RX-M Cloud Native Consulting

Advanced Kubernetes

Lab 4 – Services

In the last lab our cluster became much more complete as we added the scheduler to the contingent of master services. The `kube-apiserver` backed by `etcd` gave us the semblance of a cluster. The `kubelet` gave us actual nodes that we could run workloads (pods) on. The `kube-scheduler` allowed us to let Kubernetes determine the best place to run the workloads.

In this lab we are going to add the final pieces of core Kubernetes functionality, support for services and the ability to scale the number of containers implementing our services. In Kubernetes the `kube-controller-manager` takes care of deploying and scaling pods and the `kube-proxy` performs the functions necessary to create services.

To begin we'll add support for deploying sets of pods to give our workloads scale and high availability. **Before starting make sure that your Scheduler, API Server and etcd are running on the master and that the kubelets are running on both nodea and nodeb**. Also make sure that your cluster has no resources in the default namespace (delete all prior pods).

1. Running deployments without a controller manager

To get a picture of our cluster's function with and without a Controller Manager, let's build a test deployment and and see what happens when we create it without a Controller Manager. Perform the following steps on nodea:

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

apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  template:
```


© Copyright 2017 RX-M LLC

```
    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:~$
```

Note that the message 'deployment "nginx-deployment" created' is from the API Server and indicates nothing other than that the API server added your desired state to the etcd store (this will fail in only the most dire circumstances).

Try listing the pods in your cluster:

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

No resources found.

user@nodea:~$
```

This is a bad sign. Why didn't the cluster create the 2 pods requested?

List the other resources that should be associated with your deployment

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

No resources found.
```


© Copyright 2017 RX-M LLC

```
user@nodea:~$
```

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

NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2         0         0            0           34s

user@nodea:~$
```

So our deployment was added to the cluster target state but nothing else was. Let's look deeper:

```
user@nodea:~$ kubectl describe deploy nginx-deployment

Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Tue, 29 Aug 2017 14:39:14 -0700
Labels:                 app=nginx
Annotations:            <none>
Selector:               app=nginx
Replicas:               2 desired | 0 updated | 0 total | 0 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:       app=nginx
  Containers:
   nginx:
    Image:              nginx:1.7.9
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
OldReplicaSets:         <none>
NewReplicaSet:          <none>
Events:                 <none>

user@nodea:~$
```


© Copyright 2017 RX-M LLC

Our deployment has no ReplicaSets. As you have probably guessed this is because the Kubernetes component that acts on deployments and creates ReplicaSets is the Controller Manager and it is not yet running.

2. Starting the controller manager

The old approach to running non-APIServer Kubernetes services was to provide them with the IRI of the API Server on the command line. The kube config approach is the go forward way to centralize cluster configuration for all of the Kubernetes services on a given node. Command line switches can still be used for many features (and are still required in some cases) but for basic operation the kubeconfig should suffice.

Run the controller manager and use the kubeconfig created earlier to point the controller manager at the appropriate API server:

```
user@nodea:~$ cat nodea.kubeconfig

apiVersion: v1
clusters:
- cluster:
    server: http://nodea:8080
  name: local
contexts:
- context:
    cluster: local
    user: ""
  name: local
current-context: local
kind: Config
preferences: {}
users: []

user@nodea:~$
```

```
user@nodea:~$ ~user/k8s/_output/bin/kube-controller-manager --kubeconfig=nodea.kubeconfig

I0829 14:41:00.467076   50370 controllermanager.go:107] Version: v1.7.4+793658f2d7ca7
I0829 14:41:00.492609   50370 leaderelection.go:179] attempting to acquire leader lease...
I0829 14:41:00.523939   50370 leaderelection.go:189] successfully acquired lease kube-system/kube-controller-
manager
I0829 14:41:00.532228   50370 event.go:218] Event(v1.ObjectReference{Kind:"Endpoints", Namespace:"kube-system",
Name:"kube-controller-manager", UID:"bfc96f31-8d02-11e7-a2b5-000c293215a2", APIVersion:"v1",
ResourceVersion:"258", FieldPath:""}): type: 'Normal' reason: 'LeaderElection' nodea became leader
I0829 14:41:00.548043   50370 plugins.go:101] No cloud provider specified.
```


© Copyright 2017 RX-M LLC

```
W0829 14:41:00.548264   50370 controllermanager.go:459] "serviceaccount-token" is disabled because there is no
private key
I0829 14:41:00.549003   50370 controllermanager.go:439] Started "deployment"
I0829 14:41:00.549333   50370 controllermanager.go:439] Started "statefulset"
E0829 14:41:00.549601   50370 certificates.go:48] Failed to start certificate controller: error reading CA cert
file "/etc/kubernetes/ca/ca.pem": open /etc/kubernetes/ca/ca.pem: no such file or directory
W0829 14:41:00.549774   50370 controllermanager.go:436] Skipping "csrsigning"
I0829 14:41:00.550181   50370 controllermanager.go:439] Started "csrapproving"
W0829 14:41:00.550331   50370 controllermanager.go:423] "tokencleaner" is disabled
E0829 14:41:00.550835   50370 core.go:66] Failed to start service controller: WARNING: no cloud provider provided,
services of type LoadBalancer will fail.
W0829 14:41:00.550988   50370 controllermanager.go:436] Skipping "service"
I0829 14:41:00.551261   50370 controllermanager.go:439] Started "podgc"
W0829 14:41:00.551407   50370 controllermanager.go:436] Skipping "cronjob"
W0829 14:41:00.551540   50370 core.go:114] Unsuccessful parsing of cluster CIDR : invalid CIDR address:
I0829 14:41:00.551740   50370 core.go:130] Will not configure cloud provider routes for allocate-node-cidrs:
false, configure-cloud-routes: true.
W0829 14:41:00.551935   50370 controllermanager.go:436] Skipping "route"
I0829 14:41:00.557571   50370 deployment_controller.go:152] Starting deployment controller
I0829 14:41:00.557854   50370 controller_utils.go:994] Waiting for caches to sync for deployment controller
I0829 14:41:00.558065   50370 stateful_set.go:147] Starting stateful set controller
I0829 14:41:00.558295   50370 controller_utils.go:994] Waiting for caches to sync for stateful set controller
I0829 14:41:00.558657   50370 controllermanager.go:439] Started "persistentvolume-binder"
I0829 14:41:00.563421   50370 gc_controller.go:76] Starting GC controller
I0829 14:41:00.563447   50370 controller_utils.go:994] Waiting for caches to sync for GC controller
I0829 14:41:00.569364   50370 pv_controller_base.go:271] Starting persistent volume controller
I0829 14:41:00.569413   50370 controller_utils.go:994] Waiting for caches to sync for persistent volume controller
I0829 14:41:00.569414   50370 certificate_controller.go:110] Starting certificate controller
I0829 14:41:00.569443   50370 controller_utils.go:994] Waiting for caches to sync for certificate controller
I0829 14:41:00.569915   50370 controllermanager.go:439] Started "endpoint"
I0829 14:41:00.570090   50370 endpoints_controller.go:136] Starting endpoint controller
I0829 14:41:00.570198   50370 controller_utils.go:994] Waiting for caches to sync for endpoint controller
I0829 14:41:00.570567   50370 controllermanager.go:439] Started "daemonset"
I0829 14:41:00.570630   50370 daemoncontroller.go:221] Starting daemon sets controller
I0829 14:41:00.570636   50370 controller_utils.go:994] Waiting for caches to sync for daemon sets controller
I0829 14:41:00.571163   50370 controllermanager.go:439] Started "horizontalpodautoscaling"
I0829 14:41:00.571284   50370 horizontal.go:145] Starting HPA controller
I0829 14:41:00.571293   50370 controller_utils.go:994] Waiting for caches to sync for HPA controller
I0829 14:41:00.571641   50370 controllermanager.go:439] Started "disruption"
I0829 14:41:00.571736   50370 disruption.go:297] Starting disruption controller
I0829 14:41:00.571747   50370 controller_utils.go:994] Waiting for caches to sync for disruption controller
I0829 14:41:00.571937   50370 controllermanager.go:439] Started "ttl"
W0829 14:41:00.571996   50370 core.go:76] Unsuccessful parsing of cluster CIDR : invalid CIDR address:
```


