8.3 Ingress verwenden

Nachdem wir nun einen Ingress-Controller konfiguriert haben, wollen wir es auch einmal ausprobieren. Zuerst werden wir ein paar Upstream-Services anlegen (manchmal auch als »Backend«-Services bezeichnet), um damit herumzuspielen:

```
$ kubectl create deployment be-default \
  --image=gcr.io/kuar-demo/kuard-amd64:blue
$ kubectl scale --replicas=3 deployments/be-default
$ kubectl expose deployment be-default --port 8080
$ kubectl create deployment alpaca \
  --image=gcr.io/kuar-demo/kuard-amd64:green
$ kubectl scale --replicas=3 deployments/alpaca
$ kubectl expose deployment alpaca --port 8080
$ kubectl create deployment bandicoot \
  --image=gcr.io/kuar-demo/kuard-amd64:purple
$ kubectl scale --replicas=3 deployments/bandicoot
$ kubectl expose deployment bandicoot -port 8080
$ kubectl get services -o wide

NAME            CLUSTER-IP    ... PORT(S)  ... SELECTOR
alpaca          10.115.245.13 ... 8080/TCP ... run=alpaca
bandicoot       10.115.242.3  ... 8080/TCP ... run=bandicoot
be-default      10.115.246.6  ... 8080/TCP ... run=be-default
kubernetes      10.115.240.1  ... 443/TCP  ... <none>
```

8.3.1 Einfachste Anwendung

Die einfachste Anwendung von Ingress ist, einfach blind alles an einen Upstream-Service weiterzugeben. Es gibt in `kubectl` nur wenig Unterstützung durch imperative Befehle für Ingress, daher werden wir gleich mit einer YAML-Datei beginnen (siehe Listing 8–1).

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-ingress
spec:
  defaultBackend:
    service:
      name: alpaca
      port:
        number: 8080
```

Listing 8–1 *simple-ingress.yaml*