RX-M Cloud Native Consulting

Kubernetes

Lab 2 – Kubernetes Exploration

Kubernetes clusters track and manage objects of various “kinds”. Applications make use of four kinds of objects in particular:

- Pods – groups of containers deployed as a unit
- Replica Sets – sets of pods defined by a template which the Controller Manager replicates across the cluster
- Deployments – a rollout strategy for pods and replica sets
- Services – end points used to distribute requests to one of a pod’s replicas

Thus basic Kubernetes applications consist of pods, which implement the application functionality; replica sets, which ensure pods are always available; and Services which expose a dynamic set of pods to clients as a single endpoint. deployments describe how to launch or upgrade a given application.

1. kubectl

The `kubectl` command provides a range of features we can use with Kubernetes. Run `kubectl` without arguments to get a list of the available commands.

```
ubuntu@ip-10-0-2-200:~$ kubectl

kubectl controls the Kubernetes cluster manager.

Find more information at: https://kubernetes.io/docs/reference/kubectl/overview/

Basic Commands (Beginner):
  create         Create a resource from a file or from stdin.
  expose         Take a replication controller, service, deployment or pod and
expose it as a new Kubernetes Service
  run            Run a particular image on the cluster
  set            Set specific features on objects

Basic Commands (Intermediate):
  explain        Documentation of resources
  get            Display one or many resources
  edit           Edit a resource on the server
  delete         Delete resources by filenames, stdin, resources and names, or by
resources and label selector

Deploy Commands:
  rollout        Manage the rollout of a resource
  scale          Set a new size for a Deployment, ReplicaSet, Replication
Controller, or Job
  autoscale      Auto-scale a Deployment, ReplicaSet, or ReplicationController
```

```
Cluster Management Commands:
  certificate    Modify certificate resources.
  cluster-info   Display cluster info
  top            Display Resource (CPU/Memory/Storage) usage.
  cordon         Mark node as unschedulable
  uncordon       Mark node as schedulable
  drain          Drain node in preparation for maintenance
  taint          Update the taints on one or more nodes

Troubleshooting and Debugging Commands:
  describe       Show details of a specific resource or group of resources
  logs           Print the logs for a container in a pod
  attach         Attach to a running container
  exec           Execute a command in a container
  port-forward   Forward one or more local ports to a pod
  proxy          Run a proxy to the Kubernetes API server
  cp             Copy files and directories to and from containers.
  auth           Inspect authorization

Advanced Commands:
  diff           Diff live version against would-be applied version
  apply          Apply a configuration to a resource by filename or stdin
  patch          Update field(s) of a resource using strategic merge patch
  replace        Replace a resource by filename or stdin
  wait           Experimental: Wait for a specific condition on one or many
resources.
  convert        Convert config files between different API versions

Settings Commands:
  label          Update the labels on a resource
  annotate       Update the annotations on a resource
  completion     Output shell completion code for the specified shell (bash or
zsh)

Other Commands:
  api-resources  Print the supported API resources on the server
  api-versions   Print the supported API versions on the server, in the form of
"group/version"
  config         Modify kubeconfig files
  plugin         Provides utilities for interacting with plugins.
  version        Print the client and server version information

Usage:
  kubectl [flags] [options]

Use "kubectl <command> --help" for more information about a given command.
Use "kubectl options" for a list of global command-line options (applies to all
commands).
ubuntu@ip-10-0-2-200:~$
```

Take a moment to review available options. One useful subcommand is the global options, take a moment to review the output of `kubectl options`.

To use the `kubectl` command to control a remote cluster we must specify the cluster endpoint to `kubectl`. The `kubectl` command can be used to control several clusters from a single workstation. Clusters are given a name and

settings, including the IP address and port of the cluster API service.

To get configuration help issue the `kubectl help` subcommand.

```
ubuntu@ip-10-0-2-200:~$ kubectl help config

Modify kubeconfig files using subcommands like "kubectl config set current-
context my-context"

The loading order follows these rules:

  1. If the --kubeconfig flag is set, then only that file is loaded. The flag may
only be set once and no merging takes
place.
  2. If $KUBECONFIG environment variable is set, then it is used as a list of
paths (normal path delimitting rules for
your system). These paths are merged. When a value is modified, it is modified in
the file that defines the stanza. When
a value is created, it is created in the first file that exists. If no files in
the chain exist, then it creates the
last file in the list.
  3. Otherwise, ${HOME}/.kube/config is used and no merging takes place.

Available Commands:
  current-context Displays the current-context
  delete-cluster  Delete the specified cluster from the kubeconfig
  delete-context  Delete the specified context from the kubeconfig
  get-clusters    Display clusters defined in the kubeconfig
  get-contexts    Describe one or many contexts
  rename-context  Renames a context from the kubeconfig file.
  set             Sets an individual value in a kubeconfig file
  set-cluster     Sets a cluster entry in kubeconfig
  set-context     Sets a context entry in kubeconfig
  set-credentials Sets a user entry in kubeconfig
  unset           Unsets an individual value in a kubeconfig file
  use-context     Sets the current-context in a kubeconfig file
  view            Display merged kubeconfig settings or a specified kubeconfig
file

Usage:
  kubectl config SUBCOMMAND [options]

Use "kubectl <command> --help" for more information about a given command.
Use "kubectl options" for a list of global command-line options (applies to all
commands).
ubuntu@ip-10-0-2-200:~$
```

Run the `kubectl config view` subcommand again to display the current client configuration.

```
ubuntu@ip-10-0-2-200:~$ kubectl config view

apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
```

```
    server: https://10.0.2.200:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
ubuntu@ip-10-0-2-200:~$
```

When you run *kubectl* commands a context is required. The context tells *kubectl* which cluster to connect to and which user to authenticate as. As you can see the values kubeadm configured means the `kubectl` command tries to reach the API server on port 6443 via our host's IP with TLS.

To view the REDACTED elements, add `--flatten`.