© Copyright 2017 RX-M LLC

```
W0829 14:41:00.572008   50370 core.go:80] Unsuccessful parsing of service CIDR : invalid CIDR address:
I0829 14:41:00.572096   50370 ttlcontroller.go:117] Starting TTL controller
I0829 14:41:00.572114   50370 controller_utils.go:994] Waiting for caches to sync for TTL controller
I0829 14:41:00.572288   50370 nodecontroller.go:224] Sending events to api server.
I0829 14:41:00.572448   50370 taint_controller.go:159] Sending events to api server.
I0829 14:41:00.579600   50370 controllermanager.go:439] Started "node"
I0829 14:41:00.580968   50370 controllermanager.go:439] Started "attachdetach"
I0829 14:41:00.581950   50370 controllermanager.go:439] Started "replicationcontroller"
I0829 14:41:00.586638   50370 nodecontroller.go:481] Starting node controller
I0829 14:41:00.586665   50370 controller_utils.go:994] Waiting for caches to sync for node controller
I0829 14:41:00.586698   50370 attach_detach_controller.go:242] Starting attach detach controller
I0829 14:41:00.586703   50370 controller_utils.go:994] Waiting for caches to sync for attach detach controller
I0829 14:41:00.589266   50370 replication_controller.go:151] Starting RC controller
I0829 14:41:00.589707   50370 controllermanager.go:439] Started "resourcequota"
I0829 14:41:00.589987   50370 controller_utils.go:994] Waiting for caches to sync for RC controller
I0829 14:41:00.591166   50370 resource_quota_controller.go:241] Starting resource quota controller
I0829 14:41:00.591342   50370 controller_utils.go:994] Waiting for caches to sync for resource quota controller
I0829 14:41:00.597928   50370 controllermanager.go:439] Started "namespace"
I0829 14:41:00.598202   50370 controllermanager.go:439] Started "serviceaccount"
I0829 14:41:00.598572   50370 controller_utils.go:994] Waiting for caches to sync for namespace controller
I0829 14:41:00.598758   50370 serviceaccounts_controller.go:113] Starting service account controller
I0829 14:41:00.598908   50370 controller_utils.go:994] Waiting for caches to sync for service account controller
E0829 14:41:00.608935   50370 graph_builder.go:204] no matches for {networking.k8s.io v1 networkpolicies}. If
{networking.k8s.io v1 networkpolicies} is a non-core resource (e.g. thirdparty resource, custom resource from
aggregated apiserver), please note that the garbage collector doesn't support non-core resources yet. Once they
are supported, object with ownerReferences referring non-existing non-core objects will be deleted by the garbage
collector.
I0829 14:41:00.609610   50370 controllermanager.go:439] Started "garbagecollector"
I0829 14:41:00.609767   50370 garbagecollector.go:123] Starting garbage collector controller
I0829 14:41:00.611719   50370 controller_utils.go:994] Waiting for caches to sync for garbage collector controller
I0829 14:41:00.615864   50370 controllermanager.go:439] Started "job"
I0829 14:41:00.616479   50370 controllermanager.go:439] Started "replicaset"
W0829 14:41:00.616737   50370 controllermanager.go:423] "bootstrapsigner" is disabled
I0829 14:41:00.619833   50370 jobcontroller.go:133] Starting job controller
I0829 14:41:00.620127   50370 controller_utils.go:994] Waiting for caches to sync for job controller
I0829 14:41:00.620330   50370 replica_set.go:156] Starting replica set controller
I0829 14:41:00.620519   50370 controller_utils.go:994] Waiting for caches to sync for replica set controller
E0829 14:41:00.665447   50370 actual_state_of_world.go:500] Failed to set statusUpdateNeeded to needed true
because nodeName="nodea"  does not exist
E0829 14:41:00.665689   50370 actual_state_of_world.go:514] Failed to update statusUpdateNeeded field in actual
state of world: Failed to set statusUpdateNeeded to needed true because nodeName="nodea"  does not exist
E0829 14:41:00.665865   50370 actual_state_of_world.go:500] Failed to set statusUpdateNeeded to needed true
because nodeName="nodeb"  does not exist
```


© Copyright 2017 RX-M LLC

```
E0829 14:41:00.666037   50370 actual_state_of_world.go:514] Failed to update statusUpdateNeeded field in actual
state of world: Failed to set statusUpdateNeeded to needed true because nodeName="nodeb"  does not exist
I0829 14:41:00.672095   50370 controller_utils.go:1001] Caches are synced for disruption controller
I0829 14:41:00.672682   50370 disruption.go:305] Sending events to api server.
I0829 14:41:00.672123   50370 controller_utils.go:1001] Caches are synced for endpoint controller
I0829 14:41:00.672135   50370 controller_utils.go:1001] Caches are synced for persistent volume controller
I0829 14:41:00.672171   50370 controller_utils.go:1001] Caches are synced for certificate controller
I0829 14:41:00.672188   50370 controller_utils.go:1001] Caches are synced for daemon sets controller
I0829 14:41:00.672195   50370 controller_utils.go:1001] Caches are synced for HPA controller
I0829 14:41:00.678106   50370 controller_utils.go:1001] Caches are synced for TTL controller
I0829 14:41:00.692786   50370 controller_utils.go:1001] Caches are synced for resource quota controller
I0829 14:41:00.692848   50370 controller_utils.go:1001] Caches are synced for node controller
I0829 14:41:00.692906   50370 controller_utils.go:1001] Caches are synced for attach detach controller
I0829 14:41:00.692987   50370 controller_utils.go:1001] Caches are synced for RC controller
I0829 14:41:00.693321   50370 nodecontroller.go:542] Initializing eviction metric for zone:
W0829 14:41:00.693416   50370 nodecontroller.go:877] Missing timestamp for Node nodea. Assuming now as a
timestamp.
W0829 14:41:00.693451   50370 nodecontroller.go:877] Missing timestamp for Node nodeb. Assuming now as a
timestamp.
I0829 14:41:00.693470   50370 nodecontroller.go:793] NodeController detected that zone  is now in state Normal.
I0829 14:41:00.693635   50370 taint_controller.go:182] Starting NoExecuteTaintManager
I0829 14:41:00.694006   50370 event.go:218] Event(v1.ObjectReference{Kind:"Node", Namespace:"", Name:"nodeb",
UID:"6df9cd68-8d02-11e7-a2b5-000c293215a2", APIVersion:"", ResourceVersion:"", FieldPath:""}): type: 'Normal'
reason: 'RegisteredNode' Node nodeb event: Registered Node nodeb in NodeController
I0829 14:41:00.694034   50370 event.go:218] Event(v1.ObjectReference{Kind:"Node", Namespace:"", Name:"nodea",
UID:"15cfb6c5-8d02-11e7-a2b5-000c293215a2", APIVersion:"", ResourceVersion:"", FieldPath:""}): type: 'Normal'
reason: 'RegisteredNode' Node nodea event: Registered Node nodea in NodeController
I0829 14:41:00.698970   50370 controller_utils.go:1001] Caches are synced for namespace controller
I0829 14:41:00.699153   50370 controller_utils.go:1001] Caches are synced for service account controller
I0829 14:41:00.712794   50370 controller_utils.go:1001] Caches are synced for garbage collector controller
I0829 14:41:00.712848   50370 garbagecollector.go:132] Garbage collector: all resource monitors have synced.
Proceeding to collect garbage
I0829 14:41:00.720536   50370 controller_utils.go:1001] Caches are synced for job controller
I0829 14:41:00.724560   50370 controller_utils.go:1001] Caches are synced for replica set controller
I0829 14:41:00.758881   50370 controller_utils.go:1001] Caches are synced for stateful set controller
I0829 14:41:00.758952   50370 controller_utils.go:1001] Caches are synced for deployment controller
I0829 14:41:00.763613   50370 controller_utils.go:1001] Caches are synced for GC controller
I0829 14:41:00.773764   50370 event.go:218] Event(v1.ObjectReference{Kind:"Deployment", Namespace:"default",
Name:"nginx-deployment", UID:"8078c3ed-8d02-11e7-a2b5-000c293215a2", APIVersion:"extensions",
ResourceVersion:"182", FieldPath:""}): type: 'Normal' reason: 'ScalingReplicaSet' Scaled up replica set nginx-
deployment-171375908 to 2
I0829 14:41:00.791907   50370 event.go:218] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default",
Name:"nginx-deployment-171375908", UID:"bfeef301-8d02-11e7-a2b5-000c293215a2", APIVersion:"extensions",
```


© Copyright 2017 RX-M LLC

```
ResourceVersion:"267", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: nginx-deployment-
171375908-4w4q2
I0829 14:41:00.794130   50370 event.go:218] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default",
Name:"nginx-deployment-171375908", UID:"bfeef301-8d02-11e7-a2b5-000c293215a2", APIVersion:"extensions",
ResourceVersion:"267", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: nginx-deployment-
171375908-xlfd7
...
```

This provides us with a lot of output.

Some key takeaways:

> I0829 14:41:00.523939 50370 leaderelection.go:189] successfully acquired lease kube-system/kube-controller-manager

Like the scheduler, there can only be one active replication controller in a cluster. Anytime a new replication controller starts it forces an election. Given that this is the first controller manager it becomes the leader and will actively begin managing Deployments, ReplicaSets and Replication Controllers.

The Controller Manager manages many other resources however. Look over the log output and identify the various resource types reported.

