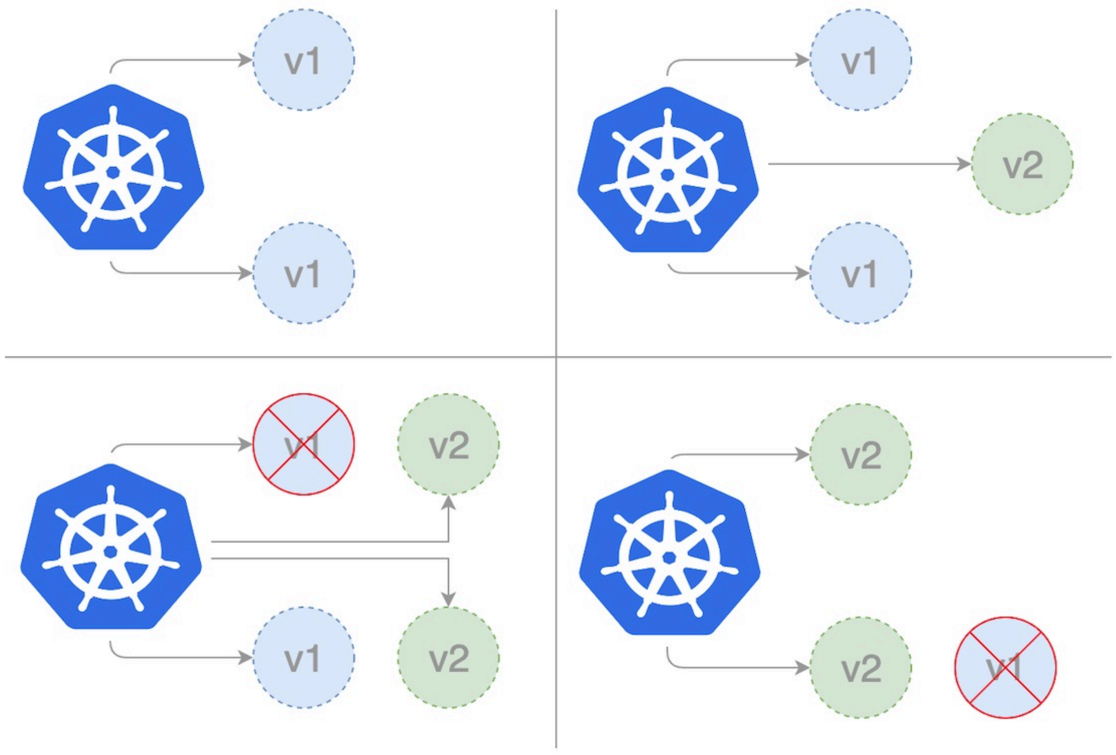
Kubernetes Rolling Updates for Zero-Downtime Deployments



Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation results from counting the eight letters between the "K" and the "s". Google open-sourced the Kubernetes project in 2014.

In this article, we will explain to you what Rolling updates in Kubernetes are, and how to achieve Zero-Downtime Deployments by taking a small Example.

What is a Rolling Update?

- A rolling update allows a Deployment update to take place with zero downtime. It does this by incrementally replacing the current Pods with new ones.
- Rolling updates also allow for a rollback to a previous or older version if there is an issue with the new version.

Benefits of rolling updates over other deployment strategies

- **Minimised Downtime:** Rolling updates allow you to update your application or system without taking it offline completely. By gradually updating instances one by one, you can maintain a high level of availability for users.
- **Risk Mitigation:** Since rolling updates are incremental, they help mitigate the risk of deploying a faulty version. If an issue is detected, you can stop the update process and rollback to the previous stable version.
- **Graceful Degradation:** Rolling updates ensure a smooth transition by gradually replacing old instances with new ones. This helps in maintaining system performance and avoiding sudden spikes or drops in traffic.
- **Easy Monitoring and Troubleshooting:** With rolling updates, you can monitor the update progress in real-time and identify any issues early on. This makes troubleshooting and debugging easier compared to bulk updates or blue-green deployments.
- **Resource Efficiency:** Rolling updates consume fewer resources compared to other strategies like blue-green deployments, where you need to maintain separate environments. This efficiency is crucial, especially in resource-constrained environments or cloud deployments with cost considerations.
- **Version Flexibility:** Rolling updates allow you to roll back to a previous version quickly if needed. This flexibility is valuable in situations where a new version introduces unforeseen issues or compatibility problems.
- **Easy to Use:** Rolling update can be achieved by simply updating the image tag in the Deployment manifest. Kubernetes will automatically perform rolling updates.