We can configure `kubectl` explicitly so that we can adjust our cluster settings in the future if need be. Get help on the `config set-cluster` subcommand:

```
ubuntu@ip-10-0-2-200:~$ kubectl help config set-cluster

Sets a cluster entry in kubeconfig.

Specifying a name that already exists will merge new fields on top of existing
values for those fields.

Examples:
  # Set only the server field on the e2e cluster entry without touching other
values.
  kubectl config set-cluster e2e --server=https://1.2.3.4

  # Embed certificate authority data for the e2e cluster entry
  kubectl config set-cluster e2e --certificate-
authority=~/.kube/e2e/kubernetes.ca.crt

  # Disable cert checking for the dev cluster entry
  kubectl config set-cluster e2e --insecure-skip-tls-verify=true

Options:
      --embed-certs=false: embed-certs for the cluster entry in kubeconfig

Usage:
  kubectl config set-cluster NAME [--server=server] [--certificate-
authority=path/to/certificate/authority]
[--insecure-skip-tls-verify=true] [options]

Use "kubectl options" for a list of global command-line options (applies to all
commands).
ubuntu@ip-10-0-2-200:~$
```

`kubectl` configuration data is saved in a YAML file in your `$HOME/.kube` directory using thes commands. Display the configuration file we copied in lab 2.

```
ubuntu@ip-10-0-2-200:~$ ls -la ~/.kube/

total 24
drwxrwxr-x  4 user user 4096 Dec  4 11:30 .
drwxr-xr-x 17 user user 4096 Dec  4 12:57 ..
drwxr-xr-x  3 user user 4096 Dec  4 11:30 cache
-rw-------  1 user root 5454 Dec  4 11:29 config
drwxrwxr-x  3 user user 4096 Dec  4 12:20 http-cache
ubuntu@ip-10-0-2-200:~$
```

