Balavignesh-manoharan

Kubectl Port Forward with Hands-on Examples

Introduction:

In this article, we will explore Port Forwarding and What it does. How kubectl Port Forward makes kubernetes application accessible on your local machine without exposing them externally. Additionally, I also have included a hands-on demo with three practical use cases.

Let’s dive in!

Overview:

- What is Port Forwarding ?
- What is SSH Port Forward ?
- What is Kubectl Port Forward ?
- Demo

What is Port Forwarding ?

Port Forwarding is a method or a mechanism in which you map a particular port of your local machine to the remote machine. These machines should be in a same network or they should be able to communicate with each other.

What is SSH Port Forward ?

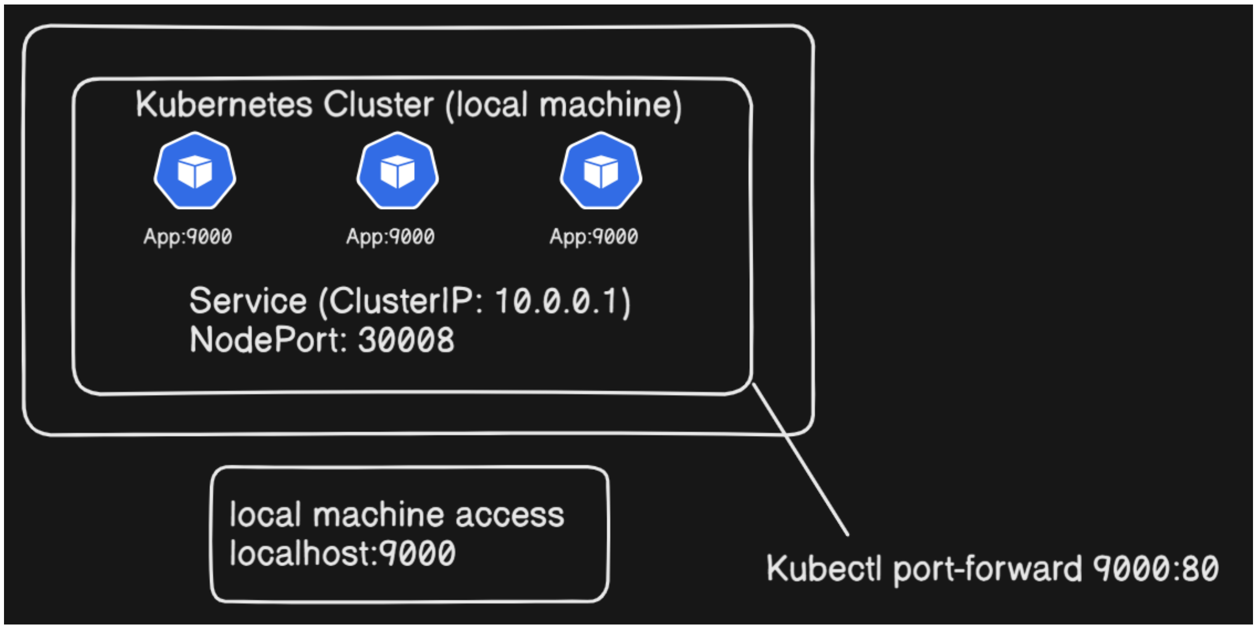
SSH Port Forward is a mechanism where data is transported securely over an encrypted SSH connection. The forwarding is is done through a connection or mapping from a local port on your machine through an encrypted tunnel to a remote machine.

Basically it creates a tunnel between local and remote machine. This allows data to travel securely over an encrypted SSH connection.

What is Kubectl Port Forward ?

This concept is inspired by SSH Port Forwarding. As both serve the purpose of forwarding the traffic from one system to another, without directly exposing them over the network. So Kubernetes provides the kubectl port forward command to establish a secure tunnel between the local machine and a pod within the cluster.