Implementing Rolling Update Strategy in Kubernetes:

nginx-deplyoment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      name: nginx
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        name: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
```

- Here, we are introducing a new section named 'strategy' that specifies the type of strategy we intend to employ.
- In strategy section we have
  - *Type: RollingUpdate* specifies that the deployment should use a rolling update strategy.
  - *maxUnavailable: 1* indicates that during the update, no more than 1 instance can be unavailable at a time.
  - *maxSurge: 1* specifies that during the update, at most 1 additional instance can be created beyond the desired number of instances.

- This nginx-deployment.yaml file creates nginx pods with 3 replica sets
- To expose this Deployment to the external world, we'll utilise a Service of type NodePort or LoadBalancer in Kubernetes.

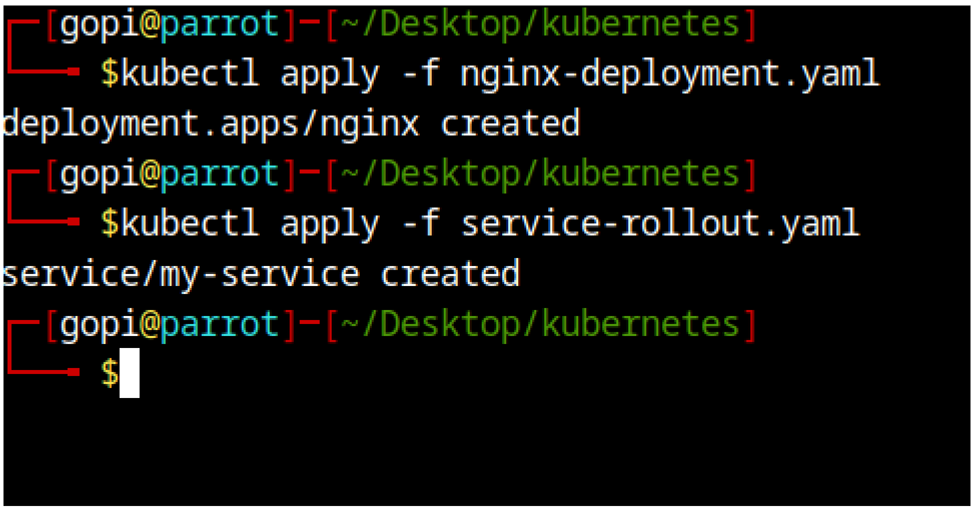
- Service-rollout.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    name: nginx
  ports:
    - port: 80
      nodePort: 30007
```

- We have created a Service of type NodePort, mapping port 30007 to port 80, to expose and make this Deployment accessible externally.

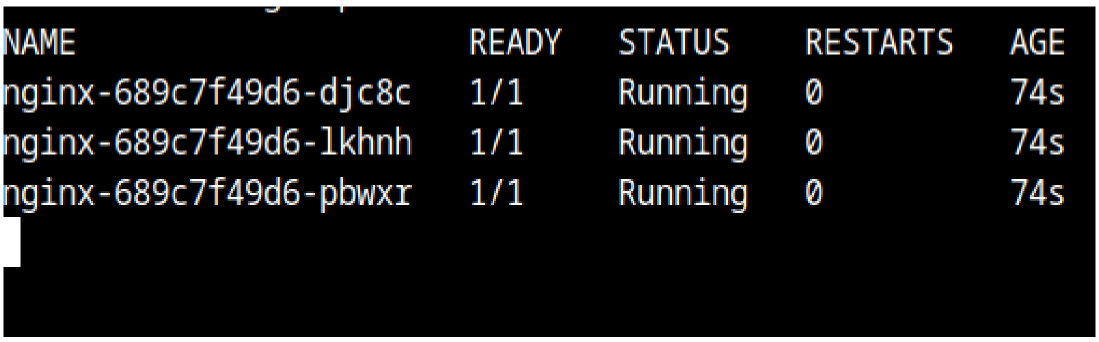
Deploying resources into K8'S cluster:

