RX-M Cloud Native Consulting

Advanced Kubernetes

==========

Lab 5 – DNS

DNS is a foundational service of the public Internet and private networks alike. So much so, that it is often overlooked. DNS also plays a critical role in most Kubernetes clusters.

Our current cluster is running:

- etcd: the cluster state store
- api-server: the central API of the cluster
- scheduler: binds pods to nodes
- controller manager: creates pod replicas and manages deployments
- kubelets: to run pods
- kube-proxy: to configure services in iptables

Though the core cluster services are running, we do not have the ability to discover newly created services by name.

To demonstrate this we will run a simple service to illustrate the problem and, after installing DNS, the solution.

Make sure you have all of the components from the previous lab running.

```
user@nodea:~$ ps -aefo comm

COMMAND
bash
bash
 \_ ps
```

```
bash
 \_ sudo
     \_ kube-proxy
bash
 \_ kube-controller
bash
 \_ kube-scheduler
bash
 \_ sudo
     \_ kubelet
         \_ journalctl
bash
 \_ sudo
     \_ kube-apiserver
bash
 \_ etcd
Xorg
agetty

user@nodea:~$
```

Sample cluster start script (k8s.sh), nodea:

```
user@nodea:~$ cat k8s.sh

### Clear logs
cd ~
rm *.log

### etcd
sudo rm -rf /var/lib/etcd
rm -rf ~user/default.etcd/
etcd > etcd.log 2>&1 &

### api-server
sudo ~user/k8s/_output/bin/kube-apiserver --etcd-servers=http://localhost:2379 --service-cluster-ip-
range=10.0.0.0/16 --insecure-bind-address=0.0.0.0 > kube-apiserver.log 2>&1 &

### controller manager
~user/k8s/_output/bin/kube-controller-manager --kubeconfig=nodea.kubeconfig > kube-controller-manager.log 2>&1 &
```

```
### scheduler
~user/k8s/_output/bin/kube-scheduler --master=http://nodea:8080 > kube-scheduler 2>&1 &

### kubelet on nodea
sudo rm -rf /var/lib/kubelet
sudo ~user/k8s/_output/bin/kubelet --kubeconfig=nodea.kubeconfig --require-kubeconfig --allow-privileged=true >
kubelet.log 2>&1 &

### kube-proxy on nodea
sudo ~user/k8s/_output/bin/kube-proxy --config=kube-proxy-config > kube-proxy.log 2>&1 &

### reset kubectl
kubectl config use-context local

user@nodea:~$
```

nodeb:

```
### kubelet on nodeb
sudo rm -rf /var/lib/kubelet
sudo ~user/kube-bin/kubelet --kubeconfig=nodeb.kubeconfig --require-kubeconfig --allow-privileged=true >
kubelet.log 2>&1 &

### kube-proxy on nodeb
sudo ~user/kube-bin/kube-proxy --config=kube-proxy-config 2>&1 &

### reset kubectl
kubectl config use-context local
```

> N.B. if your LAB VMs have been disconnected and reconnected to the external network (e.g. laptop taken home) you will need to recreate your cluster inter-node routes (the network manager removes temporary routes each time DHCP data is reassigned).

1. Run a simple service and try to discover it

As a first step we will run a simple nginx service and then try to reach the web servers from a client pod. To begin run an nginx-based deployment with two pods (same as previous lab):

```
user@nodea:~$ cat testdepl.yaml

apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80

user@nodea:~$
```

```
user@nodea:~$ kubectl create -f testdepl.yaml

deployment "nginx-deployment" created

user@nodea:~$
```

Now wait until the deployment is fully running:

```
user@nodea:~$ kubectl get deploy,rs,pod

NAME                     DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deploy/nginx-deployment   2         2         2            2           5s

NAME                               DESIRED   CURRENT   READY     AGE
rs/nginx-deployment-171375908   2         2         2         5s

NAME                                     READY     STATUS    RESTARTS   AGE
po/nginx-deployment-171375908-0bggz   1/1       Running   0          5s
```

```
po/nginx-deployment-171375908-g1cms   1/1       Running   0          5s

user@nodea:~$
```

