Date: 04.07.2025



Copyright @ 2024 PibyThree.com All Rights Reserved

K8s Service Management and Debugging Essentials

Contents

[1. Service Types 2](#_Toc202521891)

[a. ClusterIP 2](#_Toc202521892)

[b. NodePort 2](#_Toc202521893)

[c. LoadBalancer 2](#_Toc202521894)

[d. ExternalName 2](#_Toc202521895)

[2. Namespaces 3](#_Toc202521896)

[a. What are namespaces? 3](#_Toc202521897)

[b. Initial Namespaces 3](#_Toc202521898)

[c. Manifest file to create a namespace 3](#_Toc202521899)

[3. Debugging running pods 4](#_Toc202521900)

[4. Resource Cleanup 4](#_Toc202521901)

# 1. Service Types

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: <SERVICE\_NAME>  spec:  selector:  app: <DEPLOYMENT.METADATA.LABELS.APP>  type: <SERVICE\_TYPE>  ports:  - protocol: <PROTOCOL>  port: <PORT\_WHERE\_SERVICE\_EXPOSED\_IN\_CLUSTER>  targetPort: <PORT\_ON\_POD\_THAT\_RECIEVES\_TRAFFIC> |

## ClusterIP

* Exposes the Service on a cluster-internal IP.
* This value makes the Service only reachable from within the cluster.
* This is the default that is used if you don't explicitly specify a type for a Service.
* You can expose the Service to the public internet using an Ingress or a Gateway.
* Assigns an IP address from a pool of IP addresses that your cluster has reserved for that purpose.

## NodePort

* Exposes the Service on each Node's IP at a static port (the NodePort).
* To make the node port available, Kubernetes sets up a cluster IP address, the same as if you had requested a Service of type: ClusterIP.
* The Kubernetes control plane allocates a port from a range specified by --service-node-port-range flag (default: 30000-32767).

## LoadBalancer

* Exposes the Service externally using an external load balancer.
* Kubernetes does not directly offer a load balancing component; you must provide one, or you can integrate your Kubernetes cluster with a cloud provider.
* On cloud providers which support external load balancers, setting the `type` field to LoadBalancer provisions a load balancer for your Service.

## ExternalName

* Maps the Service to the contents of the externalName field (for example, to the hostname api.foo.bar.example).
* The mapping configures your cluster's DNS server to return a CNAME record with that external hostname value. No proxying of any kind is set up.
* Accepts an IPv4 address string, but treats that string as a DNS name comprised of digits.

# 2. Namespaces

## What are namespaces?

* *namespaces* provide a mechanism for isolating groups of resources within a single cluster.
* Names of resources need to be unique within a namespace, but not across namespaces.
* Namespace-based scoping is applicable only for namespaced objects *(e.g. Deployments, Services, etc.)* and not for cluster-wide objects *(e.g. StorageClass, Nodes, PersistentVolumes, etc.)*.
* It is intended for use in environments with many users spread across multiple teams, or projects.
* It is not necessary to use multiple namespaces to separate slightly different resources, such as different versions of the same software: use `labels` to distinguish resources within the same namespace.
* It is a way to divide cluster resources between multiple users (via resource quota).

## Initial Namespaces

|  |  |
| --- | --- |
| **Namespace** | **Summary** |
| default | Used for resources when no other namespace is specified. |
| kube-node-lease | Stores node heartbeat data to help detect node failures. |
| kube-public | Publicly readable namespace, typically for cluster-wide shared resources. |
| kube-system | Contains system components and resources created by Kubernetes itself. |

## Manifest file to create a namespace

|  |
| --- |
| apiVersion: v1  kind: Namespace  metadata:  name: <NAME>  labels:  name: <NAME> |

# 3. Debugging running pods

`kubectl get pods` 🡪 Get status of all pods

`kubectl describe pod <PODNAME>` 🡪 More information like configuration & status info

Analyze ‘Event’ section for any debug information

`kubectl get events` 🡪 Lists all the events

`kubectl get pod <PODNAME> -o yaml` 🡪 Returns in YAML

`kubectl logs <PODNAME> <CONTAINERNAME>` 🡪 View logs of affected container

`kubectl logs --previous <PODNAME> -c <CONTAINERNAME>` 🡪 Previous container’s logs

`kubectl exec <PODNAME> -c <CONTAINERNAME> -- <CMD>` 🡪 Run command inside a

specific container

`kubectl exec -it <PODNAME> -- /bin/bash` 🡪 Open an interactive terminal connecting to the pod

`kubectl exec --stdin --tty <PODNAME> -- /bin/bash` 🡪 Same as previous command

`kubectl debug node/<NODENAME> -it --image=ubuntu` 🡪 Interactive shell to the node

Find the node that is running the pod first

# 4. Resource Cleanup

`kubectl delete <RESOURCE> <NAME>` 🡪 Delete specific resource

`kubectl get pods --field-selector=status.phase==<Succeeded/Failed>` 🡪 Get completed/failed pods

`kubectl delete pod <PODNAME>` 🡪 Use the result of above command and then delete pod

`kubectl apply --prune -l app=myapp -f ./manifests --all` 🡪 Delete any previous resources in the cluster with label [app=myapp] but not defined in the current set of manifests being applied

`kubectl delete ns <ns-name>` 🡪 Delete entire namespace and all its resources

```  
spec:

ttlSecondsAfterFinished: <TIME\_IN\_SECs>  
``` 🡪 In Jobs and CronJobs, delete that resource after it is finished