



Deploying Your First Nginx Pod

Kubernetes - Beginners | Intermediate | Advanced

[View on GitHub](#) [Join Slack](#)

Deploying Your First Nginx Pod

What are K8s Pods?

- Kubernetes pods are the foundational unit for all higher Kubernetes objects.
- A pod hosts one or more containers.
- It can be created using either a command or a YAML/JSON file.
- Use `kubectl` to create pods, view the running ones, modify their configuration, or terminate them. Kubernetes will attempt to restart a failing pod by default.
- If the pod fails to start indefinitely, we can use the `kubectl describe` command to know what went wrong.

Why does Kubernetes use a Pod as the smallest deployable unit, and not a single container?

While it would seem simpler to just deploy a single container directly, there are good reasons to add a layer of abstraction represented by the Pod. A container is an existing

entity, which refers to a specific thing. That specific thing might be a Docker container, but it might also be a [rkt](#) container, or a VM managed by Virtlet. Each of these has different requirements.

What's more, to manage a container, Kubernetes needs additional information, such as a restart policy, which defines what to do with a container when it terminates, or a liveness probe, which defines an action to detect if a process in a container is still alive from the application's perspective, such as a web server responding to HTTP requests.

Instead of overloading the existing "thing" with additional properties, Kubernetes architects have decided to use a new entity, the Pod, that logically contains (wraps) one or more containers that should be managed as a single entity.

Why does Kubernetes allow more than one container in a Pod?

Containers in a Pod run on a "logical host"; they use the same network namespace (in other words, the same IP address and port space), and the same [IPC](#) namespace. They can also use shared volumes. These properties make it possible for these containers to efficiently communicate, ensuring data locality. Also, Pods enable you to manage several tightly coupled application containers as a single unit.

So if an application needs several containers running on the same host, why not just make a single container with everything you need? Well first, you're likely to violate the "one process per container" principle. This is important because with multiple processes in the same container it is harder to troubleshoot the container. That is because logs from different processes will be mixed together and it is harder manage the processes lifecycle. For example to take care of "zombie" processes when their parent process dies. Second, using several containers for an application is simpler, more transparent, and enables decoupling software dependencies. Also, more granular containers can be reused between teams.

Pre-requisite:

Steps

```
git clone https://github.com/collabnix/kubelabs
cd kubelabs/pods101
kubectl apply -f pods01.yaml
```

Viewing Your Pods

```
kubectl get pods
```

Which Node Is This Pod Running On?

```
kubectl get pods -o wide
```

```
$ kubectl describe po webserver
Name:                webserver
Namespace:           default
Priority:             0
PriorityClassName:    <none>
Node:                gke-standard-cluster-1-default-pool-78257330-5hs8/10.128
Start Time:          Thu, 28 Nov 2019 13:02:19 +0530
Labels:              <none>
Annotations:         kubectl.kubernetes.io/last-applied-configuration:
                      {"apiVersion":"v1","kind":"Pod","metadata":{"annotatio
                      kubernetes.io/limit-ranger: LimitRanger plugin set: cpu
Status:              Running
IP:                  10.8.0.3
Containers:
  webserver:
    Container ID:     docker://ff06c3e6877724ec706485374936ac6163aff10822246a4
    Image:            nginx:latest
    Image ID:         docker-pullable://nginx@sha256:189cce606b29fb2a33ebc2fce
    Port:             80/TCP
    Host Port:        0/TCP
    State:            Running
      Started:        Thu, 28 Nov 2019 13:02:25 +0530
    Ready:            True
```

```

Restart Count: 0
Requests:
  cpu:          100m
Environment: <none>
Mounts:
  /var/run/secrets/kubernetes.io/serviceaccount from default-token-mpxxg
Conditions:
  Type              Status
  Initialized        True
  Ready              True
  ContainersReady    True
  PodScheduled       True
Volumes:
  default-token-mpxxg:
    Type:          Secret (a volume populated by a Secret)
    SecretName:     default-token-mpxxg
    Optional:       false
QoS Class:         Burstable
Node-Selectors:     <none>
Tolerations:        node.kubernetes.io/not-ready:NoExecute for 300s
                    node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason      Age    From
  ----    -
  Normal  Scheduled   2m54s  default-scheduler
  Normal  Pulling     2m53s  kubelet, gke-standard-cluster-1-default-pool-782
  Normal  Pulled      2m50s  kubelet, gke-standard-cluster-1-default-pool-782
  Normal  Created     2m48s  kubelet, gke-standard-cluster-1-default-pool-782
  Normal  Started     2m48s  kubelet, gke-standard-cluster-1-default-pool-782