- Make sure to start any kubernetes cluster (I Am using minikube)
  - Minikube start
- Kubectl apply -f nginx-deployment.yaml
- Kubectl apply -f service-rollout.yaml



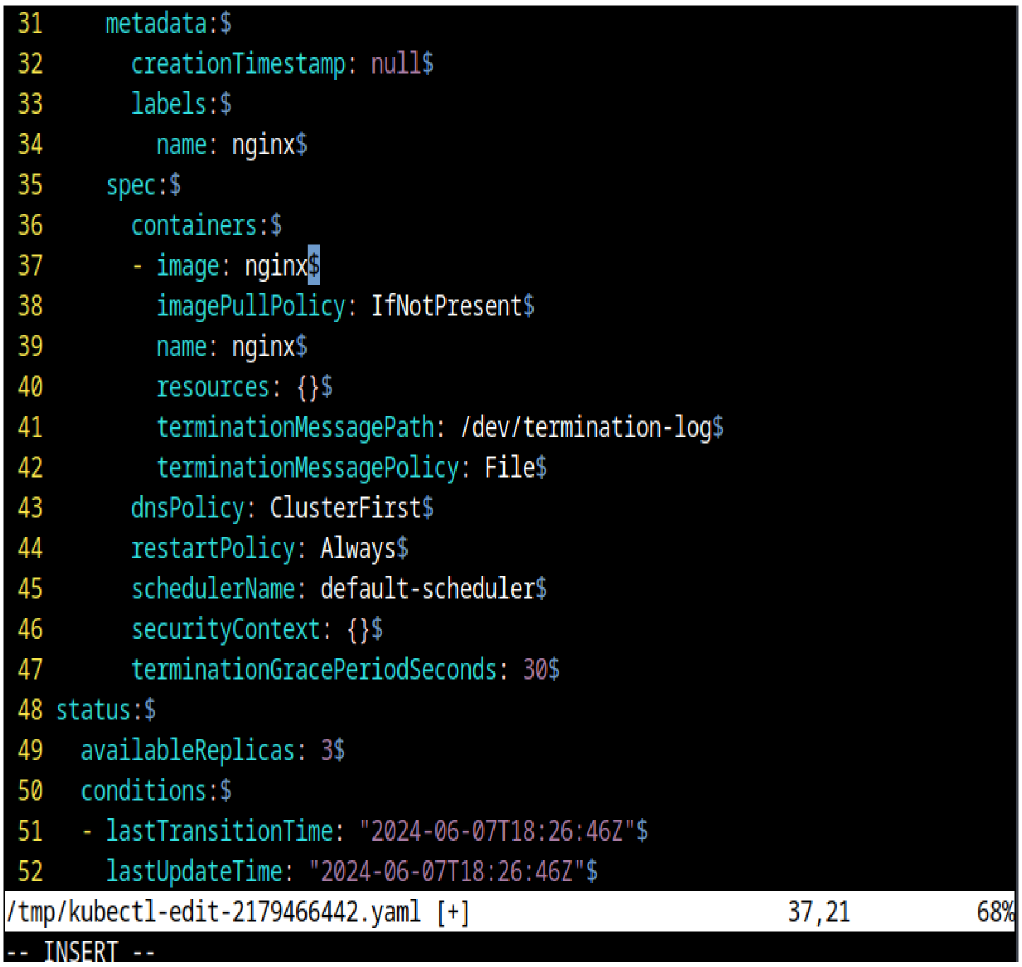
```
┌─[gopi@parrot]─[~/Desktop/kubernetes]
└──╼ $kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx created
┌─[gopi@parrot]─[~/Desktop/kubernetes]
└──╼ $kubectl apply -f service-rollout.yaml
service/my-service created
┌─[gopi@parrot]─[~/Desktop/kubernetes]
└──╼ $
```