Suppose you have a your k8s cluster inside your local machine. Now if you want to access the application, then you can do it directly using the ClusterIP within the cluster. To expose this application outside the cluster and make it accessible on the Node's IP then it can be done using the node port service. But if you want to access the same application on a port of your local machine then a mapping needs to be done between a port on the local machine with port on the k8s cluster. Only then you can access using localhost on your browser.


Kubernetes Cluster (local machine)
App:9000
App:9000
App:9000
Service (ClusterIP: 10.0.0.1)
NodePort: 30008
local machine access
localhost:9000
Kubectl port-forward 9000:80

Let us see this practically.

We will create a kubernetes cluster locally using **minikube/kind** and run an application. Later we will see about how port forwarding works.

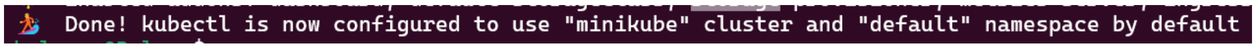
Steps:

USE CASE 1

Running from your local machine

- Start your minikube cluster. If you haven't installed minikube, click here and install according to your OS.
- Once installed, start your minikube cluster.

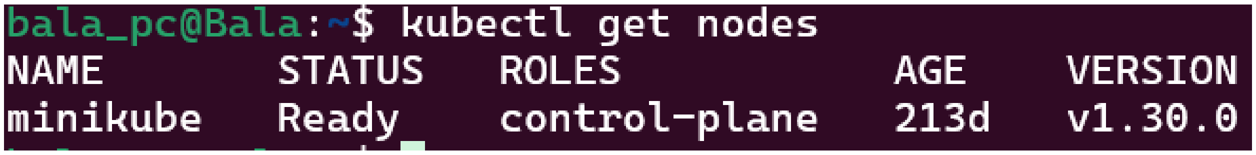
```
minikube start
```



```
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

- It might take a minute or two, then run the below command to check the nodes.

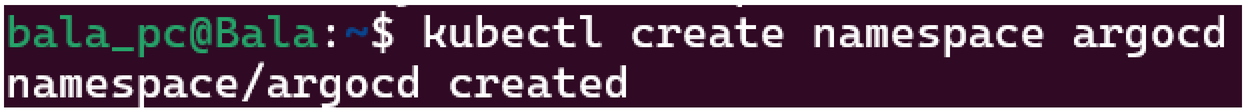
```
kubectl get nodes
```



```
bala_pc@Bala:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE    VERSION
minikube   Ready    control-plane   213d   v1.30.0
```

- We have our kubernetes cluster ready, let us run an application.
- Here we will run an ArgoCD application. The steps are mentioned in the documentation here.

```
kubectl create namespace argocd
```



```
bala_pc@Bala:~$ kubectl create namespace argocd
namespace/argocd created
```

This command creates a name space argocd.

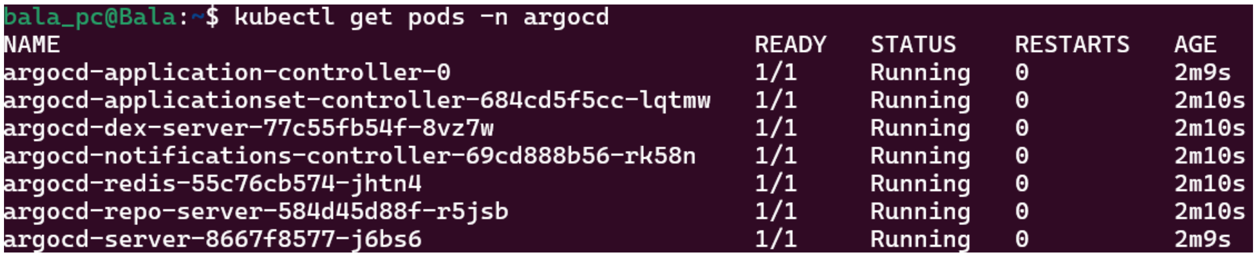
- Run the next command.

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/arg
```

This command creates a lot of resources of argocd.