Now create a service to front end the pods (modify ports information):

```
user@nodea:~$ vi websvc.yaml
user@nodea:~$ cat websvc.yaml

apiVersion: v1
kind: Service
metadata:
  name: websvc
spec:
  ports:
    - port: 80
  selector:
    app: nginx

user@nodea:~$
```

```
user@nodea:~$ kubectl create -f websvc.yaml

service "websvc" created

user@nodea:~$
```

```
user@nodea:~$ kubectl get service --all-namespaces

NAMESPACE   NAME         CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
default     kubernetes   10.0.0.1      <none>        443/TCP   1m
default     websvc       10.0.70.246   <none>        80/TCP    4s

user@nodea:~$
```

Now that we have a named service running we can start up a client pod to access it with. Run a busybox container to act as the client:

```
user@nodea:~$ vim clientpod.yaml
user@nodea:~$ cat clientpod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: clientpod
spec:
  containers:
  - name: clientpod
    image: busybox
    command: ['/bin/tail', '-f', '/dev/null']

user@nodea:~$
```

```
user@nodea:~$ kubectl create -f clientpod.yaml

pod "clientpod" created

user@nodea:~$
```

```
user@nodea:~$ kubectl get pods clientpod

NAME        READY     STATUS    RESTARTS   AGE
clientpod   1/1       Running   0          3m

user@nodea:~$
```

With our dummy client pod running we can exec a shell and try to ping the nginx service.

```
user@nodea:~$ kubectl exec -it clientpod sh

/ #
```

Try to ping the service IP:

```
/ # ping -c 1 10.0.70.246

PING 10.0.70.246 (10.0.70.246): 56 data bytes

--- 10.0.70.246 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

/ #
```

That never works. The service IP is a virtual IP and no one is listening for ICMP (ping) traffic there. Try to GET the the root route on port 80:

```
/ # wget -S 10.0.70.246 -O /dev/null

Connecting to 10.0.70.246 (10.0.70.246:80)
  HTTP/1.1 200 OK
  Server: nginx/1.7.9
  Date: Thu, 31 Aug 2017 01:08:19 GMT
  Content-Type: text/html
  Content-Length: 612
  Last-Modified: Tue, 23 Dec 2014 16:25:09 GMT
  Connection: close
  ETag: "54999765-264"
  Accept-Ranges: bytes

null                100% |********************************|   612   0:00:00 ETA

/ #
```

When we connect to TCP port 80 (as defined in the service) we get forwarded to one of the nginx pods and receive the resulting HTML.

Now try making use of the service by its name:

```
/ # wget -S websvc -O /dev/null

wget: bad address 'websvc'

/ #
```

or

```
/ # wget -S websvc -O /dev/null

Connecting to websvc (198.105.244.104:80)
wget: error getting response: Resource temporarily unavailable

/ #
```

```
/ # nslookup websvc

Server:    172.16.151.2
Address 1: 172.16.151.2

nslookup: can't resolve 'websvc'

/ #
```

or

```
/ # nslookup websvc

Server:    172.16.151.2
Address 1: 172.16.151.2

Name:      websvc
Address 1: 198.105.244.104

/ #
```

```
/ # exit

user@nodea:~$
```

We can not reach the service by name. When you try to connect to a name rather than an IP address the Linux resolver tries to turn that name into an IP address by looking in the `/etc/hosts` file (inside the container in this case) and then it tries to use DNS. In the example above the DNS server is listed as 172.16.151.2. This is the bogus setting we have been using when starting our kubelets. The kubelets then tell the Docker engine to populate the `/etc/resolv.conf` file with this DNS server address. This is problematic because clients are usually better off parameterizing connections by service name rather than by the more fragile IP address.

What we need is a real DNS service that can supply Kubernetes service VIPs in response to service name lookups.

2. Enter KubeDNS

Kubernetes offers an integrated DNS server called KubeDNS, derived from the older SkyDNS. KubeDNS is a distributed service for announcement and discovery of services built on top of etcd. It allows DNS queries to directly discover available Kubernetes Service and POD IPs. KubeDNS supports SRV records as well.

Most installers treat KubeDNS as a standard service, launching it automatically as a cluster add-on. KubeDNS is generally run as a Deployment and Service on the cluster. Kubelets are configured to tell individual containers to use the DNS Service's IP to resolve DNS names.