Display the contents of the config file:

```
ubuntu@ip-10-0-2-200:~$ cat ~/.kube/config

apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN5RENDQWJDZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQ
kFRc0ZBREFWTVJNd0VRWURWUVFERXdwcmRXSmwKY201bGRHVnpNQjRYRFRFM01USXdOREU1TVRFd01sb1hEVE
kzTVRJd01qRTVNVEV3TWxvd0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pDQ0FTSXdEUVlKS29aSWh
2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBTEI1CjhEdmpESFd0OGxnLy9XR2pqcXl1WFQ5NDFCZ0tBNzg0
Z1c4WDZDQlJDQWNZQVQ1NzdaSTdHVEF3Uks0RGJMdHAKT3NhbFZFeE1KbHVzc3Z1YXAvblBwMFlvS0hTZDByd
WRGNVVzWDdUN1NZVjJ3aklOS0JyeTBOUWpUdHJTOHo4VAp6RjVBMVVLV2lVb2RHZUhHSWtBVkR4V0NabWhRVy
80d0xoeUliVDdZcHNsbnFFaStDM1M3eHNnU1hsVTJCWi8yCm8vTldHODYyVFZlMS9IZVFWNHgyNEJQUnl1Q3R
tcjVuUGVaMmxGS3JvaDRJb3B3T0NUYlZXNjl0TjNidjhtYkMKemtNWjdhMWU2SjBJNDNBOENlR3liMUlFZWQ5
RndmcU1RVmZnR09zN1A2RVY3L3YzVEZERDZjeXJhVXFYenNlSgpIQ2x4S29UbmpkdE4rdWJTYWxrQ0F3RUFBY
U1qTUNFd0RnWURWUjBQQVFIL0JBUURBZ0trTUE4R0ExVWRFd0VCCi93UUZNQU1CQWY4d0RRWUpLb1pJaHZjTk
FRRUxCUUFEZ2dFQkFIcjU4VUwwcFlncVcrQkJEc08vOUNUei9NWkcKdXhhN1BPdEtOVUQ2V09WdWZCUXRqN2o
vR2lVaW5FNGkvTDNjVjJjNnBhYUY2TFZlQWF2VlpseWMvbERTYjZ4RQpoZFRkbFdWOG41WXFzR21pcjE3b0Zh
UjlEbi9HNUs5bDlzUmhLWDAzMUsxMUwwTW1SbnJLZ0llY2p6QUY4MFZmClhiazhpZkdFdC85cnp6OFovWjdib
TFhOFRpZ1JHN2ZRVmtZUWdBV2pabjFBWkRkVFlvcUNsZHVTTVRGUkF4RVQKTnBWZlB4bnBHZzBQU0llZ2xkYk
xkOHE4VHBWOU1qUHpFN2tPendlT3ZDQ3hVbjhnLyt0ZHZadi9kZ3MxMmlScwpGL3dJUi9qdWxlakhWRUQyNEo
2UmR0aStJVURZaXZXNDQzZlNxaERpdDI4UlVoVFY5MWxUT3pyM3hRRT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUt
LS0tLQo=
    server: https://10.0.2.200:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM4akNDQWRxZ0F3SUJBZ0lJVjR1Q056a2RSUEV3RFFZS
```

```
ktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB4TnpFeU1EUXhPVE
V4TURKYUZ3MHhPREV5TURReE9URXhNRE5hTURReApGekFWQmdOVkJBb1REbk41YzNSbGJUcHRZWE4wWlhKek1
Sa3dGd1lEVlFRREV4QnJkV0psY201bGRHVnpMV0ZrCmJXbHVNSUlCSWpBTkJna3Foa2lHOXcwQkFRUZBQU9D
QVE4QU1JSUJDZ0tDQVFFQXdzZ0FUQzdLa0szUlhMRWEKeWp1eG43S2pVR21LV3RwNlVTTnJnWEk5WWlkNXdWZ
WRNMCt2amttSGM3WjgzYTJrUFg1SExGOEdwcEFiS2FNTgpQSlVCY205ekRKZk10Q1FKWFBhaVluQkFET3hRMT
VPL0ZIWGdLTG85ZDh1ay9Mcm5jZFUybnBhclh3dnFmdnlxCitvVVdFQ3kvaTJHL0V0NkRkNlJUcmIybGk4MDl
YSXpQZW4xczQxUkZ1Yko5T01vendBa29FdkJROUR3RC9EUUwKaWxVeGwxOFY5WVhMakVFQmFES2MxR3NSOThq
UlZJRmVKT29TaWUxQmYveTJaMHphVHRHbzhKc3Mrb1JKdmVrbAorMGZLYU1UcGxRYUQrWmR6OFcrZWRwVUZsd
1h3RTFTbi9JcDVxNVJMdlh3TVB0Qkx4Tk1GdXc5bjZzcXM0Mm9vCnB5MlNjd0lEQVFBQm95Y3dKVEFPQmdOVk
hROEJBZjhFQkFNQ0JhQXdFd1lEVlIwbEJBd3dDZ1lJS3dZQkJRVUgKQXdJd0RRWUpLb1pJaHZjTkFRUxCUUF
EZ2dFQkFJQk1qV3lnbUtlVkloMk5MRkxVQnh5UDRYVExwRmZWNnpaeQp2Ylc1SmcxQXNUUGZFQmcxSEhKRk8x
b2g2L21rQ05PTnZESXN3bDR2bi9UWWlzZlJBQ09pUjNHeDkrYnZLcUl4CkYydDltaitnbXFvc0Nqa0ZDZGFud
jE3RzNwVFJIcjhrQndrVnpKeStIL1dzWUtpMkIxcHdId3hCazJkcmlXZlIKN0ErTFd5VU5uamNnNWdiam83d1
FObWIzTG9zMVJMMys4UU1scFlCTjMrdFZVL1pON3l3NG9LVGtFeDNOR1RlSApXaE9kT0xaTzAxSzhiWWtqQ2h
hSXRITDlJa0I4eW5MS0lRcXU2UndYclJobkRrdlF1MDZTY0Q4dW4xS0VkdUs0CkVrMnFmdW0vdHpRN0U2aEFB
cEhtMldaLzBxcm5jU0h0Y2U1cE93Q2RqMHRJT2s5Y1NRUT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    client-key-data:
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcEFJQkFBS0NBUUVBd3NnQVRDN0trSzNSWExFY
XlqdXhuN0tqVUdtS1d0cDZVU05yZ1hJOVlpZDV3VmVkCk0wK3Zqa21IYzdaODNhMmtQWDVITEY4R3BwQWJLYU
1OUEpVQmNtOXpESmZNdENRSlhQYWlZbkJBRE94UTE1Ty8KRkhYZ0tMbzlkOHVrL0xybmNkVTJucGFyWHd2cWZ
2eXErb1VXRUN5L2kyRy9FdDZEZDZSVHJiMmxpODA5WEl6UAplbjFzNDFSRnViSjlPTW96d0Frb0V2QlE5RHdE
L0RRTGlsVXhsMThWOVlYTGpFRUJhREtjMUdzUjk4alJWSUZlCkpPb1NpZTFCZi95MlowemFUdEdvOEpzcytvU
kp2ZWtsKzBmS2FNVHBsUWFEK1pkejhXK2VkcFVGbHdYd0UxU24KL0lwNXE1Ukx2WHdNUHRCTHhOTUZ1dzluNn
NxczQyb29weTJTY3dJREFRQUJBb0lCQVFDTWFudjBiNkx0NjdCTApQdzJPRHJ4ODROM2s0VUN4UUdEL1R5WjR
HZS93YTM3Vmg2QzIxZEk4SS91MnlpNTM3RzdET2Q1N09VSGxIMnZGCmQxcmFlcHloYjRDajZtYlMwa3ZCcmJi
Z2VnVmlpNVczNHpYVHZTcE1rZjR2OXluSVc4RHZpZ0luRDMwWk50RGsKbktkeGxER1VsWDI4TUVuN2cxZUpEM
2lYZm1qeVAwNExFR1BkbFNYRVMxbExlNHN1R0RJUTlVTHVvanpzRkFqYgpneUdEUzIwVTZ5N2FEc0lZdVdEQ2
ZhdEx2Ym5mQy9VZSsvYnN6b1Q3dEdtRzlzYlZXL013S21UQmI0MmlaeFhECnN2RUNXYlJ2c2FPaVBmckJ2bk4
1a3M4enhRQkllWlVYRnlzRmUxc09kb3Q0Tms5Z29MU2RZRklCaDBRV3drKy8KZnRUSGZUa0JBb0dCQU1oWGI0
cG8xZGhubUw1VnpxWlZIbUFScEU3STRGR1ZHY0FlVXNCczUvaFI5UGhhNE5yKwp6MlFWSktBK05ZYlJYZThPd
WxtdzRzeUpaUFN2RVBFYjlmVG1pY091clZoMHBCSHlzYnZEY0R5TjdhclNLajMzCksyQzVObGhFTFo2YmJETz
lLamU0cFNWazZwUTdVeHlQUHJyRE4rdGhvRmNDdWRsTzBlcFhRQUtCQW9HQkFQamwKSENZR3NyMklMd3RFb1p
WQy9MS3orU2t1WU0wekRkZkhtVmNXY1VUem90bG5uVWhLN2FvYzZscjRjVEYxTkZBSQpBc0NMS3k3elJzY05L
YlM1UFdwQ1VMTjlMSVhHZEsrSGFtT1FBN0NCSUJBK1V4SHU0eHA4WEZFNGRKR3dlUThHCmoyM1p0aXlJSjJQU
jd1VVUvNDBWTWI5cjJEdEk4b3pid09pQjdqTHpBb0dCQUxKdExPb1lkRnhINThHY3FtOTUKWDVRd0lpWUl3Yi
9uQ3dnUCthTm5XekhOekY5a2tONTZFanNRbVk0ZDJZNDQ2Vy8vcmZnemtjcVlrMUZZbWI0agpLT3IwWjczZjF
JTHpYeXljK2E1QVliV01zaDl4RGk0aDlJQXdkRFlvZ25pLzg1NGcyM2pFK2xCVGtKaDBQWkV5CkFxeFRNWHB4
ZUZ3R0VYOTRzM3dDT1FBQkFvR0FNS0ZyVjQ2MWU4elJERTJUbUx0bTdtKzF2ak1lbk5sZDJneFkKektoSTUyV
UhLNTFRSU9EckFQTDNZMkRwbFBWR2pHQ1VVUlNnRW1OY0wrWkZnTmMweGI5QlQyQ2t3MXFjVC9PUQplUFdaa3
ZJWDFyU212SGxGakZaQ0gyaDlkajNaMlhLNXNZZjVUVWdwRWhyaHA0YnJ5NkFaZ1VKTUZJRlRTdXhoCkM4emZ
RcnNDZ1lCTExVTjRiM0xFakVORyt5K2tFcTFmcEp4MDJhNk4zeUN5SmFyNWIzbWVjS0YvMG5nTm9PZlcKcUFk
RWE3ZzFDaVVkMVJibGtZZHhxdFNiYUZEUzVERDI5MHVLOEZjWEVRdWRlcEJ1YklDQ0ozc3ZWdndIRElHegphd
m8zc3lSd3lVdGt0MmZmYkorbHJETE8zTmg3RnJiS3NOWG5zTjgrYjVBeDBZSHhkMXRsUUE9PQotLS0tLUVORC
BSU0EgUFJJVkFURSBLRVktLS0tLQo=
ubuntu@ip-10-0-2-200:~$
```

The `kubectl config view` command will display nearly the same data, obfuscating the key data. The config file is simple and can easily be pre-generated and distributed to any client systems that require connection to a given cluster.

