```
spec:
  shareProcessNamespace: true
  containers:
  - name: nginx
    image: nginx
  - name: shell
    image: busybox
    stdin: true
    tty: true
```

这就意味着这个 Pod 里的容器要共享 PID Namespace。

在这个 YAML 文件中，我还定义了两个容器：一个是 Nginx 容器，另一个是开启了 tty 和 stdin 的 shell 容器。

前面介绍容器基础时，讲解过什么是 tty 和 stdin。而在 Pod 的 YAML 文件里声明开启它俩，其实等同于设置了 docker run 里的-it（-i 即 stdin，-t 即 tty）参数。

如果还是不太理解它们的作用的话，可以简单地把 tty 看作 Linux 给用户提供的一个常驻小程序，用于接收用户的标准输入，返回操作系统的标准输出。当然，为了能够在 tty 中输入信息，你还需要同时开启 stdin（标准输入流）。

于是，这个 Pod 被创建后，就可以使用 shell 容器的 tty 和这个容器进行交互了。下面实践一下：

```
$ kubectl create -f nginx.yaml
```

接下来，使用 kubectl attach 命令，连接到 shell 容器的 tty 上：

```
$ kubectl attach -it nginx -c shell
```

这样，就可以在 shell 容器里执行 ps 指令，查看所有正在运行的进程：

```
$ kubectl attach -it nginx -c shell
/ # ps ax
PID   USER     TIME  COMMAND
    1 root      0:00 /pause
    8 root      0:00 nginx: master process nginx -g daemon off;
   14 101       0:00 nginx: worker process
   15 root      0:00 sh
   21 root      0:00 ps ax
```

如上所示，在这个容器里，不仅可以看到它本身的 ps ax 指令，还可以看到 Nginx 容器的进程，以及 Infra 容器的/pause 进程。这就意味着，整个 Pod 里的每个容器的进程，对于所有容器来说都是可见的：它们共享了同一个 PID Namespace。

类似地，凡是 Pod 中的容器要共享宿主机的 Namespace，也一定是 Pod 级别的定义，比如：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
```