In this step we'll run and test Kubernetes DNS. Sadly, KubeDNS installation is completely undocumented. Kubeadm and other tools take their own approach to the installation, Kubeadm actually emits the yaml configs for running KubeDNS from golang code directly (!). We will base our installation on the SaltStack scripts included in the Kubernetes source tree here:

```
user@nodea:~$ ls -l ~/k8s/cluster/addons/dns/

total 64
-rw-rw-r-- 1 user user  731 Aug 17 01:29 kubedns-cm.yaml
-rw-rw-r-- 1 user user 5327 Aug 17 01:29 kubedns-controller.yaml.base
-rw-rw-r-- 1 user user 5412 Aug 17 01:29 kubedns-controller.yaml.in
-rw-rw-r-- 1 user user 5352 Aug 17 01:29 kubedns-controller.yaml.sed
-rw-rw-r-- 1 user user  187 Aug 17 01:29 kubedns-sa.yaml
-rw-rw-r-- 1 user user 1037 Aug 17 01:29 kubedns-svc.yaml.base
-rw-rw-r-- 1 user user 1106 Aug 17 01:29 kubedns-svc.yaml.in
-rw-rw-r-- 1 user user 1094 Aug 17 01:29 kubedns-svc.yaml.sed
-rw-rw-r-- 1 user user 1138 Aug 17 01:29 Makefile
-rw-rw-r-- 1 user user   56 Aug 17 01:29 OWNERS
-rw-rw-r-- 1 user user 2106 Aug 17 01:29 README.md
-rw-rw-r-- 1 user user  238 Aug 17 01:29 transforms2salt.sed
-rw-rw-r-- 1 user user  211 Aug 17 01:29 transforms2sed.sed

user@nodea:~$
```

To begin create a working directory where we can create our Kubernetes configuration manifests for the DNS service:

```
user@nodea:~$ mkdir -p kubedns

user@nodea:~$
```

```
user@nodea:~$ cd kubedns/

user@nodea:~/kubedns$
```