```

Output in JSON

```

$ kubectl get pods -o json
{
  "apiVersion": "v1",
  "items": [
    {
      "apiVersion": "v1",
      "kind": "Pod",

```

```
"metadata": {
  "annotations": {
    "kubectl.kubernetes.io/last-applied-configuration": "{\"\ntainers\": [{\"image\": \"nginx:latest\", \"name\": \"webserver\", \"ports\": [{\"\nkubernetes.io/limit-ranger\": \"LimitRanger plugin set: c\n  },\n  \"creationTimestamp\": \"2019-11-28T08:48:28Z\",\n  \"name\": \"webserver\",\n  \"namespace\": \"default\",\n  \"resourceVersion\": \"20080\",\n  \"selfLink\": \"/api/v1/namespaces/default/pods/webserver\",\n  \"uid\": \"d8e0b56b-11bb-11ea-a1bf-42010a800006\"\n},\n  \"spec\": {\n    \"containers\": [\n      {\n        \"image\": \"nginx:latest\",\n        \"imagePullPolicy\": \"Always\",\n        \"name\": \"webserver\",\n        \"ports\": [\n          {\n            \"containerPort\": 80,\n            \"protocol\": \"TCP\"\n          }\n        ],\n        \"resources\": {\n          \"requests\": {\n            \"cpu\": \"100m\"\n          }\n        },\n        \"terminationMessagePath\": \"/dev/termination-log\",\n        \"terminationMessagePolicy\": \"File\",
```

Executing Commands Against Pods

```
$ kubectl exec -it webserver -- /bin/bash
root@webserver:/#
```

```
root@webserver:/# cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 10 (buster)"
NAME="Debian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

Please exit from the shell (/bin/bash) session.

```
root@webserver:/# exit
```

Deleting the Pod

```
$ kubectl delete -f pods01.yaml
pod "webserver" deleted

$ kubectl get po -o wide
No resources found.
```

Adding a 2nd container to a Pod

In the microservices architecture, each module should live in its own space and communicate with other modules following a set of rules. But, sometimes we need to deviate a little from this principle. Suppose you have an Nginx web server running and we need to analyze its web logs in real-time. The logs we need to parse are obtained from GET requests to the web server. The developers created a log watcher application that will do this job and they built a container for it. In typical conditions, you'd have a pod for Nginx and another for the log watcher. However, we need to eliminate any network latency so that the watcher can analyze logs the moment they are available. A solution for this is to place both containers on the same pod.

Having both containers on the same pod allows them to communicate through the loopback interface (`ifconfig lo`) as if they were two processes running on the same host. They also share the same storage volume.

Let us see how a pod can host more than one container. Let's take a look to the `pods02.yaml` file. It contains the following lines:

```
apiVersion: v1
kind: Pod
metadata:
  name: webserver
spec:
  containers:
  - name: webserver
    image: nginx:latest
    ports:
    - containerPort: 80
  - name: webwatcher
    image: afakharany/watcher:latest
```

Run the following command:

```
$ kubectl apply -f pods02.yaml
```

```
$ kubectl get po -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
webserver	0/2	ContainerCreating	0	13s	<none>	gke-standa

```
$ kubectl get po,svc,deploy
```

NAME	READY	STATUS	RESTARTS	AGE
pod/webserver	2/2	Running	0	3m6s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.12.0.1	<none>	443/TCP	107m

```
$ kubectl get po -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
------	-------	--------	----------	-----	----	------

```
webserver    2/2    Running    0          3m37s    10.8.0.5    gke-standard-clu
```

How to verify 2 containers are running inside a Pod?