- Inspect pods via "kubectl get pods -w"



```
NAME                     READY   STATUS    RESTARTS   AGE
nginx-689c7f49d6-djc8c   1/1     Running   0          74s
nginx-689c7f49d6-lkhnh   1/1     Running   0          74s
nginx-689c7f49d6-pbwxr   1/1     Running   0          74s
```

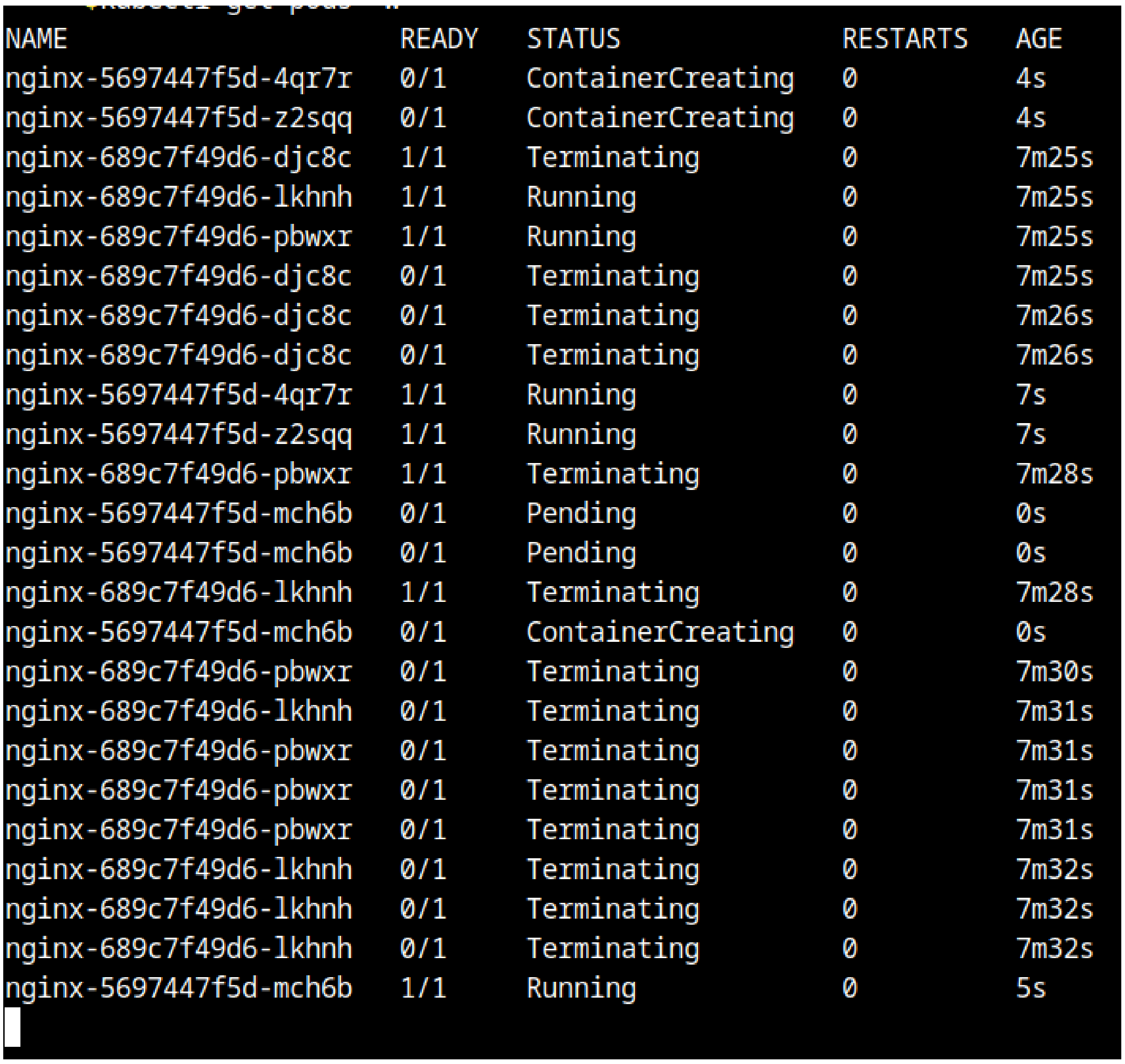
- To verify Rollout update, i will be changing version of nginx from 1.14.2 to latest
- Run kubectl edit deploy nginx-deployment and edit the nginx image version



```
    metadata:$
      creationTimestamp: null$
      labels:$
        name: nginx$
    spec:$
      containers:$
      - image: nginx$
        imagePullPolicy: IfNotPresent$
        name: nginx$
        resources: {}$
        terminationMessagePath: /dev/termination-log$
        terminationMessagePolicy: File$
      dnsPolicy: ClusterFirst$
      restartPolicy: Always$
      schedulerName: default-scheduler$
      securityContext: {}$
      terminationGracePeriodSeconds: 30$
status:$
  availableReplicas: 3$
  conditions:$
  - lastTransitionTime: "2024-06-07T18:26:46Z"$
    lastUpdateTime: "2024-06-07T18:26:46Z"$
/tmp/kubectl-edit-2179466442.yaml [+]                 37,21          68%
-- INSERT --
```