2. Test the Cluster

Now with our cluster running and `kubectl` configured lets issue some commands to test the Kubernetes cluster. The *cluster-info* subcommand can be used to test the cluster API end point and the get nodes command can be used to see the nodes in the cluster.

```
ubuntu@ip-10-0-2-200:~$ kubectl cluster-info

Kubernetes master is running at https://10.0.2.200:6443
KubeDNS is running at https://10.0.2.200:6443/api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
ubuntu@ip-10-0-2-200:~$
```

If you are really adventurous run the suggested command for a detailed cluster overview, careful though, its a lot of information!

```
ubuntu@ip-10-0-2-200:~$ kubectl cluster-info dump |& wc -l

3413
ubuntu@ip-10-0-2-200:~$
```

To get detailed node information use the *describe node* subcommand again on the desired node name:

```
ubuntu@ip-10-0-2-200:~$ kubectl describe node ip-10-0-2-200

Name:               ip-10-0-2-200
Roles:              master
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=ip-10-0-2-200
                    node-role.kubernetes.io/master=
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket:
/var/run/dockershim.sock
                    node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Thu, 06 Dec 2018 14:54:36 +0000
Taints:             <none>
Unschedulable:      false
Conditions:
  Type                 Status  LastHeartbeatTime
LastTransitionTime               Reason                       Message
  ----                 ------  -----------------
--                 ------                       -------
  NetworkUnavailable   False   Thu, 06 Dec 2018 17:01:52 +0000   Thu, 06 Dec 2018
17:01:52 +0000   WeaveIsUp                    Weave pod has set this
  OutOfDisk            False   Thu, 06 Dec 2018 18:27:32 +0000   Thu, 06 Dec 2018
14:54:26 +0000   KubeletHasSufficientDisk     kubelet has sufficient disk space
available
  MemoryPressure       False   Thu, 06 Dec 2018 18:27:32 +0000   Thu, 06 Dec 2018
14:54:26 +0000   KubeletHasSufficientMemory   kubelet has sufficient memory
available
  DiskPressure         False   Thu, 06 Dec 2018 18:27:32 +0000   Thu, 06 Dec 2018
14:54:26 +0000   KubeletHasNoDiskPressure     kubelet has no disk pressure
  PIDPressure          False   Thu, 06 Dec 2018 18:27:32 +0000   Thu, 06 Dec 2018
14:54:26 +0000   KubeletHasSufficientPID      kubelet has sufficient PID
available
```

```
  Ready                True    Thu, 06 Dec 2018 18:27:32 +0000   Thu, 06 Dec 2018
17:01:59 +0000   KubeletReady                 kubelet is posting ready status.
AppArmor enabled
Addresses:
  InternalIP:  10.0.2.200
  Hostname:    ip-10-0-2-200
Capacity:
 cpu:                2
 ephemeral-storage:  20263528Ki
 hugepages-2Mi:      0
 memory:             4045044Ki
 pods:               110
Allocatable:
 cpu:                2
 ephemeral-storage:  18674867374
 hugepages-2Mi:      0
 memory:             3942644Ki
 pods:               110
System Info:
 Machine ID:                 9f1ba4ef4b924c148a4d816af9389de3
 System UUID:                EC2C41DA-D66D-DA59-F48A-5FBDCA03EE26
 Boot ID:                    db5b7a44-fb6f-427e-a6ad-9b22812da9b6
 Kernel Version:             4.4.0-1072-aws
 OS Image:                   Ubuntu 16.04.5 LTS
 Operating System:           linux
 Architecture:               amd64
 Container Runtime Version:  docker://18.9.0
 Kubelet Version:            v1.13.2
 Kube-Proxy Version:         v1.13.2
Non-terminated Pods:         (8 in total)
  Namespace                  Name                                          CPU
Requests  CPU Limits  Memory Requests  Memory Limits
  ---------                  ----                                          -----------
-  ----------  ---------------  -------------
  kube-system                coredns-576cbf47c7-7qlht                      100m (5%)
0 (0%)      70Mi (1%)        170Mi (4%)
  kube-system                coredns-576cbf47c7-qp7qk                      100m (5%)
0 (0%)      70Mi (1%)        170Mi (4%)
  kube-system                etcd-ip-10-0-2-200                            0 (0%)
0 (0%)      0 (0%)           0 (0%)
  kube-system                kube-apiserver-ip-10-0-2-200                  250m (12%)
0 (0%)      0 (0%)           0 (0%)
  kube-system                kube-controller-manager-ip-10-0-2-200         200m (10%)
0 (0%)      0 (0%)           0 (0%)
  kube-system                kube-proxy-rmxrk                              0 (0%)
0 (0%)      0 (0%)           0 (0%)
  kube-system                kube-scheduler-ip-10-0-2-200                  100m (5%)
0 (0%)      0 (0%)           0 (0%)
  kube-system                weave-net-4xsgf                               20m (1%)
0 (0%)      0 (0%)           0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource  Requests    Limits
  --------  --------    ------
  cpu       770m (38%)  0 (0%)
  memory    140Mi (3%)  340Mi (8%)
ubuntu@ip-10-0-2-200:~$
```

Describe provides a wealth of node information. Your report will be similar but different than the one above.

- How much memory does your node have?
- How many CPUs?
- How many pods can your node run?
- What container runtime is the `kubelet` using?
- What version of `kubelet` is your node running?