You should be able to find at least:

- v1
  - ReplicationController
- v1beta1
  - DaemonSet
  - Job
  - Deployment
  - ReplicaSet
  - HorizontalPodAutoscaler
  - StatefulSet

Once the Controller Manager is up and running, toward the end of the log output you will see it discover your deployment.

> I0829 14:41:00.773764 50370 event.go:218] Event(v1.ObjectReference{Kind:"Deployment", Namespace:"default", Name:"nginx-deployment", UID:"8078c3ed-8d02-11e7-a2b5-000c293215a2", APIVersion:"extensions", ResourceVersion:"182", FieldPath:""}): type: 'Normal' reason: 'ScalingReplicaSet' Scaled up replica set nginx-deployment-171375908 to 2

© Copyright 2017 RX-M LLC

This is immediately followed by events reporting the actions take by the Controller Manager to bring the cluster in line with your wishes:

> I0829 14:41:00.791907 50370 event.go:218] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"nginx-deployment-171375908", UID:"bfeef301-8d02-11e7-a2b5-000c293215a2", APIVersion:"extensions", ResourceVersion:"267", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: nginx-deployment-171375908-4w4q2 I0829 14:41:00.794130 50370 event.go:218] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default", Name:"nginx-deployment-171375908", UID:"bfeef301-8d02-11e7-a2b5-000c293215a2", APIVersion:"extensions", ResourceVersion:"267", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: nginx-deployment-171375908-xlfd7

Now try displaying the active pods:

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

NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-171375908-4w4q2   1/1       Running   0          3m
nginx-deployment-171375908-xlfd7   1/1       Running   0          3m

user@nodea:~$
```

As advertised, the Controller manager has created the two pods required. You can examine the system events to see the progression of work involved in launching your two pods:

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

LASTSEEN   FIRSTSEEN   COUNT     NAME                                KIND         SUBOBJECT                TYPE
REASON                  SOURCE                  MESSAGE
30s        7m          10        nginx-deployment-171375908-4w4q2   Pod                                   Warning
MissingClusterDNS       kubelet, nodeb          kubelet does not have ClusterDNS IP configured and cannot create
Pod using "ClusterFirst" policy. Falling back to DNSDefault policy.
7m         7m          1         nginx-deployment-171375908-4w4q2   Pod                                   Normal
Scheduled               default-scheduler       Successfully assigned nginx-deployment-171375908-4w4q2 to nodeb
7m         7m          1         nginx-deployment-171375908-4w4q2   Pod          spec.containers{nginx}   Normal
Pulling                 kubelet, nodeb          pulling image "nginx:1.7.9"
6m         6m          1         nginx-deployment-171375908-4w4q2   Pod          spec.containers{nginx}   Normal
Pulled                  kubelet, nodeb          Successfully pulled image "nginx:1.7.9"
6m         6m          1         nginx-deployment-171375908-4w4q2   Pod          spec.containers{nginx}   Normal
```


© Copyright 2017 RX-M LLC

```
Created                 kubelet, nodeb          Created container
6m          6m          1         nginx-deployment-171375908-4w4q2   Pod          spec.containers{nginx}   Normal
Started                 kubelet, nodeb          Started container
7m          7m          1         nginx-deployment-171375908-xlfd7   Pod                                   Normal
Scheduled               default-scheduler       Successfully assigned nginx-deployment-171375908-xlfd7 to nodea
38s         7m          10        nginx-deployment-171375908-xlfd7   Pod                                   Warning
MissingClusterDNS       kubelet, nodea          kubelet does not have ClusterDNS IP configured and cannot create
Pod using "ClusterFirst" policy. Falling back to DNSDefault policy.
7m          7m          1         nginx-deployment-171375908-xlfd7   Pod          spec.containers{nginx}   Normal
Pulled                  kubelet, nodea          Container image "nginx:1.7.9" already present on machine
7m          7m          1         nginx-deployment-171375908-xlfd7   Pod          spec.containers{nginx}   Normal
Created                 kubelet, nodea          Created container
7m          7m          1         nginx-deployment-171375908-xlfd7   Pod          spec.containers{nginx}   Normal
Started                 kubelet, nodea          Started container
7m          7m          1         nginx-deployment-171375908         ReplicaSet                            Normal
SuccessfulCreate        replicaset-controller   Created pod: nginx-deployment-171375908-4w4q2
7m          7m          1         nginx-deployment-171375908         ReplicaSet                            Normal
SuccessfulCreate        replicaset-controller   Created pod: nginx-deployment-171375908-xlfd7
7m          7m          1         nginx-deployment                   Deployment                            Normal
ScalingReplicaSet       deployment-controller   Scaled up replica set nginx-deployment-171375908 to 2
11m         11m         1         nodea                              Node                                  Normal
Starting                kubelet, nodea          Starting kubelet.
11m         11m         2         nodea                              Node                                  Normal
NodeHasSufficientDisk   kubelet, nodea          Node nodea status is now: NodeHasSufficientDisk
11m         11m         2         nodea                              Node                                  Normal
NodeHasSufficientMemory kubelet, nodea          Node nodea status is now: NodeHasSufficientMemory
11m         11m         2         nodea                              Node                                  Normal
NodeHasNoDiskPressure   kubelet, nodea          Node nodea status is now: NodeHasNoDiskPressure
11m         11m         1         nodea                              Node                                  Normal
nodeallocatableEnforced kubelet, nodea          Updated Node Allocatable limit across pods
11m         11m         1         nodea                              Node                                  Normal
NodeReady               kubelet, nodea          Node nodea status is now: NodeReady
7m          7m          1         nodea                              Node                                  Normal
RegisteredNode          controllermanager       Node nodea event: Registered Node nodea in NodeController
38s         7m          10        nodea                              Node                                  Warning
MissingClusterDNS       kubelet, nodea          kubelet does not have ClusterDNS IP configured and cannot create
Pod using "ClusterFirst" policy. pod: "nginx-deployment-171375908-xlfd7_default(bff286b8-8d02-11e7-a2b5-
000c293215a2)". Falling back to DNSDefault policy.
9m          9m          1         nodeb                              Node                                  Normal
Starting                kubelet, nodeb          Starting kubelet.
9m          9m          2         nodeb                              Node                                  Normal
NodeHasSufficientDisk   kubelet, nodeb          Node nodeb status is now: NodeHasSufficientDisk
9m          9m          2         nodeb                              Node                                  Normal
```


© Copyright 2017 RX-M LLC

```
NodeHasSufficientMemory   kubelet, nodeb         Node nodeb status is now: NodeHasSufficientMemory
9m         9m          2         nodeb                                Node                                    Normal
NodeHasNoDiskPressure     kubelet, nodeb         Node nodeb status is now: NodeHasNoDiskPressure
9m         9m          1         nodeb                                Node                                    Normal
nodeallocatableEnforced   kubelet, nodeb         Updated Node Allocatable limit across pods
9m         9m          1         nodeb                                Node                                    Normal
NodeReady                 kubelet, nodeb         Node nodeb status is now: NodeReady
7m         7m          1         nodeb                                Node                                    Normal
RegisteredNode            controllermanager      Node nodeb event: Registered Node nodeb in NodeController
30s        7m          10        nodeb                                Node                                    Warning
MissingClusterDNS         kubelet, nodeb         kubelet does not have ClusterDNS IP configured and cannot create
Pod using "ClusterFirst" policy. pod: "nginx-deployment-171375908-4w4q2_default(bff2876c-8d02-11e7-a2b5-
000c293215a2)". Falling back to DNSDefault policy.

user@nodea:~$
```

After creating the Deployment's ReplicaSet the Controller Manager creates two Pods from the template and submits them to the cluster. The Scheduler is exclusively responsible for scheduling Pods to Kubelets when the Pods are not pinned to a particular host. The event listing "SOURCE" column shows the service responsible for the event message.

As soon as the Pods are scheduled the Kubelets begin pulling images and launching Pods. When the Pods are up the ReplicaSet reports successful Pod creation and when all Pods are up the Deployment reports successful scaling (from 0 to 2).

Note that all Pods are automatically registered in the Kubernetes Cluster DNS if one is configured. We see Warnings because we have not yet setup Cluster DNS. We will take care of that in a future lab.

Relist your running Pods, ReplicaSets, and Deployments:

```
user@nodea:~$ kubectl get deployments,replicasets,pods

NAME                     DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deploy/nginx-deployment   2         2         2            2           9m

NAME                               DESIRED   CURRENT   READY     AGE
rs/nginx-deployment-171375908   2         2         2         7m

NAME                                   READY     STATUS    RESTARTS   AGE
po/nginx-deployment-171375908-4w4q2   1/1       Running   0          7m
po/nginx-deployment-171375908-xlfd7   1/1       Running   0          7m
user@nodea:~$
```


© Copyright 2017 RX-M LLC

Everything looks healthy.

3. Services and networking

What if we want to retrieve some web pages from one of the Pods? Some questions:

- Do we care which one we get the pages from?
- Do we want to be wired to a single Pod, what if it crashes?