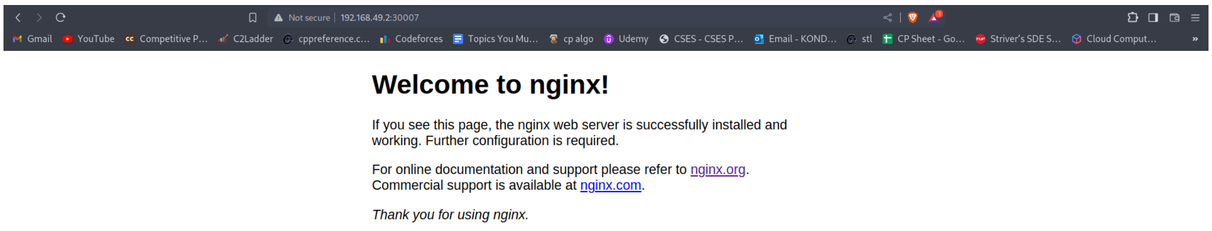
- Now press :wq to save changes

- Use kubectl get pods -w to watch pods, here you can see that new pods are created before old ones got removed



```
     $kubectl get pods -w
NAME                      READY   STATUS              RESTARTS   AGE
nginx-5697447f5d-4qr7r    0/1     ContainerCreating   0          4s
nginx-5697447f5d-z2sqq    0/1     ContainerCreating   0          4s
nginx-689c7f49d6-djc8c    1/1     Terminating         0          7m25s
nginx-689c7f49d6-lkhnh    1/1     Running             0          7m25s
nginx-689c7f49d6-pbwxr    1/1     Running             0          7m25s
nginx-689c7f49d6-djc8c    0/1     Terminating         0          7m25s
nginx-689c7f49d6-djc8c    0/1     Terminating         0          7m26s
nginx-689c7f49d6-djc8c    0/1     Terminating         0          7m26s
nginx-5697447f5d-4qr7r    1/1     Running             0          7s
nginx-5697447f5d-z2sqq    1/1     Running             0          7s
nginx-689c7f49d6-pbwxr    1/1     Terminating         0          7m28s
nginx-5697447f5d-mch6b    0/1     Pending             0          0s
nginx-5697447f5d-mch6b    0/1     Pending             0          0s
nginx-689c7f49d6-lkhnh    1/1     Terminating         0          7m28s
nginx-5697447f5d-mch6b    0/1     ContainerCreating   0          0s
nginx-689c7f49d6-pbwxr    0/1     Terminating         0          7m30s
nginx-689c7f49d6-lkhnh    0/1     Terminating         0          7m31s
nginx-689c7f49d6-pbwxr    0/1     Terminating         0          7m31s
nginx-689c7f49d6-pbwxr    0/1     Terminating         0          7m31s
nginx-689c7f49d6-pbwxr    0/1     Terminating         0          7m31s
nginx-689c7f49d6-lkhnh    0/1     Terminating         0          7m32s
nginx-689c7f49d6-lkhnh    0/1     Terminating         0          7m32s
nginx-689c7f49d6-lkhnh    0/1     Terminating         0          7m32s
nginx-5697447f5d-mch6b    1/1     Running             0          5s
```