Previously we used the `version` subcommand to discover the version of the `kubectl` client but now that our config is in place we can also see the version of the cluster API Server.

```
ubuntu@ip-10-0-2-200:~$ kubectl version

Client Version: version.Info{Major:"1", Minor:"12", GitVersion:"v1.12.3",
GitCommit:"435f92c719f279a3a67808c80521ea17d5715c66", GitTreeState:"clean",
BuildDate:"2018-11-26T12:57:14Z", GoVersion:"go1.10.4", Compiler:"gc",
Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"12", GitVersion:"v1.12.3",
GitCommit:"435f92c719f279a3a67808c80521ea17d5715c66", GitTreeState:"clean",
BuildDate:"2018-11-26T12:46:57Z", GoVersion:"go1.10.4", Compiler:"gc",
Platform:"linux/amd64"}
ubuntu@ip-10-0-2-200:~$
```

If you are familiar with Golang, notice the use of the `gc` tool chain (vs gccgo).

3. Creating Applications

With our cluster running and `kubectl` configured we can try to start a simple application on the cluster. The `kubectl` command provides a *get* subcommand which can be used to get information on any one of the key Kubernetes component types: deployments, pods, replica sets, and Services. While you can type `kubectl get replicasets`, that would be fairly inhumane so `kubectl` allows you to use the abbreviation `rs` for replica sets.

If you want to save yourself even more typing. Here is tab completion without the mentioned fix.

```
ubuntu@ip-10-0-2-200:~$ kubectl get

Desktop/                  .kube/                    Public/
...

ubuntu@ip-10-0-2-200:~$ kubectl get
```

You can enable temporary kubectl bash completion with:

```
ubuntu@ip-10-0-2-200:~$ source <(kubectl completion bash)

ubuntu@ip-10-0-2-200:~$
```

And after.

```
ubuntu@ip-10-0-2-200:~$ kubectl get

certificatesigningrequest  deployment                networkpolicy
replicaset                 statefulset
...

ubuntu@ip-10-0-2-200:~$ kubectl get
```

That is much better!

In a new shell, list the currently running services, deployments, replica sets, and pods on your cluster:

```
ubuntu@ip-10-0-2-200:~$ kubectl get service,deployments,rs,pods

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   3h40m
ubuntu@ip-10-0-2-200:~$
```

The only service running in our cluster is the *kubernetes* service itself. We have no deployments, replica sets, or pods yet (in our namespace). Do the same for the resources under the kube-system namespace, more on namespaces later.

```
ubuntu@ip-10-0-2-200:~$ kubectl get service,deployments,rs,pods --namespace=kube-
system

NAME               TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE
service/kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP   3h41m

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/coredns   2/2     2            2           3h41m

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.extensions/coredns-576cbf47c7   2         2         2       3h41m

NAME                                     READY   STATUS    RESTARTS   AGE
pod/coredns-576cbf47c7-7qlht             1/1     Running   0          3h41m
pod/coredns-576cbf47c7-qp7qk             1/1     Running   0          3h41m
pod/etcd-ip-10-0-2-200                   1/1     Running   0          3h40m
pod/kube-apiserver-ip-10-0-2-200         1/1     Running   0          3h40m
pod/kube-controller-manager-ip-10-0-2-200   1/1     Running   0          3h40m
pod/kube-proxy-rmxrk                     1/1     Running   0          3h41m
pod/kube-scheduler-ip-10-0-2-200         1/1     Running   0          3h40m
pod/weave-net-4xsgf                      2/2     Running   0          94m
ubuntu@ip-10-0-2-200:~$
```

We can view all namespaces via `--all-namespaces` (if we have permission).

To test our cluster lets run a single container pod. When configured with the Docker Engine as the container manager,

we can run any container image that Docker has preinstalled or knows how to download.

```
ubuntu@ip-10-0-2-200:~$ kubectl run my-nginx --generator=run-pod/v1 --
image=nginx:1.11 --port=80

pod/my-nginx created
ubuntu@ip-10-0-2-200:~$
```

The pod name is "my-nginx" and the image we used is "nginx", an official image pulled from Docker Hub by the Docker Engine in the background. The port switch tells Kubernetes the service port for our pod which will allow us to share the service with its users over that port (the program must actually use that port for this to work).

List the pods running on the cluster:

```
ubuntu@ip-10-0-2-200:~$ kubectl get pods

NAME       READY   STATUS    RESTARTS   AGE
my-nginx   1/1     Running   0          37s
ubuntu@ip-10-0-2-200:~$
```

This shows that our pods are deployed and up to date. It may take a bit to pull the Docker images (Ready might be 0).

You can use the `docker container ls` subcommand to display the containers running under the Docker Engine:

```
ubuntu@ip-10-0-2-200:~$ docker container ls --filter "name=nginx"

CONTAINER ID        IMAGE                  COMMAND                  CREATED
STATUS              PORTS               NAMES
01b0c8428c5c        5766334bdaa0           "nginx -g 'daemon of…"   About a
minute ago   Up About a minute                       k8s_my-nginx_my-
nginx_default_f2ab7342-f9e4-11e8-8781-02d9a858fbbc_0
a7b103781d85        k8s.gcr.io/pause:3.1   "/pause"                 About a
minute ago   Up About a minute                       k8s_POD_my-
nginx_default_f2ab7342-f9e4-11e8-8781-02d9a858fbbc_0
ubuntu@ip-10-0-2-200:~$
```

As you can see, while our *run* subcommand requested that Kubernetes run a container but 2 containers were launched at that time.

In Kubernetes, each Pod instance has an infrastructure container, which is the first container that the `kubelet` instantiates. The infrastructure container uses the image "k8s.gcr.io/pause:3.1" and acquires the pod's IP as well as a pod wide network and IPC namespace. All of the other containers in the pod then join the infrastructure container's network (--net) and IPC (--ipc) namespace allowing containers in the pod to easily communicate. The initial process ("/pause") that runs in the infrastructure container does nothing, its sole purpose is to act as the anchor for the pod and its shared namespaces.

You can learn more about the pause container by looking at the source and ultimately what is "pause()".

- https://github.com/kubernetes/kubernetes/tree/master/build/pause

- https://github.com/kubernetes/kubernetes/blob/master/build/pause/pause.c
- `man 2 pause` or http://man7.org/linux/man-pages/man2/pause.2.html

The Docker listing shows us 2 containers, the pod having an infrastructure container (pause) and the container we asked for (nginx).