- Check if all the pods are running.

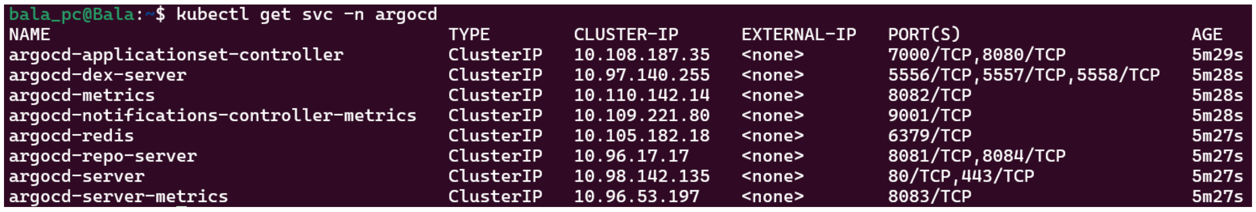
```
kubectl get pods -n argocd
```



```
bala_pc@Bala:~$ kubectl get pods -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          2m9s
argocd-applicationset-controller-684cd5f5cc-lqtmw   1/1     Running   0          2m10s
argocd-dex-server-77c55fb54f-8vz7w                  1/1     Running   0          2m10s
argocd-notifications-controller-69cd888b56-rk58n    1/1     Running   0          2m10s
argocd-redis-55c76cb574-jhtn4                       1/1     Running   0          2m10s
argocd-repo-server-584d45d88f-r5jsb                 1/1     Running   0          2m10s
argocd-server-8667f8577-j6bs6                       1/1     Running   0          2m9s
```

- Check for the services in the argocd.

```
kubectl get svc -n agrocd
```

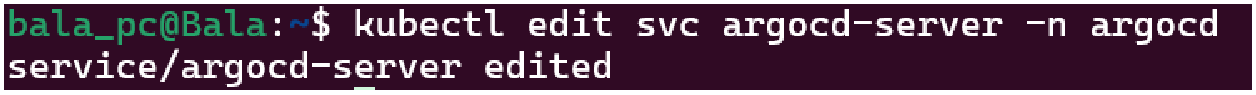


```
bala_pc@Bala:~$ kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   10.108.187.35    <none>        7000/TCP,8080/TCP            5m29s
argocd-dex-server                         ClusterIP   10.97.140.255    <none>        5556/TCP,5557/TCP,5558/TCP   5m28s
argocd-metrics                            ClusterIP   10.110.142.14    <none>        8082/TCP                     5m28s
argocd-notifications-controller-metrics   ClusterIP   10.109.221.80    <none>        9001/TCP                     5m28s
argocd-redis                              ClusterIP   10.105.182.18    <none>        6379/TCP                     5m27s
argocd-repo-server                        ClusterIP   10.96.17.17      <none>        8081/TCP,8084/TCP            5m27s
argocd-server                             ClusterIP   10.98.142.135    <none>        80/TCP,443/TCP               5m27s
argocd-server-metrics                     ClusterIP   10.96.53.197     <none>        8083/TCP                     5m27s
```

- Now to access ArgoCD from my browser, it is not possible with ClusterIP service. So let us change the service type to NodePort.

```
kubectl edit svc argocd-server -n argocd
```