The KubeDNS manifests will need to be customized with our local cluster information. Create a deployment with the following spec, replacing "nodea" in the flag `--kube-master-url=http://nodea:8080` with the IP of your master node:

```
user@nodea:~/kubedns$ vim dnsdep.yaml
user@nodea:~/kubedns$ cat dnsdep.yaml

apiVersion: apps/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  labels:
    component: kube-dns
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    name: kube-dns
    tier: node
  name: kube-dns
  namespace: kube-system
spec:
  replicas: 1
  selector:
    matchLabels:
      component: kube-dns
      k8s-app: kube-dns
      kubernetes.io/cluster-service: "true"
      name: kube-dns
      tier: node
```

```
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      annotations:
        scheduler.alpha.kubernetes.io/affinity: '{"nodeAffinity":{"requiredDuringSchedulingIgnoredDuringExecution":
{"nodeSelectorTerms":[{"matchExpressions":[{"key":"beta.kubernetes.io/arch","operator":"In","values":["amd64"]}]}]}}}'
        scheduler.alpha.kubernetes.io/tolerations: '[{"key":"dedicated","value":"master","effect":"NoSchedule"}]'
      labels:
        component: kube-dns
        k8s-app: kube-dns
        kubernetes.io/cluster-service: "true"
        name: kube-dns
        tier: node
    spec:
      containers:
      - args:
        - --domain=cluster.local
        - --dns-port=10053
        - --kube-master-url=http://nodea:8080
        - --v=2
        env:
        - name: PROMETHEUS_PORT
          value: "10055"
        image: gcr.io/google_containers/kubedns-amd64:1.9
        imagePullPolicy: IfNotPresent
        livenessProbe:
          failureThreshold: 5
          httpGet:
            path: /healthz-kubedns
            port: 8080
            scheme: HTTP
          initialDelaySeconds: 60
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        name: kube-dns
        ports:
        - containerPort: 10053
          name: dns-local
```

```
      protocol: UDP
    - containerPort: 10053
      name: dns-tcp-local
      protocol: TCP
    - containerPort: 10055
      name: metrics
      protocol: TCP
    readinessProbe:
      failureThreshold: 3
      httpGet:
        path: /readiness
        port: 8081
        scheme: HTTP
      initialDelaySeconds: 3
      periodSeconds: 10
      successThreshold: 1
      timeoutSeconds: 5
    resources:
      limits:
        memory: 170Mi
      requests:
        cpu: 100m
        memory: 70Mi
    terminationMessagePath: /dev/termination-log
  - args:
    - --cache-size=1000
    - --no-resolv
    - --server=127.0.0.1#10053
    - --log-facility=-
    image: gcr.io/google_containers/kube-dnsmasq-amd64:1.4
    imagePullPolicy: IfNotPresent
    livenessProbe:
      failureThreshold: 5
      httpGet:
        path: /healthz-dnsmasq
        port: 8080
        scheme: HTTP
      initialDelaySeconds: 60
      periodSeconds: 10
      successThreshold: 1
      timeoutSeconds: 5
    name: dnsmasq
    ports:
```

```
      - containerPort: 53
        name: dns
        protocol: UDP
      - containerPort: 53
        name: dns-tcp
        protocol: TCP
      resources:
        requests:
          cpu: 150m
          memory: 10Mi
      terminationMessagePath: /dev/termination-log
    - args:
      - --v=2
      - --logtostderr
      image: gcr.io/google_containers/dnsmasq-metrics-amd64:1.0
      imagePullPolicy: IfNotPresent
      livenessProbe:
        failureThreshold: 5
        httpGet:
          path: /metrics
          port: 10054
          scheme: HTTP
        initialDelaySeconds: 60
        periodSeconds: 10
        successThreshold: 1
        timeoutSeconds: 5
      name: dnsmasq-metrics
      ports:
      - containerPort: 10054
        name: metrics
        protocol: TCP
      resources:
        requests:
          memory: 10Mi
      securityContext:
        runAsUser: 0
      terminationMessagePath: /dev/termination-log
    - args:
      - --cmd=nslookup kubernetes.default.svc.cluster.local 127.0.0.1 >/dev/null
      - --url=/healthz-dnsmasq
      - --cmd=nslookup kubernetes.default.svc.cluster.local 127.0.0.1:10053 >/dev/null
      - --url=/healthz-kubedns
      - --port=8080
```

```
        - --quiet
        image: gcr.io/google_containers/exechealthz-amd64:1.2
        imagePullPolicy: IfNotPresent
        name: healthz
        ports:
        - containerPort: 8080
          protocol: TCP
        resources:
          limits:
            memory: 50Mi
          requests:
            cpu: 10m
            memory: 50Mi
        terminationMessagePath: /dev/termination-log
      dnsPolicy: Default
      restartPolicy: Always
      terminationGracePeriodSeconds: 30

user@nodea:~/kubedns$
```

This is an interesting deployment leveraging many useful features. Look up any keys you are not familiar with here:

https://kubernetes.io/docs/resources-reference/v1.7#deployment-v1beta1

The kube-dns service and deployment are typically run in a separate namespace from the normal applications running on the cluster. In particular the "kube-system" namespace is the standard namespace for cluster addons. Verify kube-system namespace exits and create it if not:

```
user@nodea:~/kubedns$ kubectl get namespace

NAME          STATUS    AGE
default       Active    18m
kube-public   Active    18m
kube-system   Active    18m

user@nodea:~/kubedns$
```

ONLY if you do not see `kube-system` in the output above: add the kube-system namespace using a config:

```
user@nodea:~/kubedns$ vim ksns.yaml
user@nodea:~/kubedns$ cat ksns.yaml
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: kube-system

user@nodea:~/kubedns$
```

```
user@nodea:~/kubedns$ kubectl create -f ksns.yaml

namespace "kube-system" created

user@nodea:~/kubedns$
```

```
user@nodea:~/kubedns$ kubectl get namespace

NAME          STATUS    AGE
default       Active    18m
kube-public   Active    18m
kube-system   Active    18m

user@nodea:~/kubedns$
```

Now we can create the Service spec for the dns service:

```
user@nodea:~/kubedns$ vim dnssvc.yaml
user@nodea:~/kubedns$ cat dnssvc.yaml

apiVersion: v1
kind: Service
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "KubeDNS"
```

```
spec:
  selector:
    k8s-app: kube-dns
  clusterIP: 10.0.0.10
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
    port: 53
    protocol: TCP

user@nodea:~/kubedns$
```