Kubernetes gives each pod a name and reports on the pod status, the number of times the pod has been restarted and the pod's uptime. You can find the pod names embedded in the container names displayed by the `docker container ls` command:

```
ubuntu@ip-10-0-2-200:~$ docker container ls --filter "name=nginx" --format "
{{.Names}}"

k8s_my-nginx_my-nginx_default_f2ab7342-f9e4-11e8-8781-02d9a858fbbc_0
k8s_POD_my-nginx_default_f2ab7342-f9e4-11e8-8781-02d9a858fbbc_0
ubuntu@ip-10-0-2-200:~$
```

Try killing the nginx container using the `docker container kill` subcommand and the ID of the underlying container based on the nginx image.

```
ubuntu@ip-10-0-2-200:~$ docker container kill \
$(docker container ls --filter "ancestor=nginx:1.11" --format {{.ID}} | head -1)

01b0c8428c5c
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ docker container ls --filter "name=nginx"

CONTAINER ID        IMAGE                  COMMAND                  CREATED
STATUS              PORTS               NAMES
9a1a44e5d0f1        5766334bdaa0           "nginx -g 'daemon of…"   9 seconds ago
Up 9 seconds                            k8s_my-nginx_my-nginx_default_f2ab7342-
f9e4-11e8-8781-02d9a858fbbc_1
a7b103781d85        k8s.gcr.io/pause:3.1   "/pause"                 4 minutes ago
Up 4 minutes                            k8s_POD_my-nginx_default_f2ab7342-f9e4-
11e8-8781-02d9a858fbbc_0
ubuntu@ip-10-0-2-200:~$
```

We can tell by the created time we have a new container. If you were fast enough, you may have seen the previous container exited. Docker terminates the container specified but Kubernetes has no knowledge of this action. When the Kubelet process, responsible for the pods assigned to this node, sees the missing container, it simply reruns the nginx image.

After some time, if you run the previous command with the `-a` flag, we can see the previous killed container and the newly created one.

```
ubuntu@ip-10-0-2-200:~$ docker container ls -a --filter "name=nginx"
```

```
CONTAINER ID        IMAGE                  COMMAND                  CREATED
STATUS                          PORTS               NAMES
9a1a44e5d0f1        5766334bdaa0           "nginx -g 'daemon of…"   About a
minute ago   Up About a minute                                   k8s_my-
nginx_my-nginx_default_f2ab7342-f9e4-11e8-8781-02d9a858fbbc_1
01b0c8428c5c        5766334bdaa0           "nginx -g 'daemon of…"   5 minutes ago
Exited (137) About a minute ago                         k8s_my-nginx_my-
nginx_default_f2ab7342-f9e4-11e8-8781-02d9a858fbbc_0
a7b103781d85        k8s.gcr.io/pause:3.1   "/pause"                 5 minutes ago
Up 5 minutes                                            k8s_POD_my-
nginx_default_f2ab7342-f9e4-11e8-8781-02d9a858fbbc_0
ubuntu@ip-10-0-2-200:~$
```

Notice that we killed container *01b0c8428c5c* in the example but the new container *9a1a44e5d0f1* was created to replace it. Kubernetes does not "resurrect" containers that have failed. This is important because the container's state may be the reason it failed. Rather, Kubernetes runs a fresh copy of the original image, ensuring the container has a clean new internal state (cattle not pets!).

4. Create a Service

In modern software engineering terms, a service is an encapsulated set of functionality made available to consumers through an API. The problem with our nginx application at present is that when containers die new ones are created. The fact that there are multiple containers and that containers come and go makes using the app difficult.

To simplify things Kubernetes makes it possible for us to expose our pods as a Service. The `kubectl` expose command does this.

Expose the my-nginx pod replica set as a service:

```
ubuntu@ip-10-0-2-200:~$ kubectl get pod

NAME       READY   STATUS    RESTARTS   AGE
my-nginx   1/1     Running   1          3m38s
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl expose $(kubectl get pod -o=name) --port=80

service/my-nginx exposed
ubuntu@ip-10-0-2-200:~$
```

This causes Kubernetes to create a conceptual Service for our pods, exposing the set of pods as a single endpoint for users. Use the *get services* subcommand to display your service.

```
ubuntu@ip-10-0-2-200:~$ kubectl get services

NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes          ClusterIP   10.96.0.1       <none>        443/TCP   4h25m
my-nginx-76bcc6d46  ClusterIP   10.100.43.206   <none>        80/TCP    15s
ubuntu@ip-10-0-2-200:~$
```

Kubernetes has given our service a virtual IP (VIP) address and it will now distribute client connections across the pods running my-nginx.

To test the Service try curling it:

```
ubuntu@ip-10-0-2-200:~$ NX_CIP=$(kubectl get services -o=custom-
columns=NAME:.spec.clusterIP,NAME:.metadata.name \
  | grep nginx | awk '{print $1}') && echo $NX_CIP

10.100.43.206

ubuntu@ip-10-0-2-200:~$ curl -I $NX_CIP

HTTP/1.1 200 OK
Server: nginx/1.11.13
Date: Wed, 28 Mar 2018 22:32:41 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 04 Apr 2017 15:01:57 GMT
Connection: keep-alive
ETag: "58e3b565-264"
Accept-Ranges: bytes
ubuntu@ip-10-0-2-200:~$
```

Success!

5. Pod exec

While Kubernetes delegates all of the direct container operations to the container manager (usually Docker) it does pass through some useful container features.

For example, imagine you need to discover the distro of one of your pods' containers. You can use the `kubectl exec` subcommand to run arbitrary commands within a pod.