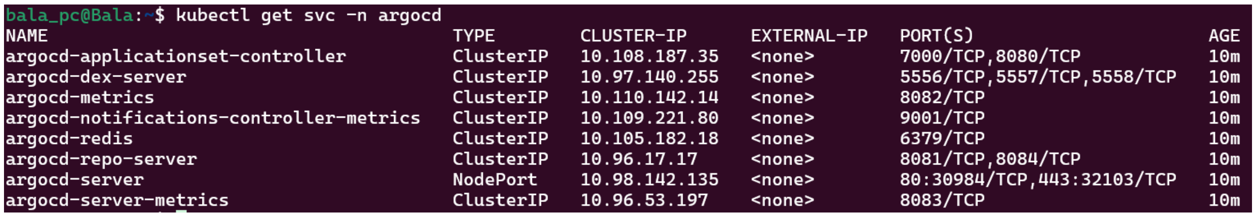
Scroll down to the service type and change it to NodePort and save the file.



```
bala_pc@Bala:~$ kubectl edit svc argocd-server -n argocd
service/argocd-server edited
```

- Check your service IP again to see if the type has changed to NodePort

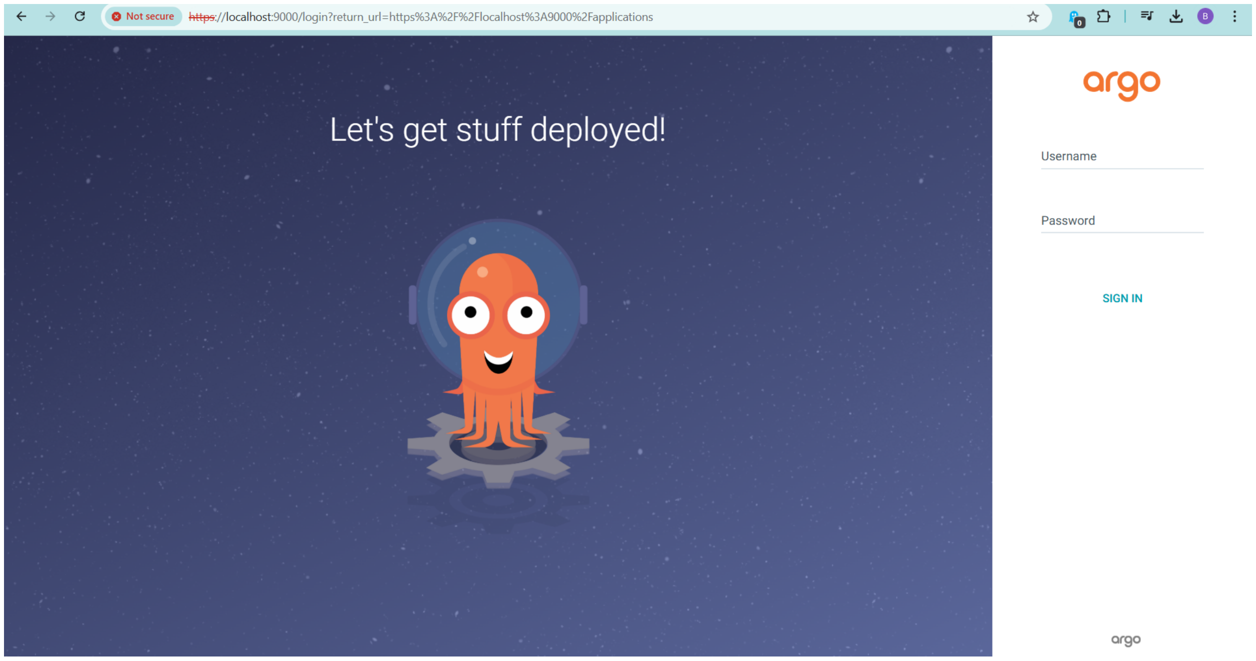
```
kubectl get svc -n argocd
```



```
bala_pc@Bala:~$ kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   10.108.187.35   <none>        7000/TCP,8080/TCP            10m
argocd-dex-server                         ClusterIP   10.97.140.255   <none>        5556/TCP,5557/TCP,5558/TCP   10m
argocd-metrics                            ClusterIP   10.110.142.14   <none>        8082/TCP                     10m
argocd-notifications-controller-metrics   ClusterIP   10.109.221.80   <none>        9001/TCP                     10m
argocd-redis                              ClusterIP   10.105.182.18   <none>        6379/TCP                     10m
argocd-repo-server                        ClusterIP   10.96.17.17     <none>        8081/TCP,8084/TCP            10m
argocd-server                             NodePort    10.98.142.135   <none>        80:30984/TCP,443:32103/TCP   10m
argocd-server-metrics                     ClusterIP   10.96.53.197    <none>        8083/TCP                     10m
```

