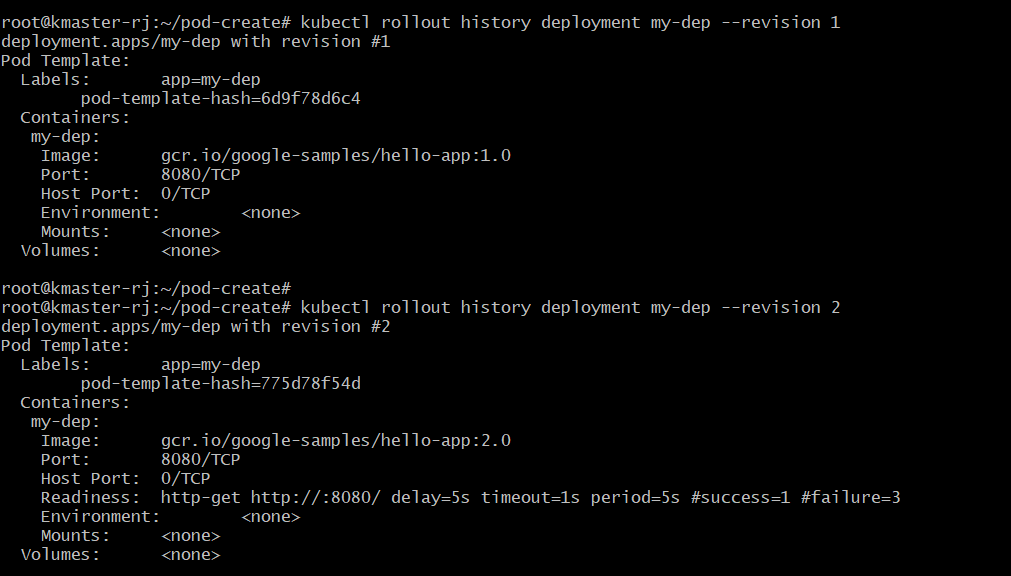
It has been. wow :)

Look at the rollout history and the details about each revision ..





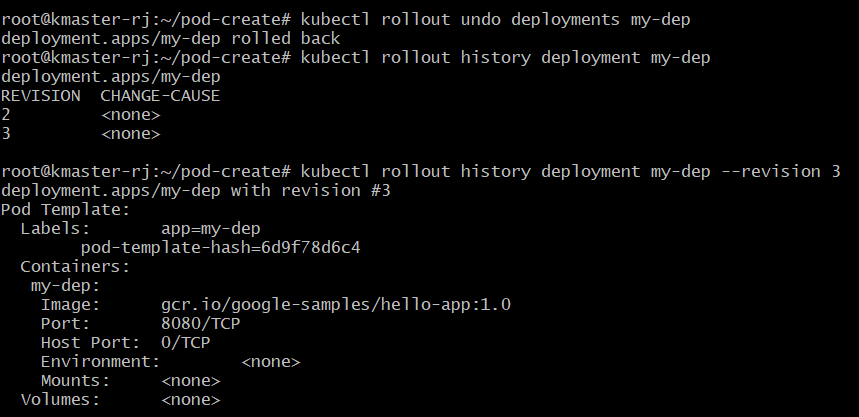




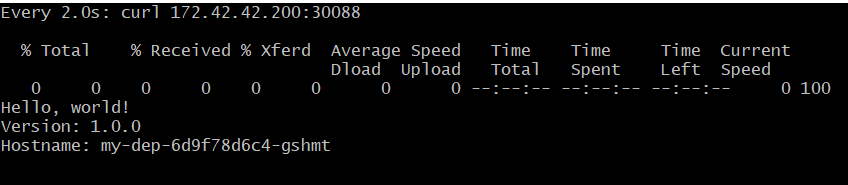
Oh No! Wait ! We did something wrong with the new version and because of that our customers are getting impacted .. Lets ROLLBACK! But How ??

**Here is the answer :-**Rollback to last revision

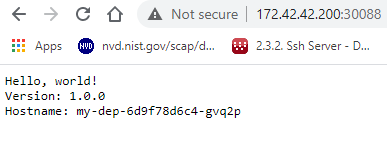








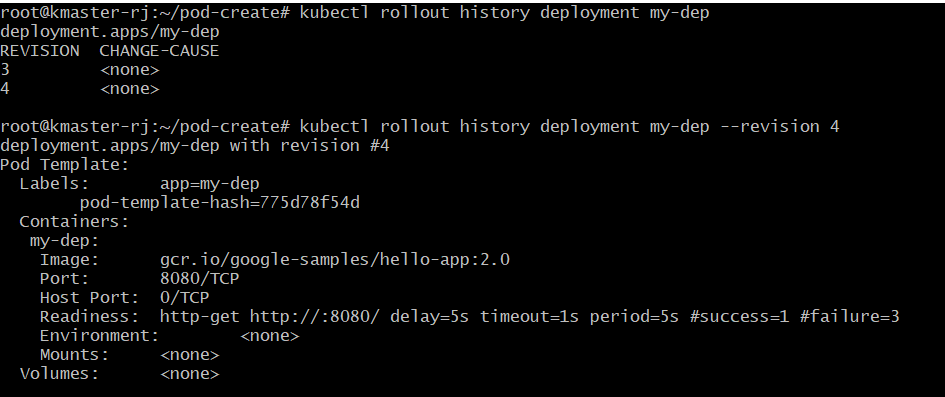




**That’s Cool isn’t it.**

**Now lets learn how we can go to a specific revision of rollout history.**





**Commands to remember -**

**To rollback to the third revision of the Deployment, run the following command:**

#kubectl rollout undo deployments my-dep --to-revision=3

**To rollback to the previous version of the my-dep Deployment, run the following command:**

#kubectl rollout undo deployments my-dep

**To see the history of the third revision, run the following command:**

#kubectl rollout history deployment my-dep --revision 3

**To view the my-dep Deployment's rollout history, run the following command:**

**#**kubectl rollout history deployment **my-dep**

Hope you like the tutorial. **Feel free to clap for this article** and let me know your feedback in the response section.

Happy Learning!