The answers to these questions are typically "No" and "No". Deployments create ReplicaSets and ReplicaSets create replicas. The reason we have replicas is for scale and HA (i.e. to ensure that failure of one replica does not cause failure of the whole). In essence we want access to the "service" without being tied to the Pod that implements it.

In Kubernetes, "Services" provide a layer of abstraction on top of a set of Pod replicas implementing the service. Services identify the pods that implement them using a label selector. Identify the labels assigned to your Pods:

```
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 get pods
```


© Copyright 2017 RX-M LLC

```
NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-171375908-4w4q2   1/1       Running   0          11m
nginx-deployment-171375908-xlfd7   1/1       Running   0          11m

user@nodea:~$
```

```
user@nodea:~$ kubectl describe pod nginx-deployment-171375908-4w4q2 | grep -iA1 labels

Labels:         app=nginx
                pod-template-hash=171375908

user@nodea:~$
```

Our template defines the label "app=nginx". Note that the pod also contains a template hash. This allows you to identify the template that was used to create the pod and to detect pods that are not implementing the current template.

Create a Kubernetes Service that selects the two Pods to back a service called "nsvc":

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

apiVersion: v1
kind: Service
metadata:
  name: nsvc
spec:
  ports:
    - port: 2000
      targetPort: 80
  selector:
    app: nginx

user@nodea:~$
```

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

service "nsvc" created
```


© Copyright 2017 RX-M LLC

```
user@nodea:~$
```

Verify the creation of the service:

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

NAME         CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
kubernetes   10.0.0.1     <none>        443/TCP    18m
nsvc         10.0.0.253   <none>        2000/TCP   5s

user@nodea:~$
```

```
user@nodea:~$ kubectl describe service nsvc

Name:                   nsvc
Namespace:              default
Labels:                 <none>
Annotations:            <none>
Selector:               app=nginx
Type:                   ClusterIP
IP:                     10.0.0.253
Port:                   <unset> 2000/TCP
Endpoints:              172.17.0.2:80,172.17.0.2:80
Session Affinity:       None
Events:                 <none>

user@nodea:~$
```

The API Server has created our service and given it an IP (in the example above) of 10.0.0.253 and a port of 2000 (as we requested in the spec).

You may have noticed in earlier labs the kube-apiserver flag `--service-cluster-ip-range=10.0.0.0/16`. This range is the pool of IPs that the ClusterIP pulls from for service IPs (10.0.0.253 in our example.) Often, this IP is called the VIP (virtual IP), ClusterIP, or just IP. This range must not overlap with your nodes subnet (172.16.151.0/24) or your container network(S) (172.17.0.0/16).

Try curling this end point:

© Copyright 2017 RX-M LLC

```
user@nodea:~$ curl -I 10.0.0.253:2000

curl: (7) Failed to connect to 10.0.0.253 port 2000: Connection refused

user@nodea:~$
```

No luck. This is a Virtual IP (VIP). Virtual IPs are, well, virtual. They are not connected with real listening endpoints, rather they are hardware/software table entries that redirect traffic somewhere else. In Kubernetes the process responsible for creating the rules on every node to redirect VIP traffic is the Kube Proxy and we have not started it yet.

The service description also reports endpoints associated with each of the Pods running the service.

Try curling one of them:

```
user@nodea:~$ curl -I 172.17.0.2:80

HTTP/1.1 200 OK
Server: nginx/1.7.9
Date: Tue, 29 Aug 2017 21:57:03 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 23 Dec 2014 16:25:09 GMT
Connection: keep-alive
ETag: "54999765-264"
Accept-Ranges: bytes

user@nodea:~$
```

This works but only if you try it on the machine that the Pod is running on. Why? Because our two nodes are using default Docker installations and all current Docker installations create containers on the docker0 bridge and the docker0 bridge has the subnet 172.17.0.0/16 by default, *on every node*! You may, for example see your two service pods having the exact same IP address!

Examine the docker0 network on your two nodes:

nodea:

```
user@nodea:~$ ip a show dev docker0
```


© Copyright 2017 RX-M LLC

```
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:10:a3:5a:ce brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:10ff:fea3:5ace/64 scope link
       valid_lft forever preferred_lft forever
user@nodea:~$
```

nodeb:

```
user@nodeb:~$ ip a show dev docker0

3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:2e:20:db:74 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:2eff:fe20:db74/64 scope link
       valid_lft forever preferred_lft forever
user@nodeb:~$
```

There are many many ways to configure networking in a Kubernetes cluster. The simplest way is to statically configure each node in the cluster with a unique docker0 subnet and then to set routes in every node to all of the other node docker0 subnets. This will ensure that each node assigns unique IPs to its Pods and that all Pods can reach each other directly via the static routes.

Docker automatically configures a route on the Docker host to the docker0 bridge by default. Display the route table on nodea for example:

```
user@nodea:~$ ip route

default via 172.16.151.2 dev ens33
172.16.151.0/24 dev ens33  proto kernel  scope link  src 172.16.151.203
172.17.0.0/16 dev docker0  proto kernel  scope link  src 172.17.0.1

user@nodea:~$
```

All 172.17/16 traffic will be placed on the docker0 Linux Bridge (which acts like an L2 switch).

© Copyright 2017 RX-M LLC

4. Configuring a flat network on nodeb

While static network configuration is straightforward and does not involve SDN, tunnels, or other slow downs, it is static. This means that it requires a static infrastructure to be reliable and changes require work. We'll try SDN in a later lab, for now we'll configure static routes and docker0 bridges with non-overlapping subnets.

To begin delete all of the resources on your cluster.

Deployments:

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

NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2         2         2            2           19m

user@nodea:~$
```

```
user@nodea:~$ kubectl delete deployment nginx-deployment

deployment "nginx-deployment" deleted

user@nodea:~$
```

Services:

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

NAME         CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
kubernetes   10.0.0.1     <none>        443/TCP    24m
nsvc         10.0.0.253   <none>        2000/TCP   6m

user@nodea:~$
```

The kubernetes service is created by the API server and is the VIP for the API server, allowing any pod in the cluster to easily lookup and call the API Server. Delete the nsvc service you created but **do not** delete the kubernetes service.

© Copyright 2017 RX-M LLC

```
user@nodea:~$ kubectl delete service nsvc

service "nsvc" deleted

user@nodea:~$
```

Verify that all pods are terminated:

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

No resources found.

user@nodea:~$
```

```
user@nodea:~$ docker container ls -a

CONTAINER ID        IMAGE        COMMAND          CREATED             STATUS              PORTS
NAMES

user@nodea:~$
```

Change to nodeb and verify that no containers are running under Docker:

```
user@nodeb:~$ docker container ls -a

CONTAINER ID        IMAGE         COMMAND          CREATED          STATUS          PORTS           NAMES

user@nodeb:~$
```

Now we can change nodeb's docker0 subnet.

On Ubuntu 16.04 Docker runs as a systemd service. We can augment the service configuration by editing the docker service file. Locate the systemd Docker service file (press 'q' to exit the log listing):

© Copyright 2017 RX-M LLC

```
user@nodeb:~$ sudo systemctl status docker.service

● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2017-08-29 12:44:49 PDT; 2h 16min ago
     Docs: https://docs.docker.com
 Main PID: 1350 (dockerd)
    Tasks: 45
   Memory: 301.5M
      CPU: 43.871s
   CGroup: /system.slice/docker.service
           ├─1350 /usr/bin/dockerd -H fd://
           └─1457 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-
interval=0 --start-tim

Aug 29 12:44:48 nodeb dockerd[1350]: time="2017-08-29T12:44:48.879828385-07:00" level=warning msg="Your kernel
does not support swa
Aug 29 12:44:48 nodeb dockerd[1350]: time="2017-08-29T12:44:48.879869998-07:00" level=warning msg="Your kernel
does not support cgr
Aug 29 12:44:48 nodeb dockerd[1350]: time="2017-08-29T12:44:48.879879585-07:00" level=warning msg="Your kernel
does not support cgr
Aug 29 12:44:48 nodeb dockerd[1350]: time="2017-08-29T12:44:48.881942983-07:00" level=info msg="Loading
containers: start."
Aug 29 12:44:49 nodeb dockerd[1350]: time="2017-08-29T12:44:49.145285110-07:00" level=info msg="Default bridge
(docker0) is assigne
Aug 29 12:44:49 nodeb dockerd[1350]: time="2017-08-29T12:44:49.193662606-07:00" level=info msg="Loading
containers: done."
Aug 29 12:44:49 nodeb dockerd[1350]: time="2017-08-29T12:44:49.247726194-07:00" level=info msg="Daemon has
completed initialization
Aug 29 12:44:49 nodeb dockerd[1350]: time="2017-08-29T12:44:49.247784698-07:00" level=info msg="Docker daemon"
commit=874a737 graph
Aug 29 12:44:49 nodeb dockerd[1350]: time="2017-08-29T12:44:49.256536487-07:00" level=info msg="API listen on
/var/run/docker.sock"
Aug 29 12:44:49 nodeb systemd[1]: Started Docker Application Container Engine.
user@nodeb:~$
```

Now inspect the Docker service file:

```
user@nodeb:~$ cat /lib/systemd/system/docker.service
```


© Copyright 2017 RX-M LLC

```
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network-online.target docker.socket firewalld.service
Wants=network-online.target
Requires=docker.socket

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd://
ExecReload=/bin/kill -s HUP $MAINPID
LimitNOFILE=1048576
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNPROC=infinity
LimitCORE=infinity
# Uncomment TasksMax if your systemd version supports it.
# Only systemd 226 and above support this version.
TasksMax=infinity
TimeoutStartSec=0
# set delegate yes so that systemd does not reset the cgroups of docker containers
Delegate=yes
# kill only the docker process, not all processes in the cgroup
KillMode=process
# restart the docker process if it exits prematurely
Restart=on-failure
StartLimitBurst=3
StartLimitInterval=60s

[Install]
WantedBy=multi-user.target
user@nodeb:~$
```

We will make changes to this file in subsequent steps.