- Now if you try to access the browser with `localhost:30984`, you will see the page is not getting loaded.
- Since our application is inside the kubernetes cluster and this is running inside a container. We are trying to access it from local machine. It is actually a different server, so for this to work we will set up a kubectl port forward.
- To setup a port forward, just run the below command.

```
kubectl port-forward svc/argocd-server 9000:80
```


Not secure
https://localhost:9000/login?return_url=https%3A%2F%2Flocalhost%3A9000%2Fapplications
Let's get stuff deployed!
argo
Username
Password
SIGN IN
argo

🎉 We've successfully port-forwarded the service, and the application is up and running on the browser!

USE CASE 2

Running from AWS EC2 machine

Spin Up an AWS EC2 Instance. Select t2.medium as the Instance Type.

- Once the instance is up and running, we need to install docker. As Minikube uses a container runtime to create and manage kubernetes clusters.
- Connect to your instance and first update your instance.

```
sudo apt update
```

- Install docker using the below command.

```
sudo apt install docker.io
```

- Add your user to the docker group, which allows the user to run docker commands without using sudo every time.

```
sudo usermod -aG docker ubuntu
newgrp docker
```

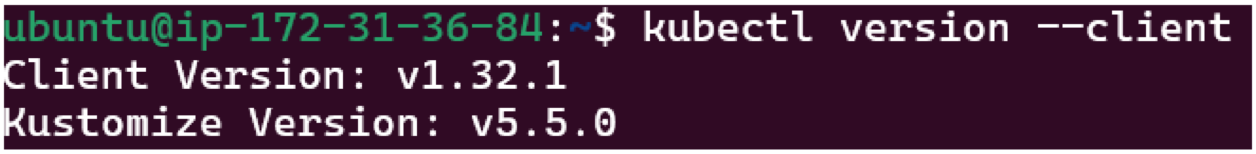
- Install minikube. Download from here

```
curl -LO https://github.com/kubernetes/minikube/releases/latest/
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm
```

This is for the Linux OS, you can select your OS accordingly to install.

- Install kubectl from the Official docs download.
- After installation, check the version.

```
kubectl version --client
```



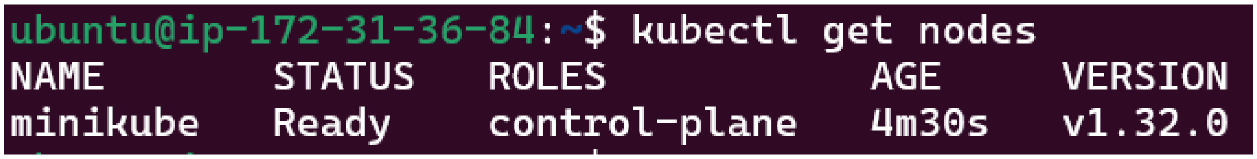
```
ubuntu@ip-172-31-36-84:~$ kubectl version --client
Client Version: v1.32.1
Kustomize Version: v5.5.0
```

