**Service Discovery**

1. kubectl create deployment alpaca-prod --image=gcr.io/kuar-demo/kuard-amd64:1 --replicas=3 –port=8080
2. kubectl expose deployment alpaca-prod
3. kubectl create deployment bandicoot-prod --image=gcr.io/kuar-demo/kuard-amd64:2 --replicas=2 –port=8080
4. kubectl expose deployment bandicoot-prod
5. kubectl get services -o wide
   1. After running these commands, we have three services. The ones we just created are alpaca-prod and bandicoot-prod. The kubernetes service is automatically created for you so that you can find and talk to the Kubernetes API from within the app.
   2. Furthermore, that service is assigned a new type of virtual IP called a cluster IP. This is a special IP address the system will load-balance across all of the pods that are identified by the selector.
6. open minikube dashboard in a new terminal tab
7. check section on pods, deployments, ReplicaSets, nodes
8. ALPACA\_POD=$(kubectl get pods -l app=alpaca-prod -o jsonpath='{.items[0].metadata.name}')
9. kubectl port-forward $ALPACA\_POD 48858:8080
10. Open on browser <http://localhost:48858>
11. Go to DNS Query
    1. Type ‘alpaca-prod’ in name field
    2. Press query button
12. The full DNS name here is alpaca-prod.default.svc.cluster.local.. Let’s break this down:
    1. alpaca-prod
       1. The name of the service in question.
    2. default
       1. The namespace that this service is in.
    3. svc
       1. Recognizing that this is a service. This allows Kubernetes to expose other types of things as DNS in the future.
    4. cluster.local
       1. The base domain name for the cluster. This is the default and what you will see for most clusters. Administrators may change this to allow unique DNS names across multiple clusters.
13. When referring to a service in your own namespace you can just use the service name (alpaca-prod). You can also refer to a service in another namespace with alpaca-prod.default. And, of course, you can use the fully qualified service name (alpaca-prod.default.svc.cluster.local.). Try each of these out in the “DNS Query” section of kuard
14. Open the dashboard tab of minikube
15. Go to the deployment section and find alpaca-prod
16. Click edit
17. Add the following section
    1. Specified in “readiness check.txt” shared
    2. This sets up the pods this deployment will create so that they will be checked for readiness via an HTTP GET to /ready on port 8080. This check is done every 2 seconds starting as soon as the pod comes up. If three successive checks fail, then the pod will be considered not ready. However, if only one check succeeds, then the pod will again be considered ready.
18. Repeat steps 8, 9 and 10
19. Expand “Readiness Probe” section
    1. You should see this page update every time there is a new readiness check from the system, which should happen every 2 seconds.
20. Open a new terminal window
    1. kubectl get endpoints alpaca-prod –watch
21. In the browser window of kuard, click Fail
22. The server starts returning 500 for failure
23. After few attempts the server is removed from the endpoints of service
    1. Observe in the watch terminal
24. Click succeed in the browser
25. Observe that status code changes and the watch terminal shows server added
26. Go to the service section in dashboard
27. Find alpaca-prod and edit it
    1. Change the ‘spec.type’ to NodePort
28. Execute the following on terminal window
    1. kubectl describe service alpaca-prod
    2. notice that system assigned a port to NodePort
29. Some applications (and the system itself) want to be able to use services without using a cluster IP. This is done with another type of object called Endpoints. For every Service object, Kubernetes creates a buddy Endpoints object that contains the IP addresses for that service:
    1. kubectl describe endpoints alpaca-prod
    2. observe the IPAddresses associated with the service
30. in a terminal window start the endpoint watch
    1. kubectl get endpoints alpaca-prod –watch
31. Delete the alpaca-prod deployment
    1. kubectl delete deployment alpaca-prod
32. Observe the changes in the watch terminal
33. Redeploy using step 1
34. Check the watch terminal again
35. Kubernetes services are built on top of label selectors over pods. That means that you can use the Kubernetes API to do rudimentary service discovery without using a Service object at all
    1. kubectl get pods -o wide --show-labels
    2. This will not be great if we have huge number of pods. We need to apply filters
       1. kubectl get pods -o wide --selector=app=alpaca-prod
36. BANDICOOT\_POD=$(kubectl get pods -l app=bandicoot-prod -o jsonpath='{.items[0].metadata.name}')
37. kubectl port-forward $BANDICOOT\_POD 48858:8080
38. log into the browser with <http://localhost:48858>
39. Go to ‘Server Env’ section and check out the environment variables
40. The two main environment variables to use are ALPACA\_PROD\_SERVICE\_HOST and ALPACA\_PROD\_SERVICE\_PORT. The other environment variables are created to be compatible with (now deprecated) Docker link variables.
41. kubectl delete services,deployments -l app