4.a. Assign docker0 a unique subnet

Now add the `--bip` switch to the Docker start string (*ExecStart*) with a value that will cause the docker0 bridge to use subnet 172.18/16. First stop the Kubelet

© Copyright 2017 RX-M LLC

(with control+c) and Docker (using systemctl) on **nodeb**:

1. Press control+c in the kubelet terminal to exit the kubelet
2. Shutdown docker

```
user@nodeb:~$ sudo systemctl stop docker

user@nodeb:~$
```

Next, remove the old bridge IP address (it has the old, now incorrect subnet):

```
user@nodeb:~$ ip a

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:68:cd:9d brd ff:ff:ff:ff:ff:ff
    inet 172.16.151.204/24 brd 172.16.151.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe68:cd9d/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:2e:20:db:74 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:2eff:fe20:db74/64 scope link
       valid_lft forever preferred_lft forever

user@nodeb:~$
```

```
user@nodeb:~$ sudo ip addr del 172.17.0.1/16 dev docker0
```


© Copyright 2017 RX-M LLC

```
user@nodeb:~$
```

Now update the Docker startup command so that docker assigns the docker0 bridge the 172.18 subnet with the bridge address of 0.1:

```
user@nodeb:~$ sudo vim  /lib/systemd/system/docker.service
user@nodeb:~$ cat /lib/systemd/system/docker.service | grep ExecStart

ExecStart=/usr/bin/dockerd -H fd://  --bip=172.18.0.1/16

user@nodeb:~$
```

Reload the systemd config and restart docker:

```
user@nodeb:~$ sudo systemctl daemon-reload

user@nodeb:~$
```

```
user@nodeb:~$ sudo systemctl start docker

user@nodeb:~$
```

Verify the configuration:

```
user@nodeb:~$ ip a show dev docker0

3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:2e:20:db:74 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:2eff:fe20:db74/64 scope link
       valid_lft forever preferred_lft forever

user@nodeb:~$
```


© Copyright 2017 RX-M LLC

Perfect.

Now nodea has subnet 172.17 under its control and nodeb has subnet 172.18 under its control.

We have more work to do however. Test run an nginx container on nodeb:

```
user@nodeb:~$ docker container run -d nginx

5a8d5ee42d140010871186e287786e600f101b4f03f1c2550d9eacb708937817

user@nodeb:~$
```

Now try to curl the container on port 80:

```
user@nodeb:~$ docker container inspect $(docker container ls \
--filter=ancestor=nginx -q) | jq .[].NetworkSettings.Networks.bridge.IPAddress -r

172.18.0.2

user@nodeb:~$
```

```
user@nodeb:~$ curl -I 172.18.0.2

HTTP/1.1 200 OK
Server: nginx/1.13.3
Date: Tue, 29 Aug 2017 22:05:18 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 11 Jul 2017 13:06:07 GMT
Connection: keep-alive
ETag: "5964cd3f-264"
Accept-Ranges: bytes

user@nodeb:~$
```

Perfect, we can reach the container. This works because the host has a route to 172.18 (Docker creates it automatically):

© Copyright 2017 RX-M LLC

```
user@nodeb:~$ ip route

default via 172.16.151.2 dev ens33
172.16.151.0/24 dev ens33  proto kernel  scope link  src 172.16.151.204
172.18.0.0/16 dev docker0  proto kernel  scope link  src 172.18.0.1

user@nodeb:~$
```

Now change machines to nodea and retry the curl experiment:

```
user@nodea:~$ curl 172.18.0.2

curl: (7) Failed to connect to 172.18.0.2 port 80: Connection refused

user@nodea:~$
```

> N.B. your terminal may hang for several moments before returning with the above message.

What is wrong?

nodea, of course, has no way to know where this new 172.18 subnet is. What we need is a route on nodea that forwards all 172.18 traffic to nodeb. nodeb already has a route to its docker0 bridge for all 172.18 traffic (as we have seen) and so it will forward the traffic to docker0 completing the route.

4.b. Create a route on nodea to docker0 on nodeb

First identify the external IP of nodeb, this is where we will need to route 172.18 traffic to from nodea:

```
user@nodeb:~$ ip a show ens33

2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:e8:e4:46 brd ff:ff:ff:ff:ff:ff
    inet 172.16.151.204/24 brd 192.168.225.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fee8:e446/64 scope link
       valid_lft forever preferred_lft forever
```


© Copyright 2017 RX-M LLC

```
user@nodeb:~$
```

Now add the route on nodea (**be sure to substitute the correct external IP for your nodeb system**):

```
user@nodea:~$ sudo ip route add 172.18.0.0/16 via 172.16.151.204

user@nodea:~$
```

```
user@nodea:~$ ip route

default via 172.16.151.2 dev ens33
172.16.151.0/24 dev ens33  proto kernel  scope link  src 172.16.151.203
172.17.0.0/16 dev docker0  proto kernel  scope link  src 172.17.0.1 linkdown
172.18.0.0/16 via 172.16.151.204 dev ens33

user@nodea:~$
```

Perfect, now try to curl the nginx container from nodea:

```
user@nodea:~$ curl -I 172.18.0.2

curl: (7) Failed to connect to 172.18.0.2 port 80: Connection timed out

user@nodea:~$
```

No luck. Let's see if nodeb is reachable:

```
user@nodea:~$ ping -c 1 nodeb

PING nodeb (172.16.151.204) 56(84) bytes of data.
64 bytes from nodeb (172.16.151.204): icmp_seq=1 ttl=64 time=0.486 ms

--- nodeb ping statistics ---
```


© Copyright 2017 RX-M LLC

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.486/0.486/0.486/0.000 ms

user@nodea:~$
```

So we can reach nodeb.

Now lets try to reach docker0 on nodeb:

```
user@nodea:~$ ping -c 1 172.18.0.1

PING 172.18.0.1 (172.18.0.1) 56(84) bytes of data.
64 bytes from 172.18.0.1: icmp_seq=1 ttl=64 time=0.484 ms

--- 172.18.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.484/0.484/0.484/0.000 ms

user@nodea:~$
```

Also good!

Let's try to reach the nginx container:

```
user@nodea:~$ ping -c 1 172.18.0.2

PING 172.18.0.2 (172.18.0.2) 56(84) bytes of data.

--- 172.18.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

user@nodea:~$
```

No good. What could stop our packets from getting forwarded?

Change to a terminal on nodeb and display the IP Filter table:

© Copyright 2017 RX-M LLC

```
user@nodeb:~$ sudo iptables -L -vn -t filter

Chain INPUT (policy ACCEPT 388 packets, 195K bytes)
 pkts bytes target     prot opt in     out     source               destination
31755   91M KUBE-FIREWALL  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP 5 packets, 324 bytes)
 pkts bytes target     prot opt in     out     source               destination
    5   324 DOCKER-USER  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    5   324 DOCKER-ISOLATION  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0            ctstate
RELATED,ESTABLISHED
    5   324 DOCKER     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  docker0 docker0  0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy ACCEPT 374 packets, 60038 bytes)
 pkts bytes target     prot opt in     out     source               destination
28000 2713K KUBE-FIREWALL  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER (1 references)
 pkts bytes target     prot opt in     out     source               destination

Chain DOCKER-ISOLATION (1 references)
 pkts bytes target     prot opt in     out     source               destination
    5   324 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER-USER (1 references)
 pkts bytes target     prot opt in     out     source               destination
    5   324 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain KUBE-FIREWALL (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes firewall
for dropping marked packets */ mark match 0x8000/0x8000
user@nodeb:~$
```

A ha! The FORWARD chain is dropping packets. The default FORWARD policy is "DROP" and in the example above, 5 packets have been dropped. This is because there is no rule to ACCEPT packets headed to 172.18, so if the packet has to be FORWARDED it will be DROPped instead.

4.c. Create a rule in the filter table FORWARD chain that allows traffic to docker0

© Copyright 2017 RX-M LLC

Let's add a rule that ACCEPTs traffic from ens33 headed to 172.18:

```
user@nodeb:~$ sudo iptables -A FORWARD -i ens33 -d 172.18.0.0/16 -j ACCEPT

user@nodeb:~$
```

```
user@nodeb:~$ sudo iptables -L -vn -t filter

Chain INPUT (policy ACCEPT 10 packets, 5751 bytes)
 pkts bytes target     prot opt in     out     source               destination
31860   91M KUBE-FIREWALL  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    5   324 DOCKER-USER  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    5   324 DOCKER-ISOLATION  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0            ctstate
RELATED,ESTABLISHED
    5   324 DOCKER     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  docker0 docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  ens33  *       0.0.0.0/0            172.18.0.0/16

Chain OUTPUT (policy ACCEPT 9 packets, 1757 bytes)
 pkts bytes target     prot opt in     out     source               destination
28111 2730K KUBE-FIREWALL  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER (1 references)
 pkts bytes target     prot opt in     out     source               destination

Chain DOCKER-ISOLATION (1 references)
 pkts bytes target     prot opt in     out     source               destination
    5   324 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER-USER (1 references)
 pkts bytes target     prot opt in     out     source               destination
    5   324 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain KUBE-FIREWALL (2 references)
 pkts bytes target     prot opt in     out     source               destination
```


© Copyright 2017 RX-M LLC

```
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes firewall
for dropping marked packets */ mark match 0x8000/0x8000
user@nodeb:~$
```

Looks good. Now return to nodea and retry your ping and curl of the nginx container on nodeb:

```
user@nodea:~$ ping -c 1 172.18.0.2

PING 172.18.0.2 (172.18.0.2) 56(84) bytes of data.
64 bytes from 172.18.0.2: icmp_seq=1 ttl=63 time=0.547 ms

--- 172.18.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.547/0.547/0.547/0.000 ms

user@nodea:~$
```

```
user@nodea:~$ curl -I 172.18.0.2

HTTP/1.1 200 OK
Server: nginx/1.13.3
Date: Tue, 29 Aug 2017 22:11:30 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 11 Jul 2017 13:06:07 GMT
Connection: keep-alive
ETag: "5964cd3f-264"
Accept-Ranges: bytes

user@nodea:~$
```

Magic!

Return to nodeb and redisplay the filter table:

```
user@nodeb:~$ sudo iptables -L -vn -t filter
```


© Copyright 2017 RX-M LLC

```
Chain INPUT (policy ACCEPT 51 packets, 43772 bytes)
 pkts bytes target     prot opt in     out     source               destination
31901   91M KUBE-FIREWALL  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
   17  1341 DOCKER-USER  all  --  *      *       0.0.0.0/0            0.0.0.0/0
   17  1341 DOCKER-ISOLATION  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    5   335 ACCEPT     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0            ctstate
RELATED,ESTABLISHED
    7   468 DOCKER     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
    5   538 ACCEPT     all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  docker0 docker0  0.0.0.0/0            0.0.0.0/0
    2   144 ACCEPT     all  --  ens33  *       0.0.0.0/0            172.18.0.0/16

Chain OUTPUT (policy ACCEPT 61 packets, 13588 bytes)
 pkts bytes target     prot opt in     out     source               destination
28163 2741K KUBE-FIREWALL  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER (1 references)
 pkts bytes target     prot opt in     out     source               destination

Chain DOCKER-ISOLATION (1 references)
 pkts bytes target     prot opt in     out     source               destination
   17  1341 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER-USER (1 references)
 pkts bytes target     prot opt in     out     source               destination
   17  1341 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain KUBE-FIREWALL (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes firewall
for dropping marked packets */ mark match 0x8000/0x8000

user@nodeb:~$
```

Notice that our new rule has ACCEPTed packets (shown in the pkts column) allowing connectivity from nodea to docker0 on nodeb.

5. Configuring a flat network on nodea

© Copyright 2017 RX-M LLC

We are only 1/2 way done. While we have configured things so that nodea can reach nodeb, we need to make the same changes in reverse so that nodeb can reach containers on nodea.

5.a. Assign docker0 a unique subnet

On nodea display the subnet for docker0:

```
user@nodea:~$ ip a show dev docker0

3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:10:a3:5a:ce brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:10ff:fea3:5ace/64 scope link
       valid_lft forever preferred_lft forever

user@nodea:~$
```

The nodea docker0 bridge is using 172.17. No one else is using this so we can leave it as is.

5.b. Create a route on nodeb to docker0 on nodea

The docker0 bridge on nodea uses the 172.17 subnet so we need to create a route to this subnet on nodeb (**be sure to substitute the correct external IP for your nodea system**):

```
user@nodeb:~$ ip route

default via 172.16.151.2 dev ens33
172.16.151.0/24 dev ens33  proto kernel  scope link  src 172.16.151.204
172.18.0.0/16 dev docker0  proto kernel  scope link  src 172.18.0.1

user@nodeb:~$
```

```
user@nodeb:~$ sudo ip route add 172.17.0.0/16 via 172.16.151.203

user@nodeb:~$
```


© Copyright 2017 RX-M LLC

```
user@nodeb:~$ ip route

default via 172.16.151.2 dev ens33
172.16.151.0/24 dev ens33  proto kernel  scope link  src 172.16.151.204
172.17.0.0/16 via 172.16.151.203 dev ens33
172.18.0.0/16 dev docker0  proto kernel  scope link  src 172.18.0.1

user@nodeb:~$
```

5.c. Create a rule in the filter table FORWARD chain that allows traffic to docker0 on nodea

Finally, add the iptables rule on nodea that allows inbound traffic to 172.17:

```
user@nodea:~$ sudo iptables -A FORWARD -i ens33 -d 172.17.0.0/16 -j ACCEPT

user@nodea:~$
```

```
user@nodea:~$ sudo iptables -L -vn -t filter

Chain INPUT (policy ACCEPT 165 packets, 50749 bytes)
 pkts bytes target     prot opt in     out     source               destination
 886K  258M KUBE-FIREWALL  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DOCKER-USER  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    0     0 DOCKER-ISOLATION  all  --  *      *       0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0           ctstate
RELATED,ESTABLISHED
    0     0 DOCKER     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  docker0 docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  ens33  *       0.0.0.0/0            172.17.0.0/16

Chain OUTPUT (policy ACCEPT 162 packets, 51001 bytes)
```


© Copyright 2017 RX-M LLC

```
 pkts bytes target     prot opt in     out     source               destination
1458K 2666M KUBE-FIREWALL  all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER (1 references)
 pkts bytes target     prot opt in     out     source               destination

Chain DOCKER-ISOLATION (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER-USER (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain KUBE-FIREWALL (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes firewall
for dropping marked packets */ mark match 0x8000/0x8000

user@nodea:~$
```

5.d. Test the configuration

Now lets test our network setup by running a second container on nodea and then see if we can curl the nginx container on nodeb.

On **nodea** run a busybox container, then ping the IP and retrieve the nginx root doc:

```
user@nodea:~$ docker container run -it busybox

/ # ping -c 1 172.18.0.2

PING 172.18.0.2 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=62 time=0.699 ms

--- 172.18.0.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.699/0.699/0.699 ms

/ # wget -qO - 172.18.0.2
```


© Copyright 2017 RX-M LLC

```
Connecting to 172.18.0.2 (172.18.0.2:80)
<!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:~$
```

Perfect. We have setup cluster networking the hard way!

6. Services revisited

Now that we have Pod to Pod networking functioning in our cluster we can return to our initial goal. Setting up and running Kubernetes services.

To begin, terminate all containers running under Docker on both nodes:

nodea:

© Copyright 2017 RX-M LLC

```
user@nodea:~$ docker container rm $(docker container stop $(docker container ls -qa))

...
user@nodea:~$
```

nodeb:

```
user@nodeb:~$ docker container rm $(docker container stop $(docker container ls -qa))

...
user@nodeb:~$
```

Verify that the API server, etcd, kubelet, controller manager, and scheduler are all running on nodea (if not restart the missing services):

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

COMMAND
bash
 \_ ps
bash
 \_ kube-controller
bash
 \_ kube-scheduler
bash
 \_ sudo
     \_ kubelet
         \_ journalctl
bash
 \_ sudo
     \_ kube-apiserver
bash
 \_ etcd
Xorg
agetty

user@nodea:~$
```