To start things, run the DNS deployment:

```
user@nodea:~/kubedns$ kubectl create -f dnsdep.yaml

deployment "kube-dns" created

user@nodea:~/kubedns$
```

```
user@nodea:~/kubedns$ kubectl get pod --all-namespaces

NAMESPACE     NAME                                READY     STATUS    RESTARTS   AGE
default       clientpod                           1/1       Running   0          7s
default       nginx-deployment-171375908-0bggz    1/1       Running   0          16m
default       nginx-deployment-171375908-g1cms    1/1       Running   0          16m
kube-system   kube-dns-2188085985-1dblc           4/4       Running   0          2m

user@nodea:~/kubedns$
```

Now create the DNS service:

```
user@nodea:~/kubedns$ kubectl create -f dnssvc.yaml

service "kube-dns" created
```

```
user@nodea:~/kubedns$
```

```
user@nodea:~/kubedns$ kubectl get service --all-namespaces

NAMESPACE     NAME         CLUSTER-IP    EXTERNAL-IP   PORT(S)         AGE
default       kubernetes   10.0.0.1      <none>        443/TCP         18m
default       websvc       10.0.70.246   <none>        80/TCP          16m
kube-system   kube-dns     10.0.0.10     <none>        53/UDP,53/TCP   21s

user@nodea:~/kubedns$
```

```
user@nodea:~/kubedns$ kubectl describe service kube-dns --namespace=kube-system

Name:                   kube-dns
Namespace:              kube-system
Labels:                 addonmanager.kubernetes.io/mode=Reconcile
                        k8s-app=kube-dns
                        kubernetes.io/cluster-service=true
                        kubernetes.io/name=KubeDNS
Annotations:            <none>
Selector:               k8s-app=kube-dns
Type:                   ClusterIP
IP:                     10.0.0.10
Port:                   dns     53/UDP
Endpoints:              172.17.0.3:53
Port:                   dns-tcp 53/TCP
Endpoints:              172.17.0.3:53
Session Affinity:       None
Events:                 <none>

user@nodea:~/kubedns$
```

Check with the cluster to see if it knows about DNS:

```
user@nodea:~/kubedns$ kubectl cluster-info
```

```
Kubernetes master is running at http://localhost:8080
KubeDNS is running at http://localhost:8080/api/v1/namespaces/kube-system/services/kube-dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

user@nodea:~/kubedns$
```

Perfect, DNS is up and running!

Take a look at the Docker containers running on the DNS pod node:

```
user@nodea:~/kubedns$ docker container ls

CONTAINER ID        IMAGE                                                COMMAND                  CREATED
STATUS              PORTS               NAMES
bcd87d8ddd5b        gcr.io/google_containers/exechealthz-amd64       "/exechealthz '--c..."   4 minutes ago
Up 4 minutes                            k8s_healthz_kube-dns-2188085985-1dblc_kube-system_b2d86e4c-8dee-11e7-ae9d-
000c293215a2_0
a85fac257d2b        gcr.io/google_containers/dnsmasq-metrics-amd64   "/dnsmasq-metrics ..."   4 minutes ago
Up 4 minutes                            k8s_dnsmasq-metrics_kube-dns-2188085985-1dblc_kube-system_b2d86e4c-8dee-
11e7-ae9d-000c293215a2_0
48fdb7268cb9        gcr.io/google_containers/kube-dnsmasq-amd64      "/usr/sbin/dnsmasq..."   4 minutes ago
Up 4 minutes                            k8s_dnsmasq_kube-dns-2188085985-1dblc_kube-system_b2d86e4c-8dee-11e7-ae9d-
000c293215a2_0
8527af73f725        gcr.io/google_containers/kubedns-amd64           "/kube-dns --domai..."   4 minutes ago
Up 4 minutes                            k8s_kube-dns_kube-dns-2188085985-1dblc_kube-system_b2d86e4c-8dee-11e7-
ae9d-000c293215a2_0
df9787f0b7d4        gcr.io/google_containers/pause-amd64:3.0         "/pause"                 4 minutes ago
Up 4 minutes                            k8s_POD_kube-dns-2188085985-1dblc_kube-system_b2d86e4c-8dee-11e7-ae9d-
000c293215a2_0
6cbc0def7df8        nginx                                            "nginx -g 'daemon ..."   18 minutes ago
Up 18 minutes                           k8s_nginx_nginx-deployment-171375908-g1cms_default_bd573e75-8dec-11e7-
ae9d-000c293215a2_0
43199e50209f        gcr.io/google_containers/pause-amd64:3.0         "/pause"                 18 minutes ago
Up 18 minutes                           k8s_POD_nginx-deployment-171375908-g1cms_default_bd573e75-8dec-11e7-ae9d-
000c293215a2_0

user@nodea:~/kubedns$
```