Try listing the running pods and then executing the `cat /etc/os-release` command within one of your pods.

```
ubuntu@ip-10-0-2-200:~$ kubectl get pods

NAME       READY   STATUS    RESTARTS   AGE
my-nginx   1/1     Running   1          8m38s

ubuntu@ip-10-0-2-200:~$ kubectl exec my-nginx cat /etc/os-release

PRETTY_NAME="Debian GNU/Linux 8 (jessie)"
NAME="Debian GNU/Linux"
VERSION_ID="8"
VERSION="8 (jessie)"
ID=debian
HOME_URL="http://www.debian.org/"
SUPPORT_URL="http://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

```
ubuntu@ip-10-0-2-200:~$
```

Running `cat /etc/os-release` via `kubectl exec` produces the information we needed. The `exec` subcommand chooses the first container within the pod to execute the command.

If you would like to execute the command within a specific container you can use the *-c* switch. The *describe pod* command will give you a list of the containers within the pod. We can also retrieve JSON output and filter for it. Our current pod has only one container but we can still test the command.

Try it:

```
ubuntu@ip-10-0-2-200:~$ kubectl get pod my-nginx -o json | jq
".spec.containers[].name" -r

my-nginx
ubuntu@ip-10-0-2-200:~$
```

Use the *–c* switch to display the `os-release` file in the my-nginx container in the pod:

```
ubuntu@ip-10-0-2-200:~$ kubectl exec -c my-nginx my-nginx cat /etc/os-release

PRETTY_NAME="Debian GNU/Linux 8 (jessie)"
NAME="Debian GNU/Linux"
VERSION_ID="8"
VERSION="8 (jessie)"
ID=debian
HOME_URL="http://www.debian.org/"
SUPPORT_URL="http://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
ubuntu@ip-10-0-2-200:~$
```

6. System Logs

Each of the services composing our Kubernetes cluster emits a log file. In the current configuration, the `kubelet` log is controlled by systemd.

You can use the `journalctl` command to tail (`-n`) the output for the kubelet service unit (`-u`)

```
ubuntu@ip-10-0-2-200:~$ journalctl -n 400 --no-pager -u kubelet.service | grep -v
"no observation"

-- Logs begin at Wed 2018-03-28 20:50:52 UTC, end at Thu 2018-03-29 01:39:50 UTC.
--
Mar 28 22:11:54 ip-10-0-2-200 kubelet[6783]: Flag --client-ca-file has been
deprecated, This parameter should be set via the config file specified by the
Kubelet's --config flag. See https://kubernetes.io/docs/tasks/administer-
cluster/kubelet-config-file/ for more information.
Mar 28 22:11:54 ip-10-0-2-200 kubelet[6783]: Flag --cadvisor-port has been
deprecated, The default will change to 0 (disabled) in 1.12, and the cadvisor
```

```
port will be removed entirely in 1.13
Mar 28 22:11:54 ip-10-0-2-200 kubelet[6783]: I0328 22:11:54.739979    6783
feature_gate.go:226] feature gates: &{{} map[]}
Mar 28 22:11:54 ip-10-0-2-200 kubelet[6783]: F0328 22:11:54.740056    6783
server.go:218] unable to load client CA file /etc/kubernetes/pki/ca.crt: open
/etc/kubernetes/pki/ca.crt: no such file or directory
Mar 28 22:11:54 ip-10-0-2-200 systemd[1]: kubelet.service: Main process exited,
code=exited, status=255/n/a
Mar 28 22:11:54 ip-10-0-2-200 systemd[1]: kubelet.service: Unit entered failed
state.
Mar 28 22:11:54 ip-10-0-2-200 systemd[1]: kubelet.service: Failed with result
'exit-code'.
Mar 28 22:12:04 ip-10-0-2-200 systemd[1]: kubelet.service: Service hold-off time
over, scheduling restart.
Mar 28 22:12:04 ip-10-0-2-200 systemd[1]: Stopped kubelet: The Kubernetes Node
Agent.
Mar 28 22:12:04 ip-10-0-2-200 systemd[1]: Started kubelet: The Kubernetes Node
Agent.
Mar 28 22:12:04 ip-10-0-2-200 kubelet[6791]: Flag --pod-manifest-path has been
deprecated, This parameter should be set via the config file specified by the
Kubelet's --config flag. See https://kubernetes.io/docs/tasks/administer-
cluster/kubelet-config-file/ for more information.
Mar 28 22:12:04 ip-10-0-2-200 kubelet[6791]: Flag --allow-privileged has been
deprecated, will be removed in a future version

...

ubuntu@ip-10-0-2-200:~$
```

The rest of our services are running as containers.

We can use the `kubectl logs` command to display log output from our pods. Remember that Kubernetes system services run within the kube-system namespace by convention.

List the pods in the *kube-system* namespace:

```
ubuntu@ip-10-0-2-200:~$ kubectl get pods --namespace=kube-system

NAME                                    READY     STATUS    RESTARTS   AGE
etcd-ip-10-0-2-200                      1/1       Running   0          3h
kube-apiserver-ip-10-0-2-200            1/1       Running   0          3h
kube-controller-manager-ip-10-0-2-200   1/1       Running   0          3h
kube-dns-86f4d74b45-dqfs5               3/3       Running   0          3h
kube-proxy-xw6nh                        1/1       Running   0          3h
kube-scheduler-ip-10-0-2-200            1/1       Running   0          3h
weave-net-rmsx8                         2/2       Running   0          3h
ubuntu@ip-10-0-2-200:~$
```

Now display the last 10 lines from the API service:

```
ubuntu@ip-10-0-2-200:~$ kubectl logs --namespace=kube-system --tail=10 kube-
apiserver-ip-10-0-2-200
```

```
I0124 21:20:21.743616       1 storage_rbac.go:276] created
rolebinding.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-
system
I0124 21:20:21.785787       1 storage_rbac.go:276] created
rolebinding.rbac.authorization.k8s.io/system:controller:cloud-provider in kube-
system
I0124 21:20:21.825880       1 storage_rbac.go:276] created
rolebinding.rbac.authorization.k8s.io/system:controller:token-cleaner in kube-
system
I0124 21:20:21.866111       1 storage_rbac.go:276] created
rolebinding.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-
public
I0124 21:20:22.119230       1 controller.go:608] quota admission added evaluator
for: serviceaccounts
I0124 21:20:23.152183       1 controller.go:608] quota admission added evaluator
for: deployments.apps
I0124 21:20:23.189041       1 controller.go:608] quota admission added evaluator
for: daemonsets.apps
I0124 21:20:28.565046       1 controller.go:608] quota admission added evaluator
for: replicasets.apps
I0124 21:20:28.669494       1 controller.go:608] quota admission added evaluator
for: controllerrevisions.apps
I0124 21:30:43.442529       1 controller.go:608] quota admission added evaluator
for: daemonsets.extensions
ubuntu@ip-10-0-2-200:~$
```

Each Kubernetes service has its own log verbosity and each can be tuned. You can learn much by tracking the operations involved in starting a deployment.

Create a new single pod deployment with a descriptive name and then grep its activity in the logs.