- In the above image, older pods were deleted only after the new ones entered the running state.



Not secure | 192.168.49.2:30007

Gmail YouTube Competitive P... C2Ladder cppreference.c... Codeforces Topics You Mu... cp algo Udemy CSES - CSES P... Email - KOND... stl CP Sheet - Go... Striver's SDE S... Cloud Comput...

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

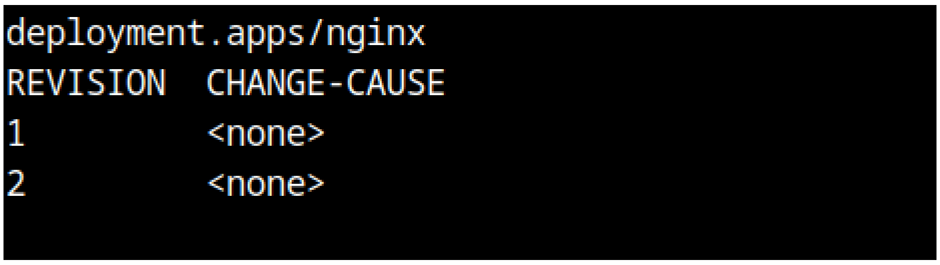
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

- Here we can see that the Nginx welcome page is working properly

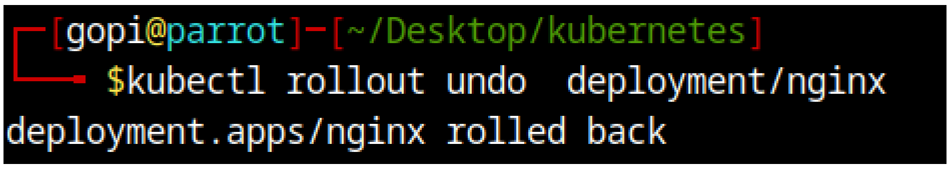
Rolling Back to Older Version

- kubectl rollout history deployment/nginx command shows all the newly updated versions