```
$ kubectl describe po
```

Containers:

webserver:

```
Container ID:    docker://0564fcb88f7c329610e7da24cba9de6555c0183814cf517
Image:           nginx:latest
Image ID:        docker-pullable://nginx@sha256:36b77d8bb27ffca25c7f6f53c
Port:            80/TCP
State:           Running
  Started:       Wed, 08 Jan 2020 13:21:57 +0530
Ready:           True
Restart Count:   0
Requests:
  cpu:           100m
Environment:     <none>
Mounts:
  /var/run/secrets/kubernetes.io/serviceaccount from default-token-xhgmm
```

webwatcher:

```
Container ID:    docker://4cebbb220f7f9695f4d6492509e58152ba661f3ab8f4b5d
Image:           afakharany/watcher:latest
Image ID:        docker-pullable://afakharany/watcher@sha256:43d1b12bb4ce
Port:            <none>
State:           Running
  Started:       Wed, 08 Jan 2020 13:22:26 +0530
Ready:           True
Restart Count:   0
Requests:
```

Since we have two containers in a pod, we will need to use the `-c` option with `kubectl` when we need to address a specific container. For example:

```
$ kubectl exec -it webserver -c webwatcher -- /bin/bash
```



```
root@webserver:/# cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1        localhost
::1             localhost ip6-localhost ip6-loopback
fe00::0          ip6-localnet
fe00::0          ip6-mcastprefix
fe00::1          ip6-allnodes
fe00::2          ip6-allrouters
10.8.0.5         webserver
```

Please exit from the shell (/bin/bash) session.

```
root@webserver:/# exit
```

Cleaning up

```
kubectl delete -f pods02.yaml
```

Example of Multi-Container Pod

Let's talk about communication between containers in a Pod. Having multiple containers in a single Pod makes it relatively straightforward for them to communicate with each other. They can do this using several different methods.

Use Cases for Multi-Container Pods

The primary purpose of a multi-container Pod is to support co-located, co-managed helper processes for a primary application. There are some general patterns for using helper processes in Pods:

Sidecar containers help the main container. Some examples include log or data change watchers, monitoring adapters, and so on. A log watcher, for example, can be built once by a different team and reused across different applications. Another example of a

sidecar container is a file or data loader that generates data for the main container.

Proxies, bridges, and adapters connect the main container with the external world. For example, Apache HTTP server or nginx can serve static files. It can also act as a reverse proxy to a web application in the main container to log and limit HTTP requests. Another example is a helper container that re-routes requests from the main container to the external world. This makes it possible for the main container to connect to the localhost to access, for example, an external database, but without any service discovery.

Shared volumes in a Kubernetes Pod

In Kubernetes, you can use a shared Kubernetes Volume as a simple and efficient way to share data between containers in a Pod. For most cases, it is sufficient to use a directory on the host that is shared with all containers within a Pod.

Kubernetes Volumes enables data to survive container restarts, but these volumes have the same lifetime as the Pod. That means that the volume (and the data it holds) exists exactly as long as that Pod exists. If that Pod is deleted for any reason, even if an identical replacement is created, the shared Volume is also destroyed and created anew.

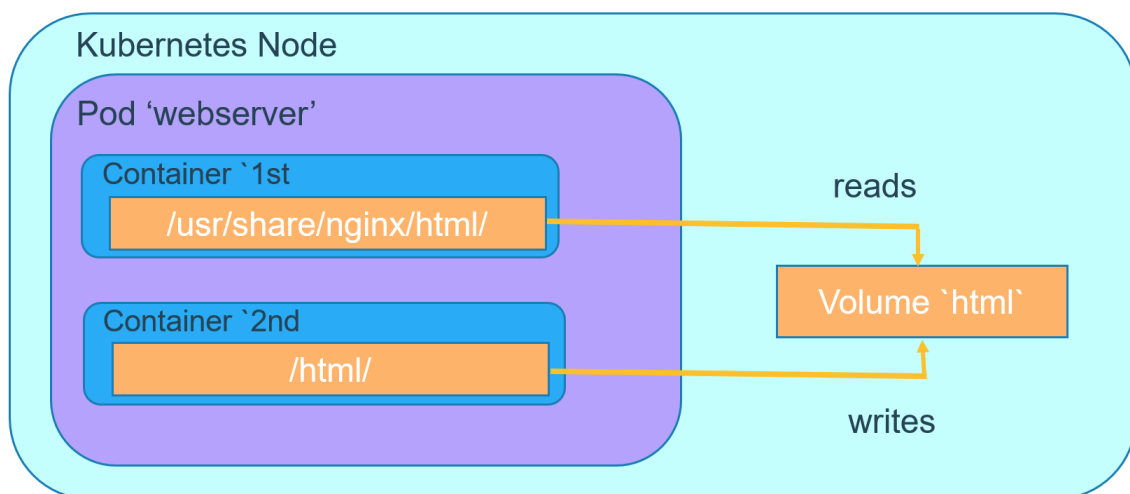
A standard use case for a multi-container Pod with a shared Volume is when one container writes logs or other files to the shared directory, and the other container reads from the shared directory. For example, we can create a Pod like so ([pods03.yaml](#)):