```
ubuntu@ip-10-0-2-200:~$ kubectl run --generator=run-pod/v1 mylogtracker --image
nginx:1.11

pod/mylogtracker created
ubuntu@ip-10-0-2-200:~$
```

Again list the k8s system services, from here we can pick which logs to search for our new pod.

```
ubuntu@ip-10-0-2-200:~$ kubectl get pod --namespace=kube-system

NAME                                READY     STATUS    RESTARTS   AGE
etcd-ip-10-0-2-200                  1/1       Running   0          3h
kube-apiserver-ip-10-0-2-200        1/1       Running   0          3h
kube-controller-manager-ip-10-0-2-200   1/1       Running   0          3h
kube-dns-86f4d74b45-dqfs5           3/3       Running   0          3h
kube-proxy-xw6nh                    1/1       Running   0          3h
kube-scheduler-ip-10-0-2-200        1/1       Running   0          3h
weave-net-rmsx8                     2/2       Running   0          3h
ubuntu@ip-10-0-2-200:~$
```

Try the controller manager server first:

```
ubuntu@ip-10-0-2-200:~$ kubectl logs --namespace=kube-system kube-controller-
manager-ip-10-0-2-200 | grep mylogtracker

ubuntu@ip-10-0-2-200:~$
```

Controller manager only deals with replicated pods (ones using a controller); there won't be anything here for us.

Now take a look at the kubelet log:

```
ubuntu@ip-10-0-2-200:~$ journalctl -u kubelet.service | grep mylogtracker

Dec 04 13:28:35 ip-10-0-2-200 kubelet[74364]: I1204 13:28:35.182932   74364
reconciler.go:212] operationExecutor.VerifyControllerAttachedVolume started for
volume "default-token-2kmhb" (UniqueName: "kubernetes.io/secret/1584a8f2-d93a-
11e7-a277-000c29ae8ddc-default-token-2kmhb") pod "mylogtracker-6ff4ff6fd5-k5qrq"
(UID: "1584a8f2-d93a-11e7-a277-000c29ae8ddc")
ubuntu@ip-10-0-2-200:~$
```

Kubelet only reports information about our pod's volume.

You can also view the events taking place within the Kubernetes cluster itself using the events resource type.

Try getting events with `kubectl`:

```
ubuntu@ip-10-0-2-200:~$ kubectl get events

LAST SEEN   TYPE     REASON                   KIND   MESSAGE
11m         Normal   Scheduled                Pod    Successfully assigned
default/my-nginx to ubuntu
11m         Normal   Pulling                  Pod    pulling image "nginx:1.11"
10m         Normal   Pulled                   Pod    Successfully pulled image
"nginx:1.11"
9m13s       Normal   Created                  Pod    Created container
9m13s       Normal   Started                  Pod    Started container
9m13s       Normal   Pulled                   Pod    Container image
"nginx:1.11" already present on machine
81s         Normal   Scheduled                Pod    Successfully assigned
default/mylogtracker to ubuntu
81s         Normal   Pulled                   Pod    Container image
"nginx:1.11" already present on machine
81s         Normal   Created                  Pod    Created container
80s         Normal   Started                  Pod    Started container
31m         Normal   Starting                 Node   Starting kubelet.
31m         Normal   NodeHasSufficientMemory  Node   Node ubuntu status is now:
NodeHasSufficientMemory
31m         Normal   NodeHasNoDiskPressure    Node   Node ubuntu status is now:
NodeHasNoDiskPressure
31m         Normal   NodeHasSufficientPID     Node   Node ubuntu status is now:
NodeHasSufficientPID
31m         Normal   NodeAllocatableEnforced  Node   Updated Node Allocatable
limit across pods
31m         Normal   RegisteredNode           Node   Node ubuntu event:
```

```
Registered Node ubuntu in Controller
31m         Normal   Starting                 Node   Starting kube-proxy.
20m         Normal   NodeReady                Node   Node ubuntu status is now:
NodeReady
ubuntu@ip-10-0-2-200:~$
```

While your events will be different you can see the value of the cluster event log. You can display event data associated with a given resource by supplying its name. You can also control the output format.

For example to make the data machine readable you could output it in JSON:

```
ubuntu@ip-10-0-2-200:~$ kubectl get -o json events | tail

            },
            "type": "Normal"
        }
    ],
    "kind": "List",
    "metadata": {
        "resourceVersion": "",
        "selfLink": ""
    }
}
ubuntu@ip-10-0-2-200:~$
```

7. Cleaning Up

Now that we have given our new cluster a good test we can clean up by deleting the service and deployments we have created. The `kubectl delete` subcommand allows you to delete objects you have created in the cluster.

To begin, delete the *my-nginx* Service:

```
ubuntu@ip-10-0-2-200:~$ kubectl get services

NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes          ClusterIP   10.96.0.1       <none>        443/TCP   5h
my-nginx-87464966f   ClusterIP   10.102.46.131   <none>        80/TCP    8m
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl delete service my-nginx

service "my-nginx" deleted
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl get services

NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
```

```
svc/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   2h

ubuntu@ip-10-0-2-200:~$
```

Do not delete the kubernetes service.

Next we can delete the deployments (which in turn removes the associated replica set).

```
ubuntu@ip-10-0-2-200:~$ kubectl get pod

NAME                READY   STATUS    RESTARTS   AGE
my-nginx            1/1     Running   1          17m
mylogtracker        1/1     Running   0          5m31s
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl delete pod my-nginx

pod "my-nginx" deleted
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl delete pod mylogtracker

pod "mylogtracker" deleted
ubuntu@ip-10-0-2-200:~$
```

```
ubuntu@ip-10-0-2-200:~$ kubectl get pods

No resources found.
ubuntu@ip-10-0-2-200:~$
```

You Kubernetes cluster should now be cleaned up and ready for the next lab:

```
ubuntu@ip-10-0-2-200:~$ kubectl get services,deployments,replicasets,pods

NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   34m
user@ubuntu:~$
```

Be sure to leave the `service/kubernetes` service!

When we delete a deployment, Kubernetes ensures that all pods controlled by that replica set are also deleted. If you have trouble with deleting resources, the order matters. Deployments watch replica sets, replica sets watch pods; start with deployments. Services fall outside of that restart logic, and can be deleted at any point.

Congratulations, you have completed the lab!