Dump the logs of your kubedns container:

```
user@nodea:~/kubedns$ docker container logs 8527af73f725

I0831 01:50:00.584999       7 dns.go:42] version: v1.6.0-alpha.0.680+3872cb93abf948-dirty
I0831 01:50:00.585565       7 server.go:107] Using http://172.16.151.203:8080 for kubernetes master, kubernetes
API: v1
I0831 01:50:00.585767       7 server.go:63] ConfigMap not configured, using values from command line flags
I0831 01:50:00.585800       7 server.go:113] FLAG: --alsologtostderr="false"
I0831 01:50:00.585810       7 server.go:113] FLAG: --config-map=""
I0831 01:50:00.585813       7 server.go:113] FLAG: --config-map-namespace="kube-system"
I0831 01:50:00.585817       7 server.go:113] FLAG: --dns-bind-address="0.0.0.0"
I0831 01:50:00.585820       7 server.go:113] FLAG: --dns-port="10053"
I0831 01:50:00.585825       7 server.go:113] FLAG: --domain="cluster.local."
I0831 01:50:00.585828       7 server.go:113] FLAG: --federations=""
I0831 01:50:00.585832       7 server.go:113] FLAG: --healthz-port="8081"
I0831 01:50:00.585835       7 server.go:113] FLAG: --kube-master-url="http://172.16.151.203:8080"
I0831 01:50:00.585840       7 server.go:113] FLAG: --kubecfg-file=""
I0831 01:50:00.585842       7 server.go:113] FLAG: --log-backtrace-at=":0"
I0831 01:50:00.585850       7 server.go:113] FLAG: --log-dir=""
I0831 01:50:00.585853       7 server.go:113] FLAG: --log-flush-frequency="5s"
I0831 01:50:00.585856       7 server.go:113] FLAG: --logtostderr="true"
I0831 01:50:00.585859       7 server.go:113] FLAG: --stderrthreshold="2"
I0831 01:50:00.585882       7 server.go:113] FLAG: --v="2"
I0831 01:50:00.585885       7 server.go:113] FLAG: --version="false"
I0831 01:50:00.585890       7 server.go:113] FLAG: --vmodule=""
I0831 01:50:00.587738       7 server.go:155] Starting SkyDNS server (0.0.0.0:10053)
I0831 01:50:00.590356       7 server.go:165] Skydns metrics enabled (/metrics:10055)
I0831 01:50:00.590373       7 dns.go:144] Starting endpointsController
I0831 01:50:00.590377       7 dns.go:147] Starting serviceController
I0831 01:50:00.590383       7 dns.go:163] Waiting for Kubernetes service
I0831 01:50:00.590387       7 dns.go:169] Waiting for service: default/kubernetes
I0831 01:50:00.595748       7 logs.go:41] skydns: ready for queries on cluster.local. for tcp://0.0.0.0:10053
[rcache 0]
I0831 01:50:00.595765       7 logs.go:41] skydns: ready for queries on cluster.local. for udp://0.0.0.0:10053
[rcache 0]
I0831 01:50:00.601793       7 dns.go:274] New service: kubernetes
I0831 01:50:00.602073       7 dns.go:384] Added SRV record &{Host:kubernetes.default.svc.cluster.local. Port:443
Priority:10 Weight:10 Text: Mail:false Ttl:30 TargetStrip:0 Group: Key:}
I0831 01:50:00.602153       7 dns.go:274] New service: websvc
I0831 01:50:00.602266       7 server.go:126] Setting up Healthz Handler (/readiness)
I0831 01:50:00.602276       7 server.go:131] Setting up cache handler (/cache)
```

```
I0831 01:50:00.602279       7 server.go:120] Status HTTP port 8081
I0831 01:52:49.324314       7 dns.go:274] New service: kube-dns
I0831 01:52:49.324387       7 dns.go:384] Added SRV record &{Host:kube-dns.kube-system.svc.cluster.local. Port:53
Priority:10 Weight:10 Text: Mail:false Ttl:30 TargetStrip:0 Group: Key:}
I0831 01:52:49.324406       7 dns.go:384] Added SRV record &{Host:kube-dns.kube-system.svc.cluster.local. Port:53
Priority:10 Weight:10 Text: Mail:false Ttl:30 TargetStrip:0 Group: Key:}
I0831 01:55:01.541728       7 dns.go:274] New service: kubernetes
I0831 01:55:01.541844       7 dns.go:384] Added SRV record &{Host:kubernetes.default.svc.cluster.local. Port:443
Priority:10 Weight:10 Text: Mail:false Ttl:30 TargetStrip:0 Group: Key:}
I0831 01:55:01.541973       7 dns.go:274] New service: websvc
I0831 01:55:01.542009       7 dns.go:274] New service: kube-dns
I0831 01:55:01.542033       7 dns.go:384] Added SRV record &{Host:kube-dns.kube-system.svc.cluster.local. Port:53
Priority:10 Weight:10 Text: Mail:false Ttl:30 TargetStrip:0 Group: Key:}
I0831 01:55:01.542058       7 dns.go:384] Added SRV record &{Host:kube-dns.kube-system.svc.cluster.local. Port:53
Priority:10 Weight:10 Text: Mail:false Ttl:30 TargetStrip:0 Group: Key:}

user@nodea:~/kubedns$
```