```
apiVersion: v1
kind: Pod
metadata:
  name: mc1
spec:
  volumes:
  - name: html
    emptyDir: {}
  containers:
  - name: 1st
    image: nginx
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
```

```
- name: 2nd
  image: debian
  volumeMounts:
  - name: html
    mountPath: /html
  command: ["/bin/sh", "-c"]
  args:
  - while true; do
    date >> /html/index.html;
    sleep 1;
  done
```

In this file (pods03.yaml) a volume named `html` has been defined. Its type is `emptyDir` , which means that the volume is first created when a Pod is assigned to a node, and exists as long as that Pod is running on that node. As the name says, it is initially empty. The 1st container runs nginx server and has the shared volume mounted to the directory `/usr/share/nginx/html`. The 2nd container uses the Debian image and has the shared volume mounted to the directory `/html` . Every second, the 2nd container adds the current date and time into the `index.html` file, which is located in the shared volume. When the user makes an HTTP request to the Pod, the Nginx server reads this file and transfers it back to the user in response to the request.

Multi-container Pods



48



```
kubectl apply -f pods03.yaml
```

```
[Captains-Bay] ➤ > kubectl get po,svc
```

NAME	READY	STATUS	RESTARTS	AGE
po/mc1	2/2	Running	0	11s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/kubernetes	ClusterIP	10.15.240.1	<none>	443/TCP	1h