- Start your Cluster.

```
minikube start
```

- Check the Nodes.

```
kubectl get nodes
```



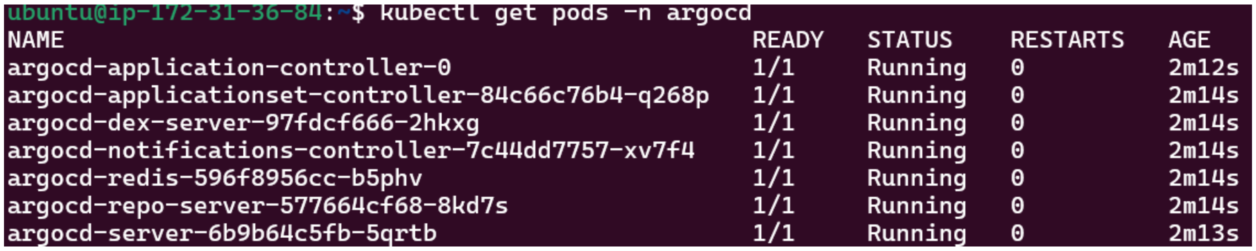
```
ubuntu@ip-172-31-36-84:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE     VERSION
minikube   Ready    control-plane   4m30s   v1.32.0
```

- We will run an ArgoCD Application. First create a namespace and install the argocd resources. Official Docs here.

```
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/ar
```

- Check if all the Pods are running.

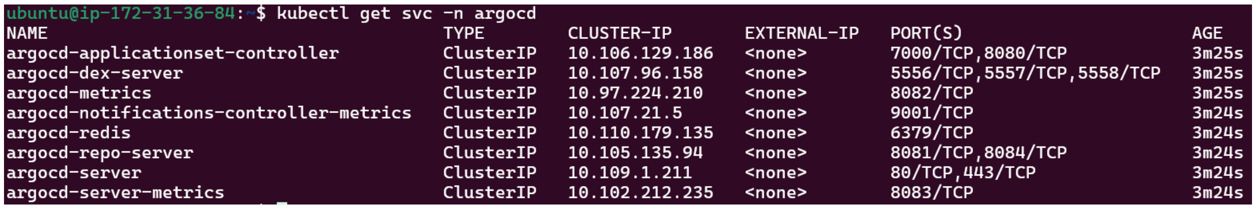
```
kubectl get pods -n argocd
```



```
ubuntu@ip-172-31-36-84:~$ kubectl get pods -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          2m12s
argocd-applicationset-controller-84c66c76b4-q268p   1/1     Running   0          2m14s
argocd-dex-server-97fdcf666-2hkxg                   1/1     Running   0          2m14s
argocd-notifications-controller-7c44dd7757-xv7f4   1/1     Running   0          2m14s
argocd-redis-596f8956cc-b5phv                       1/1     Running   0          2m14s
argocd-repo-server-577664cf68-8kd7s                 1/1     Running   0          2m14s
argocd-server-6b9b64c5fb-5qrtb                      1/1     Running   0          2m13s
```

- Get the service and edit the service type to NodePort.

```
kubectl get svc -n argocd
```



```
ubuntu@ip-172-31-36-84:~$ kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   10.106.129.186   <none>        7000/TCP,8080/TCP            3m25s
argocd-dex-server                         ClusterIP   10.107.96.158    <none>        5556/TCP,5557/TCP,5558/TCP   3m25s
argocd-metrics                            ClusterIP   10.97.224.210    <none>        8082/TCP                     3m25s
argocd-notifications-controller-metrics   ClusterIP   10.107.21.5      <none>        9001/TCP                     3m24s
argocd-redis                              ClusterIP   10.110.179.135   <none>        6379/TCP                     3m24s
argocd-repo-server                        ClusterIP   10.105.135.94    <none>        8081/TCP,8084/TCP            3m24s
argocd-server                             ClusterIP   10.109.1.211     <none>        80/TCP,443/TCP               3m24s
argocd-server-metrics                     ClusterIP   10.102.212.235   <none>        8083/TCP                     3m24s
```

- Edit the service type using the below command.

```
kubectl edit svc argocd-server -n argocd
```