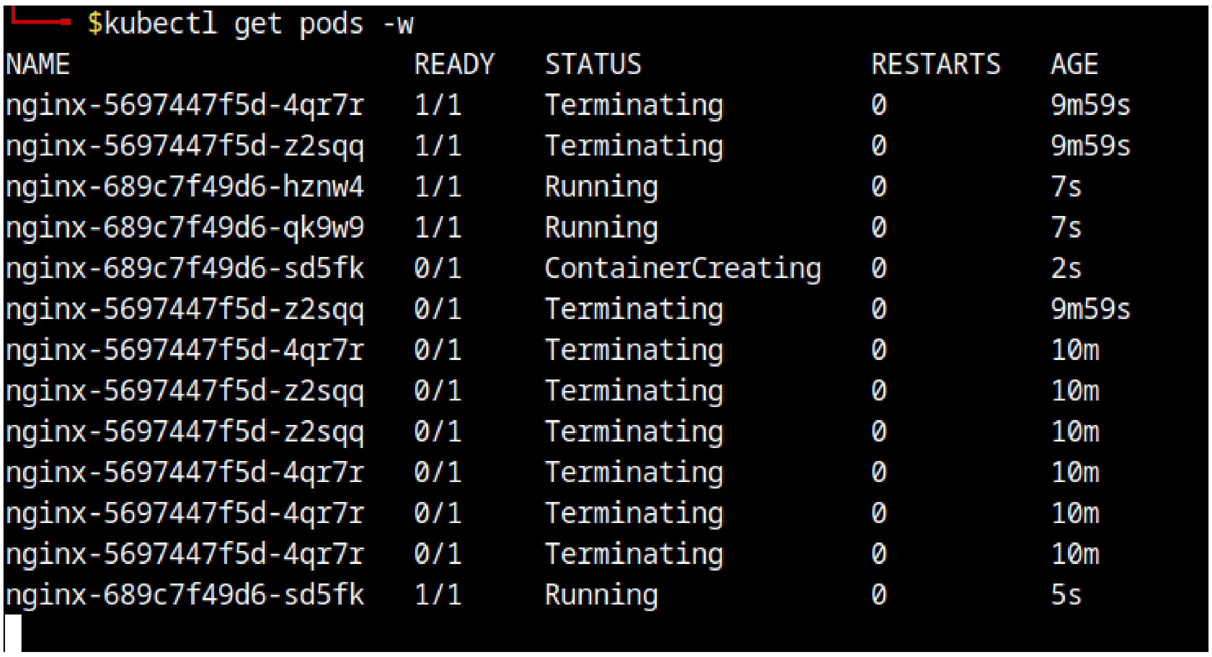
```
deployment.apps/nginx
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

- In Order to Rollback to older version (1 step) use “kubectl rollout undo deployment/nginx” command.



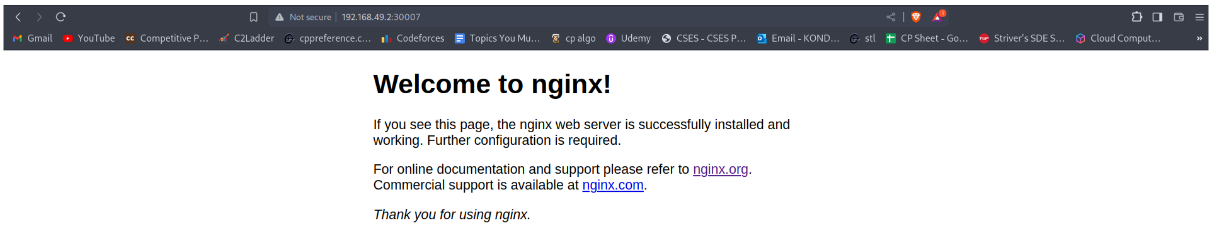
```
┌─[gopi@parrot]─[~/Desktop/kubernetes]
└──╼ $kubectl rollout undo  deployment/nginx
deployment.apps/nginx rolled back
```

- We can inspect newly generated pods using “kubectl get pods -w”, here the “-w” argument continuously monitors all the pods.