As you can see, kubedns is busy creating SRV records for all of the services it has discovered through the API Server.

Now lets try doing some lookups with the new DNS server using its pod ip as the DNS host target:

```
user@nodea:~/kubedns$ nslookup websvc.default.svc.cluster.local. 172.17.0.3

Server:         172.17.0.3
Address:        172.17.0.3#53

Name:   websvc.default.svc.cluster.local
Address: 10.0.70.246

user@nodea:~/kubedns$
```

nslookup returned the service IP, 10.0.70.246 in the above case, nice.

Service names are fully qualified in Kubernetes by appending their namespace, the "svc" subdomain, and the domain name of the cluster. The cluster domain name is set through the kubelet (e.g. `--cluster_domain=cluster.local`) and defaults to cluster.local.

Try using dig to look up some other service IPs:

```
user@nodea:~/kubedns$ dig +short websvc.default.svc.cluster.local. @172.17.0.3
10.0.243.7
user@nodea:~/kubedns$
```

```
user@nodea:~/kubedns$ dig +short kubernetes.default.svc.cluster.local. @172.17.0.3
10.0.0.1
user@nodea:~/kubedns$
```

```
user@nodea:~/kubedns$ dig +short kube-dns.kube-system.svc.cluster.local. @172.17.0.3
10.0.0.10
user@nodea:~/kubedns$
```

3. Test lookups from inside a pod

So our DNS server is up and running but our overall cluster configuration is not yet optimal. Let's try accessing the DNS server from inside a pod again. Shell into your clientpod:

```
user@nodea:~/kubedns$ kubectl exec -it clientpod sh
/ #
```

Now rerun the nslookup command you ran on the host by try using not only the DNS pod IP but also the DNS service IP:

```
/ # nslookup websvc.default.svc.cluster.local. 172.17.0.3
Server:    172.17.0.3
```

```
Address 1: 172.17.0.3 kube-dns-2188085985-1dblc

Name:      websvc.default.svc.cluster.local.
Address 1: 10.0.70.246 websvc.default.svc.cluster.local

/ #
```

```
/ # nslookup websvc.default.svc.cluster.local. 10.0.0.10

Server:    10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name:      websvc.default.svc.cluster.local.
Address 1: 10.0.70.246 websvc.default.svc.cluster.local

/ #
```

They both work, great. Now try a lookup with no DNS target:

```
/ # nslookup websvc.default.svc.cluster.local.

Server:    172.16.151.2
Address 1: 172.16.151.2

nslookup: can't resolve 'websvc.default.svc.cluster.local.'

/ #
```

This fails because our default DNS server IP is bogus. Examine the container's resolv.conf:

```
/ # cat /etc/resolv.conf

nameserver 172.16.151.2
search localdomain

/ #
```

The old IP was just a place holder we gave the kubelet on startup with the `--cluster-dns` switch.