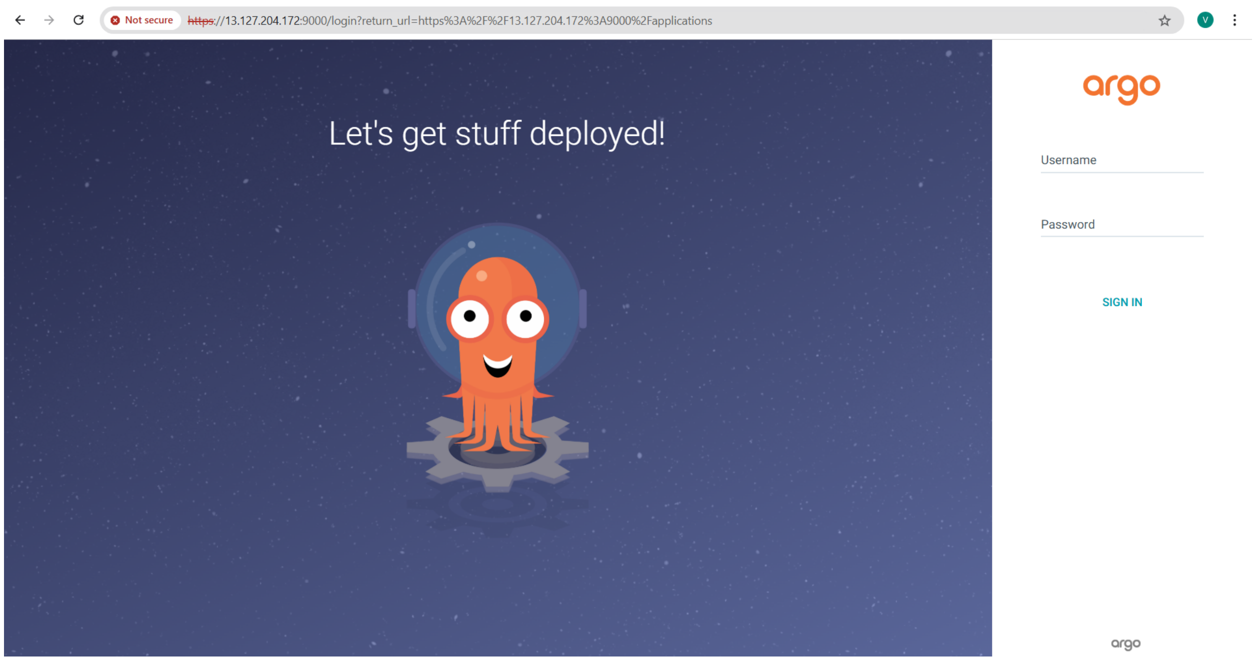
- Once edited, save the file and check if the type has changed into NodePort.
- Run the Port-forward command.

```
kubectl port-forward svc/argocd-server 9000:80 -n argocd
```

Here is the catch! If you try to access your application with the Public IP with port `9000`. You will see the app is not loading. This happens because the default kubectl port-forward command binds only to the localhost. To access from the external source like EC2 instance, we need to modify the command slightly to include the `–address=0.0.0.0` option.

```
kubectl port-forward svc/argocd-server 9000:80 --address=0.0.0.
```

- Now copy your Public IP with port 9000 and you can see ArgoCD UI running successfully.


Not secure
https://13.127.204.172:9000/login?return_url=https%3A%2F%2F13.127.204.172%3A9000%2Fapplications
Let's get stuff deployed!
argo
Username
Password
SIGN IN
argo

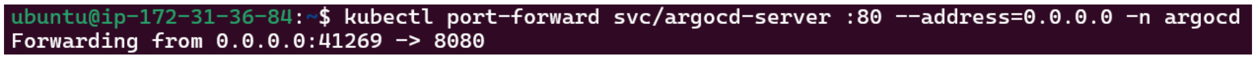
🚀 **Success**! The service has been successfully port-forwarded.