```
└──╼ $kubectl get pods -w
NAME                        READY   STATUS              RESTARTS   AGE
nginx-5697447f5d-4qr7r      1/1     Terminating         0          9m59s
nginx-5697447f5d-z2sqq      1/1     Terminating         0          9m59s
nginx-689c7f49d6-hznw4      1/1     Running             0          7s
nginx-689c7f49d6-qk9w9      1/1     Running             0          7s
nginx-689c7f49d6-sd5fk      0/1     ContainerCreating   0          2s
nginx-5697447f5d-z2sqq      0/1     Terminating         0          9m59s
nginx-5697447f5d-4qr7r      0/1     Terminating         0          10m
nginx-5697447f5d-z2sqq      0/1     Terminating         0          10m
nginx-5697447f5d-z2sqq      0/1     Terminating         0          10m
nginx-5697447f5d-4qr7r      0/1     Terminating         0          10m
nginx-5697447f5d-4qr7r      0/1     Terminating         0          10m
nginx-5697447f5d-4qr7r      0/1     Terminating         0          10m
nginx-689c7f49d6-sd5fk      1/1     Running             0          5s
```

- Even after rolling back to older version there was no downtime observed whatsoever



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Some Observed issues while Rolling

- **Downtime**: If your application does not have sufficient redundancy or if the update process involves stopping and restarting pods, there may be a brief period of downtime. This downtime can impact user experience if users are unable to access the application during the update.
- **Degraded Performance**: During a rollout update, some pods may be unavailable or under heavy load as new versions are being deployed and old versions are terminated. This can lead to degraded performance, slower response times, or increased latency for users.
- **Temporary Errors**: If the new version introduces bugs or compatibility issues, users may experience temporary errors or unexpected behavior.

It's crucial to thoroughly test new versions before rolling them out to minimise the risk of such issues.

- **Increased Resource Usage**: The rollout process may temporarily increase resource usage, such as CPU, memory, and network bandwidth, as Kubernetes creates new pods and manages the transition between old and new versions. This can affect overall application performance and scalability.
- **Session Management**: Depending on your application's session management strategy, users may experience disruptions or loss of session state during the update process. It's essential to handle session persistence or stateful data appropriately to minimise user impact.

<u>Potential Risks</u>

- **Traffic Issues**: Changes during updates can cause traffic spikes or drops.
  Solution: Use readiness probes and adjust deployment settings for a smooth transition. Consider auto-scaling to handle traffic changes automatically.
- **Resource Strain**: Updates may strain cluster resources like CPU and memory.
  Solution: Monitor resource usage and adjust settings. Use auto-scaling to scale resources based on demand.
- **Service Interruptions**: Updates can lead to service interruptions or downtime.
  Solution: Test updates with a small group first (canary deployment) and automate testing. Have rollback plans ready.
- **Data Safety**: Updates might affect data integrity, especially for databases.
  Solution: Use persistent volumes and backups. Plan for data recovery if needed.
- **Rollback Complexity**: Rolling back can be complex if an update fails.
  Solution: Plan for rollback scenarios and automate rollback procedures. Maintain version control.
- **Dependency Management**: Updates may involve changes to dependent services.
  Solution: Use Helm charts or package managers for managing dependencies. Implement versioning and dependency practices in CI/CD pipelines.

<u>POC Key Findings:</u>

- **Feasibility**: The POC demonstrates the feasibility of using Kubernetes for application deployment and management.
- **Scalability**: Kubernetes shows scalability capabilities, allowing easy scaling of resources based on demand.
- **Fault Tolerance**: Kubernetes exhibits fault tolerance, with features like pod health checks and automatic pod restarts.
- **Resource Efficiency**: Kubernetes efficiently manages resources, optimising resource utilisation across the cluster.
- **Automation**: Kubernetes automation features streamline deployment, scaling, and management tasks, reducing manual effort.
- **Monitoring**: Kubernetes provides robust monitoring and logging capabilities, enabling effective performance tracking and troubleshooting.

Benefits

- **Zero-downtime deployments:** Updates happen without disrupting service availability.
- **Lower risk:** New versions are rolled out gradually, reducing the chance of failures.
- **Efficient resource use:** Kubernetes replaces old instances smoothly to optimise resources.
- **Smooth traffic management:** Users experience minimal interruptions during updates.
- **Quick recovery:** If there's an issue with the update, reverting to the previous version is easy.
- **Scalability:** Applications can scale up or down seamlessly based on demand.
- **Better user experience:** Continuous availability means users enjoy reliable services.

Summary:

When you need to roll back a deployment in Kubernetes, it typically means there's an issue with the latest version of your application or configuration that you need to address. Rolling back essentially means reverting to a previous known-good state to restore functionality and stability.

After the rollback is initiated, Kubernetes will start the process of reverting the deployment to the specified revision. It will create new pods with the older version of your application while gradually scaling down the pods running the problematic version. This process ensures that your application stays available throughout the rollback, avoiding downtime for your users.

In summary, rolling back a deployment in Kubernetes involves identifying the problem, checking revision history, initiating the rollback, verifying the rollback's success, and monitoring the application post-rollback to ensure stability and functionality.