Update the `resolv.conf` to use the correct DNS service VIP and test some lookups:

```
/ # vi /etc/resolv.conf
/ # cat /etc/resolv.conf

nameserver 10.0.0.10 172.16.151.2
search localdomain

/ #
```

```
/ # nslookup websvc.default.svc.cluster.local.

Server:    10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name:      websvc.default.svc.cluster.local.
Address 1: 10.0.70.246 websvc.default.svc.cluster.local

/ #
```

An improvement, now we don't need to specify the DNS server address. Now try looking up a service without the qualifying suffix:

```
/ # nslookup websvc

Server:    10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

nslookup: can't resolve 'websvc'

/ #
```

This fails because the default search suffix is not set correctly. Update the `resolv.conf` and try again:

```
/ # vi /etc/resolv.conf
/ # cat /etc/resolv.conf

search default.svc.cluster.local.
nameserver 10.0.0.10 172.16.151.2

/ #
```

```
/ # nslookup websvc

Server:    10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name:      websvc
Address 1: 10.0.70.246 websvc.default.svc.cluster.local

/ #
```

Magic! Now we can achieve our original goal, having one pod just reach another using a service name:

```
/ # wget -qO- websvc

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

```
<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

/ #
```

```
/ # exit

user@nodea:~/kubedns$
```

Everything is working perfectly in this pod but all of the new pods we create will still get a broken `resolv.conf`.

Let's update the kubelet command to fix the search suffix and the DNS server IP.

4. Update the kubelet configuration

To update the kubelet, stop the currently running kubelet and pass it the new `--cluster-dns=10.0.0.10 --cluster_domain=cluster.local` parameters:

```
W0322 17:35:23.613487  106540 container_manager_linux.go:731] MemoryAccounting not enabled for pid: 106540

^C

user@nodea:~/kubedns$
```

```
user@nodea:~$ sudo /home/user/k8s/_output/bin/kubelet \
--kubeconfig=nodea.kubeconfig \
--require-kubeconfig \
--allow-privileged=true \
--cluster-dns=10.0.0.10 \
```

```
--cluster_domain=cluster.local > ~/kubelet.log 2>&1 &

[1] 9666

user@nodea:~$
```

Perform the same operation on the other node:

```
user@nodeb:~$ sudo ./kube-bin/kubelet \
--kubeconfig=nodeb.kubeconfig \
--require-kubeconfig \
--allow-privileged=true \
--cluster-dns=10.0.0.10 \
--cluster_domain=cluster.local > kubelet.log 2>&1 &

user@nodeb:~$
```

Now run a new test pod and try out some DNS:

```
user@nodea:~$ kubectl run newpod --image=busybox --command -- /bin/tail -f /dev/null

deployment "newpod" created

user@nodea:~$
```

```
user@nodea:~$ kubectl get pod

NAME                              READY     STATUS    RESTARTS   AGE
clientpod                         1/1       Running   0          33m
newpod-3723177113-dbvcr           1/1       Running   0          6s
nginx-deployment-171375908-0bggz  1/1       Running   0          49m
nginx-deployment-171375908-g1cms  1/1       Running   0          49m

user@nodea:~$
```

```
user@nodea:~$ kubectl exec -it newpod-3723177113-dbvcr sh

/ # wget -qO- websvc

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

/ #
```

```
/ # cat /etc/resolv.conf

nameserver 10.0.0.10
search default.svc.cluster.local svc.cluster.local cluster.local localdomain
options ndots:5

/ #
```

```
/ # exit

user@nodea:~$
```

Congratulations, you have built a full featured Kubernetes cluster by hand! Now when you have problems in k8s clusters installed by a tool, you'll know who's responsible for what; how to discover problems and how to get the configuration working again.