USE CASE 3

Automatic Port Assignment

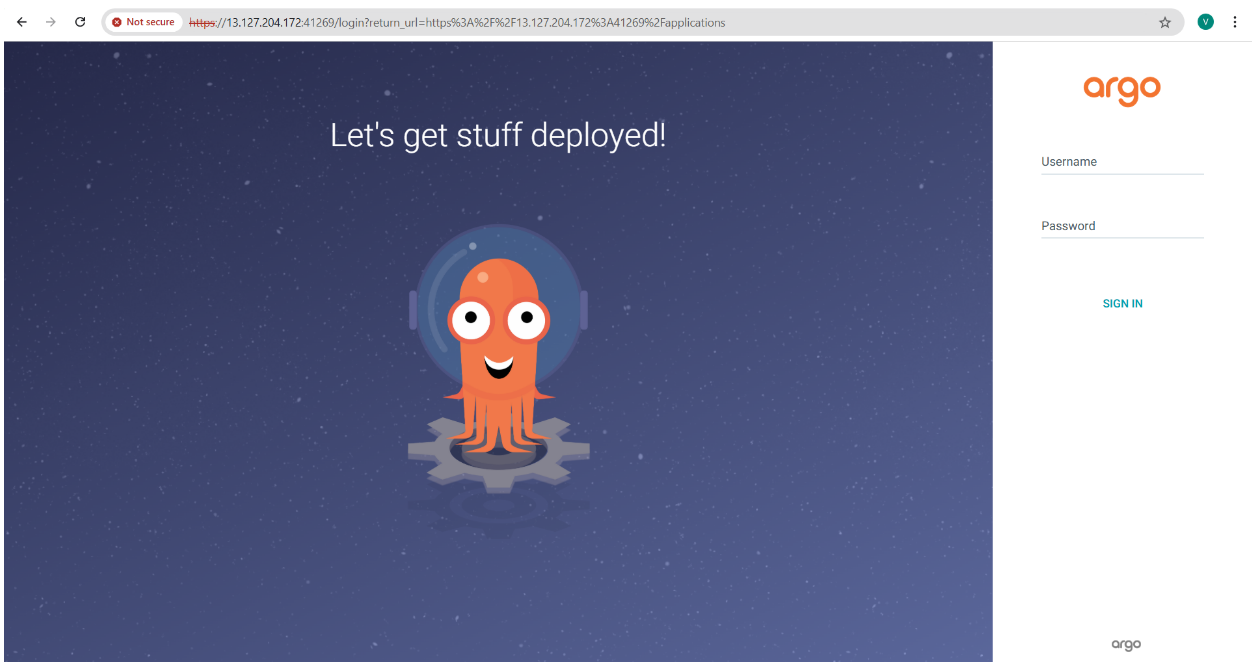
- This is specially used when you're running multiple port forwarding commands or doesn't know which port is available on your machine.
- Again modify the command slightly. The `:` sign before `80` tells kubernetes to dynamically assign a free port on your local machine and map it to port `80` on the service.

```
kubectl port-forward svc/argocd-server :80 --address=0.0.0.0 -n
```



```
ubuntu@ip-172-31-36-84:~$ kubectl port-forward svc/argocd-server :80 --address=0.0.0.0 -n argocd
Forwarding from 0.0.0.0:41269 -> 8080
```

The port assigned is `41269`, open this port in your security group and try to access your application in the browser.


Not secure
https://13.127.204.172:41269/login?return_url=https%3A%2F%2F13.127.204.172%3A41269%2Fapplications
Let's get stuff deployed!
argo
Username
Password
SIGN IN
argo

✅ The port forwarding is successful in all the three use cases.

Conclusion: By understanding and applying these techniques you can securely test and debug your application running in kubernetes clusters whether they are hosted locally or on a cloud instance. We have seen binding ports to specific addresses and also dynamically assigning for our convenience.

Hope you liked this article!

Happy Port-Forwarding! 🔥

If you found this post useful, give it a like👍

Repost♻️

Follow @Bala Vignesh for more such posts 💯🚀