© Copyright 2017 RX-M LLC

On nodeb, restart the kubelet:

```
user@nodeb:~$ sudo ./kube-bin/kubelet --kubeconfig=nodeb.kubeconfig --require-kubeconfig --allow-privileged=true

...
I0829 15:16:07.096653    7034 kubelet_node_status.go:82] Attempting to register node nodeb
I0829 15:16:07.114454    7034 kubelet_node_status.go:133] Node nodeb was previously registered
I0829 15:16:07.114520    7034 kubelet_node_status.go:85] Successfully registered node nodeb
...
```

Now lets recreate our original deployment on nodea:

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

deployment "nginx-deployment" created

user@nodea:~$
```

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

NAME                     DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deploy/nginx-deployment   2         2         2            2           9s

NAME                               DESIRED   CURRENT   READY     AGE
rs/nginx-deployment-171375908   2         2         2         9s

NAME                                  READY     STATUS    RESTARTS   AGE
po/nginx-deployment-171375908-czqm9   1/1       Running   0          9s
po/nginx-deployment-171375908-gq5lz   1/1       Running   0          9s

user@nodea:~$
```

Next recreate the service for the deployment:

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


© Copyright 2017 RX-M LLC

```
service "nsvc" created

user@nodea:~$
```

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

NAME         CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
kubernetes   10.0.0.1      <none>        443/TCP    43m
nsvc         10.0.44.224   <none>        2000/TCP   15s

user@nodea:~$
```

```
user@nodea:~$ kubectl describe service nsvc

Name:                   nsvc
Namespace:              default
Labels:                 <none>
Annotations:            <none>
Selector:               app=nginx
Type:                   ClusterIP
IP:                     10.0.44.224
Port:                   <unset> 2000/TCP
Endpoints:              172.17.0.2:80,172.18.0.2:80
Session Affinity:       None
Events:                 <none>

user@nodea:~$
```

Try pinging each pod.

```
user@nodea:~$ ping -c 1 172.17.0.2

PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.097 ms
```


© Copyright 2017 RX-M LLC

```
--- 172.17.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.097/0.097/0.097/0.000 ms

user@nodea:~$
```

```
user@nodea:~$ ping -c 1 172.18.0.2

PING 172.18.0.2 (172.18.0.2) 56(84) bytes of data.
64 bytes from 172.18.0.2: icmp_seq=1 ttl=63 time=0.488 ms

--- 172.18.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.488/0.488/0.488/0.000 ms

user@nodea:~$
```

Now try to ping the pods from nodeb:

```
user@nodeb:~$ ping -c 1 172.17.0.2

PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=63 time=0.416 ms

--- 172.17.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.416/0.416/0.416/0.000 ms

user@nodeb:~$
```

```
user@nodeb:~$ ping -c 1 172.18.0.2

PING 172.18.0.2 (172.18.0.2) 56(84) bytes of data.
64 bytes from 172.18.0.2: icmp_seq=1 ttl=64 time=0.045 ms

--- 172.18.0.2 ping statistics ---
```


© Copyright 2017 RX-M LLC

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.045/0.045/0.045/0.000 ms

user@nodeb:~$
```

Excellent, we can reach both pods from anywhere in the cluster.

Now let's try to reach the pod using the nginx port (80):

```
user@nodea:~$ curl -I 172.17.0.2

HTTP/1.1 200 OK
Server: nginx/1.7.9
Date: Tue, 29 Aug 2017 22:19:22 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 23 Dec 2014 16:25:09 GMT
Connection: keep-alive
ETag: "54999765-264"
Accept-Ranges: bytes

user@nodea:~$
```

Good! Now try the Service VIP and port:

```
user@nodea:~$ curl 10.0.44.224:2000

curl: (7) Failed to connect to 10.0.44.224 port 2000: Connection refused

user@nodea:~$
```

What now!? We are missing the final piece of the service equation, kube-proxy. Remember, the API server simply records your wishes in etcd. It is up to other Kubernetes components to perform operations on the cluster that make those wishes real. In the case of services, it is the kube-proxy, which must run on every node, that creates the forwarding rules that bring service VIPs and Ports to life.

Display the NAT table rules on nodea:

© Copyright 2017 RX-M LLC

```
user@nodea:~$ sudo iptables -L -vn -t nat

Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
  507 30496 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0            ADDRTYPE match dst-type
LOCAL

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
  129  7740 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type
LOCAL

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
  936 70031 KUBE-POSTROUTING  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes
postrouting rules */
    2   144 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0

Chain KUBE-MARK-DROP (0 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MARK       all  --  *      *       0.0.0.0/0            0.0.0.0/0            MARK or 0x8000

Chain KUBE-MARK-MASQ (0 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MARK       all  --  *      *       0.0.0.0/0            0.0.0.0/0            MARK or 0x4000

Chain KUBE-POSTROUTING (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MASQUERADE  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes service
traffic requiring SNAT */ mark match 0x4000/0x4000

user@nodea:~$
```

Note that there are no rules referencing our virtual IP, 10.0.44.224.

© Copyright 2017 RX-M LLC

As of Kubernetes 1.7, the kube-proxy component has been converted to use a configuration file. The old flags still work in 1.7, but they are being deprecated and will be removed in a future release. The `--write-config-to` flag has been provided to allow users to write the default kube-proxy configuration settings to a file.

Now **in a new tab or terminal** run kube-proxy on nodea, writing the default config to the file kube-dns-config:

```
user@nodea:~$ sudo ~user/k8s/_output/bin/kube-proxy --write-config-to=kube-proxy-config

Wrote configuration to: kube-proxy-config

user@nodea:~$
```

```
user@nodea:~$ cat kube-proxy-config

apiVersion: componentconfig/v1alpha1
bindAddress: 0.0.0.0
clientConnection:
  acceptContentTypes: ""
  burst: 10
  contentType: application/vnd.kubernetes.protobuf
  kubeconfig: ""
  qps: 5
clusterCIDR: ""
configSyncPeriod: 15m0s
conntrack:
  max: 0
  maxPerCore: 32768
  min: 131072
  tcpCloseWaitTimeout: 1h0m0s
  tcpEstablishedTimeout: 24h0m0s
enableProfiling: false
featureGates: ""
healthzBindAddress: 0.0.0.0:10256
hostnameOverride: ""
iptables:
  masqueradeAll: false
  masqueradeBit: 14
  minSyncPeriod: 0s
  syncPeriod: 30s
kind: KubeProxyConfiguration
metricsBindAddress: 127.0.0.1:10249
```


© Copyright 2017 RX-M LLC

```
mode: ""
oomScoreAdj: -999
portRange: ""
resourceContainer: /kube-proxy
udpTimeoutMilliseconds: 250ms

user@nodea:~$
```

Note the `kubeconfig` and `clusterCIDR` configs; in previous versions of K8s, we used the `--kubeconfig` to tell kube-proxy about our cluster. Now we will put the path to the kubeconfig file in kube-proxy's config along with the value for the `--service-cluster-ip-range` flag we gave to the api-server.

```
user@nodea:~$ sudo vim kube-proxy-config
user@nodea:~$ head kube-proxy-config

apiVersion: componentconfig/v1alpha1
bindAddress: 0.0.0.0
clientConnection:
  acceptContentTypes: ""
  burst: 10
  contentType: application/vnd.kubernetes.protobuf
  kubeconfig: "nodea.kubeconfig"
  qps: 5
clusterCIDR: "10.0.0.0/16"
configSyncPeriod: 15m0s
```

Now we can run kube-proxy on nodea:

```
user@nodea:~$ sudo ~user/k8s/_output/bin/kube-proxy --config=kube-proxy-config

I0829 15:25:10.824741   52031 feature_gate.go:144] feature gates: map[]
I0829 15:25:10.863573   52031 server.go:478] Using iptables Proxier.
I0829 15:25:10.896757   52031 server.go:513] Tearing down userspace rules.
I0829 15:25:10.941624   52031 conntrack.go:98] Set sysctl 'net/netfilter/nf_conntrack_max' to 131072
I0829 15:25:10.942088   52031 conntrack.go:52] Setting nf_conntrack_max to 131072
I0829 15:25:10.961762   52031 conntrack.go:83] Setting conntrack hashsize to 32768
I0829 15:25:10.968380   52031 conntrack.go:98] Set sysctl 'net/netfilter/nf_conntrack_tcp_timeout_established' to
86400
I0829 15:25:10.968799   52031 conntrack.go:98] Set sysctl 'net/netfilter/nf_conntrack_tcp_timeout_close_wait' to
3600
```


© Copyright 2017 RX-M LLC

```
I0829 15:25:10.969226   52031 config.go:202] Starting service config controller
I0829 15:25:10.969456   52031 controller_utils.go:994] Waiting for caches to sync for service config controller
I0829 15:25:10.969962   52031 config.go:102] Starting endpoints config controller
I0829 15:25:10.970001   52031 controller_utils.go:994] Waiting for caches to sync for endpoints config controller
I0829 15:25:11.070517   52031 controller_utils.go:1001] Caches are synced for endpoints config controller
I0829 15:25:11.070568   52031 controller_utils.go:1001] Caches are synced for service config controller
...
```