```
[Captains-Bay] ➤ > kubectl describe po mc1
```

```
Name:          mc1
Namespace:     default
Node:          gke-k8s-lab1-default-pool-fd9ef5ad-pc18/10.140.0.16
Start Time:    Wed, 08 Jan 2020 14:29:08 +0530
Labels:        <none>
Annotations:   kubectl.kubernetes.io/last-applied-configuration={"apiVersion": "v1", "kind": "Pod", "metadata": {"name": "mc1", "namespace": "default"}, "spec": {"containers": [{"name": "nginx", "image": "nginx", "ports": [{"containerPort": 80}], "resources": {"requests": {"cpu": "100m"}}, "terminationGracePeriodSeconds": 30}], "restartPolicy": "Always", "schedulerName": "default-scheduler"}
Status:        Running
IP:            10.12.2.6
Containers:
  1st:
    Container ID:  docker://b08eb646f90f981cd36c605bf8fead3ca62178c7863598f
    Image:         nginx
    Image ID:      docker-pullable://nginx@sha256:36b77d8bb27ffca25c7f6f53c
    Port:         <none>
    State:         Running
      Started:     Wed, 08 Jan 2020 14:29:09 +0530
    Ready:         True
    Restart Count: 0
    Requests:
      cpu:         100m
    Environment:   <none>
    Mounts:
      /usr/share/nginx/html from html (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-xhgmm
  2nd:
    Container ID:  docker://63180b4128d477810d6062342f4b8e499de684ffd69ad245
    Image:         debian
    Image ID:      docker-pullable://debian@sha256:c99ed5d068d4f7ff36c7a6f31
    Port:         <none>
```

```

Command:
  /bin/sh
  -c
Args:
  while true; do date >> /html/index.html; sleep 1; done
State:      Running
  Started:   Wed, 08 Jan 2020 14:29:14 +0530
Ready:      True
Restart Count: 0
Requests:
  cpu:       100m
Environment: <none>
Mounts:
  /html from html (rw)
  /var/run/secrets/kubernetes.io/serviceaccount from default-token-xhgmm
Conditions:
  Type              Status
  Initialized        True
  Ready              True
  ContainersReady    True
  PodScheduled       True
Volumes:
  html:
    Type:          EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
  default-token-xhgmm:
    Type:          Secret (a volume populated by a Secret)
    SecretName:    default-token-xhgmm
    Optional:      false
QoS Class:         Burstable
Node-Selectors:    <none>
Tolerations:       node.kubernetes.io/not-ready:NoExecute for 300s
                   node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason      Age   From
  ----    -
  Normal  Scheduled   18s   default-scheduler
  Normal  Pulling     17s   kubelet, gke-k8s-lab1-default-pool-fd9ef5ad-pc18
  Normal  Pulled      17s   kubelet, gke-k8s-lab1-default-pool-fd9ef5ad-pc18
  Normal  Created     17s   kubelet, gke-k8s-lab1-default-pool-fd9ef5ad-pc18
  Normal  Started     17s   kubelet, gke-k8s-lab1-default-pool-fd9ef5ad-pc18

```

Normal	Pulling	17s	kubelet, gke-k8s-lab1-default-pool-fd9ef5ad-pc18
Normal	Pulled	13s	kubelet, gke-k8s-lab1-default-pool-fd9ef5ad-pc18
Normal	Created	12s	kubelet, gke-k8s-lab1-default-pool-fd9ef5ad-pc18
Normal	Started	12s	kubelet, gke-k8s-lab1-default-pool-fd9ef5ad-pc18

```
$ kubectl exec mc1 -c 1st -- /bin/cat /usr/share/nginx/html/index.html
```

```
...
```

```
Wed Jan  8 08:59:14 UTC 2020
```

```
Wed Jan  8 08:59:15 UTC 2020
```

```
Wed Jan  8 08:59:16 UTC 2020
```

```
$ kubectl exec mc1 -c 2nd -- /bin/cat /html/index.html
```

```
...
```

```
Wed Jan  8 08:59:14 UTC 2020
```

```
Wed Jan  8 08:59:15 UTC 2020
```

```
Wed Jan  8 08:59:16 UTC 2020
```

Cleaning Up

```
kubectl delete -f pods03.yaml
```

Contributor

[Ajeet S Raina](#)

Reviewers

[vinay agarwal](#)


[Next »](#)


Join KubeDaily


8 Members Online

Support


MEMBERS ONLINE

 h3ll_boy


 MEE6

 Nitinkashyap


Apex Legends

 ojaswa

 Parmeshwar

 prasad

 trimankaur

 vikas027

Free voice chat from Discord

Connect

[Tweets by collabnix](#)

kubelabs is maintained by **collabnix**.

This page was generated by [GitHub Pages](#).