Rerun the iptables dump on the NAT table on nodea:

```
user@nodea:~$ sudo iptables -L -nv -t nat

Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-SERVICES  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes service
portals */
  510 30688 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0           ADDRTYPE match dst-type
LOCAL

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-SERVICES  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes service
portals */
  133  7980 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8          ADDRTYPE match dst-type
LOCAL

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
  943 70726 KUBE-POSTROUTING  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes
postrouting rules */
    2   144 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0

Chain KUBE-MARK-DROP (0 references)
 pkts bytes target     prot opt in     out     source               destination
```


© Copyright 2017 RX-M LLC

```
    0     0 MARK       all  --  *      *       0.0.0.0/0            0.0.0.0/0            MARK or 0x8000

Chain KUBE-MARK-MASQ (5 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MARK       all  --  *      *       0.0.0.0/0            0.0.0.0/0            MARK or 0x4000

Chain KUBE-NODEPORTS (1 references)
 pkts bytes target     prot opt in     out     source               destination

Chain KUBE-POSTROUTING (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MASQUERADE  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes service
traffic requiring SNAT */ mark match 0x4000/0x4000

Chain KUBE-SEP-2YXAMM7IVAYA72O7 (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       172.18.0.2           0.0.0.0/0            /* default/nsvc: */
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            /* default/nsvc: */ tcp
to:172.18.0.2:80

Chain KUBE-SEP-JTQ5MRUTX2PVOEMO (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       172.16.151.203       0.0.0.0/0            /*
default/kubernetes:https */
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            /*
default/kubernetes:https */ recent: SET name: KUBE-SEP-JTQ5MRUTX2PVOEMO side: source mask: 255.255.255.255 tcp
to:172.16.151.203:6443

Chain KUBE-SEP-XDPNDAH2CYMNR5MR (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       172.17.0.2           0.0.0.0/0            /* default/nsvc: */
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            /* default/nsvc: */ tcp
to:172.17.0.2:80

Chain KUBE-SERVICES (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !10.0.0.0/16          10.0.0.1             /*
default/kubernetes:https cluster IP */ tcp dpt:443
    0     0 KUBE-SVC-NPX46M4PTMTKRN6Y  tcp  --  *      *       0.0.0.0/0            10.0.0.1             /*
default/kubernetes:https cluster IP */ tcp dpt:443
    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !10.0.0.0/16          10.0.44.224          /* default/nsvc:
cluster IP */ tcp dpt:2000
    0     0 KUBE-SVC-254CYKZ73JNCZ5NW  tcp  --  *      *       0.0.0.0/0            10.0.44.224          /*
```


© Copyright 2017 RX-M LLC

```
default/nsvc: cluster IP */ tcp dpt:2000
    0     0 KUBE-NODEPORTS  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* kubernetes
service nodeports; NOTE: this must be the last rule in this chain */ ADDRTYPE match dst-type LOCAL

Chain KUBE-SVC-254CYKZ73JNCZ5NW (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-SEP-XDPNDAH2CYMNR5MR  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /*
default/nsvc: */ statistic mode random probability 0.50000000000
    0     0 KUBE-SEP-2YXAMM7IVAYA72O7  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /*
default/nsvc: */

Chain KUBE-SVC-NPX46M4PTMTKRN6Y (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-SEP-JTQ5MRUTX2PVOEMO  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /*
default/kubernetes:https */ recent: CHECK seconds: 10800 reap name: KUBE-SEP-JTQ5MRUTX2PVOEMO side: source mask:
255.255.255.255
    0     0 KUBE-SEP-JTQ5MRUTX2PVOEMO  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /*
default/kubernetes:https */

user@nodea:~$
```

Wow, the kube-proxy has been busy! Search for the VIP of our service:

```
user@nodea:~$ sudo iptables -L -nv -t nat | grep 10.0.44.224

    0     0 KUBE-MARK-MASQ  tcp  --  *      *      !10.0.0.0/16          10.0.44.224          /* default/nsvc:
cluster IP */ tcp dpt:2000
    0     0 KUBE-SVC-254CYKZ73JNCZ5NW  tcp  --  *      *       0.0.0.0/0            10.0.44.224          /*
default/nsvc: cluster IP */ tcp dpt:2000

user@nodea:~$
```

The proxy has created a rule to intercept all traffic heading to our service VIP on port 2000. Examine the chain created for our service:

```
user@nodea:~$ sudo iptables -L -nv -t nat | grep -A4 'Chain KUBE-SVC-254CYKZ73JNCZ5NW'

Chain KUBE-SVC-254CYKZ73JNCZ5NW (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-SEP-XDPNDAH2CYMNR5MR  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /*
```


© Copyright 2017 RX-M LLC

```
default/nsvc: */ statistic mode random probability 0.50000000000
    0     0 KUBE-SEP-2YXAMM7IVAYA72O7  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /*
default/nsvc: */

user@nodea:~$
```

The iptables chain listed randomly chooses one of the two implementation pods backing our service. Display the chain for the first target pod:

```
user@nodea:~$ sudo iptables -L -nv -t nat | grep -A4 'Chain KUBE-SEP-XDPNDAH2CYMNR5MR'

Chain KUBE-SEP-XDPNDAH2CYMNR5MR (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-MARK-MASQ  all  --  *      *       172.17.0.2           0.0.0.0/0            /* default/nsvc: */
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            /* default/nsvc: */ tcp
to:172.17.0.2:80

user@nodea:~$
```

The DNAT rule takes all traffic and sends it to 172.17.0.2:80. Perfect!

Try curling your service using the service VIP and port:

```
user@nodea:~$ curl -I 10.0.44.224:2000

HTTP/1.1 200 OK
Server: nginx/1.7.9
Date: Tue, 29 Aug 2017 22:27:42 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 23 Dec 2014 16:25:09 GMT
Connection: keep-alive
ETag: "54999765-264"
Accept-Ranges: bytes

user@nodea:~$
```

Miracles.

© Copyright 2017 RX-M LLC

7. Completing the nodeb configuration

Lastly, to complete our compliment of cluster services let's start the kube-proxy on nodeb and test the service.

To begin, copy the `kube-proxy-config` from nodea to nodeb or run kube-proxy with the `--write-config-to` flag to generate the default config and edit it. In either case, remember to replace the *nodea.kubeconfig* so that it uses the nodeb.kubeconfig!

When the config file is ready, run the kube-proxy **in a new terminal or tab** on nodeb:

```
user@nodeb:~$ sudo ~user/kube-bin/kube-proxy --config=kube-proxy-config

I0829 15:29:01.976639    7452 feature_gate.go:144] feature gates: map[]
I0829 15:29:02.007526    7452 server.go:478] Using iptables Proxier.
I0829 15:29:02.046912    7452 server.go:513] Tearing down userspace rules.
I0829 15:29:02.082347    7452 conntrack.go:98] Set sysctl 'net/netfilter/nf_conntrack_max' to 131072
I0829 15:29:02.082441    7452 conntrack.go:52] Setting nf_conntrack_max to 131072
I0829 15:29:02.083162    7452 conntrack.go:83] Setting conntrack hashsize to 32768
I0829 15:29:02.085916    7452 conntrack.go:98] Set sysctl 'net/netfilter/nf_conntrack_tcp_timeout_established' to
86400
I0829 15:29:02.085988    7452 conntrack.go:98] Set sysctl 'net/netfilter/nf_conntrack_tcp_timeout_close_wait' to
3600
I0829 15:29:02.088327    7452 config.go:202] Starting service config controller
I0829 15:29:02.088348    7452 controller_utils.go:994] Waiting for caches to sync for service config controller
I0829 15:29:02.088387    7452 config.go:102] Starting endpoints config controller
I0829 15:29:02.088396    7452 controller_utils.go:994] Waiting for caches to sync for endpoints config controller
I0829 15:29:02.194582    7452 controller_utils.go:1001] Caches are synced for endpoints config controller
I0829 15:29:02.194656    7452 controller_utils.go:1001] Caches are synced for service config controller
...
```

Now try curling the nsvc service from nodeb:

```
user@nodeb:~$ curl -I 10.0.44.224:2000

HTTP/1.1 200 OK
Server: nginx/1.7.9
Date: Tue, 29 Aug 2017 22:29:37 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 23 Dec 2014 16:25:09 GMT
```


© Copyright 2017 RX-M LLC

```
Connection: keep-alive
ETag: "54999765-264"
Accept-Ranges: bytes

user@nodeb:~$
```

Mega.

You have now setup all of the core parts of a Kubernetes cluster the hard way (and hopefully learned something and had some fun in the process).

- remove resources

Congratulations you have successfully completed the Kubernetes services Lab!

Copyright (c) 2014-2017 RX-M LLC, Cloud Native Consulting, all rights reserved

© Copyright 2017 RX-M LLC