Cloud Foundry Container Runtime

Lab 10 - 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.

```
user@ubuntu:~$ 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
  run-container  Run a particular image on the cluster. This command is deprecated, use "run" instead

Basic Commands (Intermediate):
  get            Display one or many resources
  explain        Documentation of 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
  rolling-update Perform a rolling update of the given ReplicationController
  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:
  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
  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-versions   Print the supported API versions on the server, in the form of "group/version"
  config         Modify kubeconfig files
  help           Help about any command
  plugin         Runs a command-line plugin
  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).
user@ubuntu:~$
```

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.

```
user@ubuntu:~$ 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 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).
user@ubuntu:~$
```

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

```
user@ubuntu:~$ kubectl config view

apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://172.16.151.229:6443
  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
user@ubuntu:~$
```

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:

```
user@ubuntu:~$ 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:
      --certificate-authority='': Path to certificate-authority file for the cluster entry in kubeconfig
      --embed-certs=false: embed-certs for the cluster entry in kubeconfig
      --insecure-skip-tls-verify=false: insecure-skip-tls-verify for the cluster entry in kubeconfig
      --server='': server 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).
user@ubuntu:~$
```

`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.

```
user@ubuntu:~$ 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
user@ubuntu:~$
```

Display the contents of the config file:

```
user@ubuntu:~$ cat ~/.kube/config

apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:
```

```
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN5RENDQWJDZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFWTVJNd0VRWURWUVFERXdw
cmRXSmwKY201bGRHVnpNQjRYRFRFM01USXdOREU1TVRFd01sb1hEVEkzTVRJd01qRTVNVEV3TWxvd0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJs
Y3pDQ0FTSXdEUVlKS29aSWh2Y05BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBTEI1CjhEdmpESFd0OGxnLy9XR2pqcXl1WFQ5NDFCZ0tBNzg0Z1c4WDZD
QlJDQWNZQVQ1NzdaSTdHVEF3Uks0RGJMdHAKT3NhbFZFeE1KbHVzc3Z1YXAvblBwMFlvS0hTZDBydWRGNVVzWDdUN1NZVjJ3aklOS0JyeTBOUWpUdHJT
OHo4VAp6RjVBMVVLV2lVb2RHZUhHSWtBVkR4V0NabWhRVy80d0xoeUliVDdZcHNsbnFFaStDM1M3eHNnU1hsVTJCWi8yCm8vTldHODYyVFZlMS9IZVFW
NHgyNEJQUnl1Q3RtcjVuUGVaMmxGS3JvaDRJb3B3T0NUYlZXNjl0TjNidjhtYkMKemtNWjdhMWU2SjBJNDNBOENlR3liMUlFZWQ5RndmcU1RVmZnR09z
N1A2RVY3L3YzVEZERDZjeXJhVXFYenNlSgpIQ2x4S29UbmpkdE4rdWJTYWxrQ0F3RUFBYU1qTUNFd0RnWURWUjBQQVFIL0JBUURBZ0trTUE4R0ExVWRF
d0VCCi93UUZNQU1CQWY4d0RRWUpLb1pJaHZjTkFRUxCUUFEZ2dFQkFIcjU4VUwwcFlncVcrQkJEc08vOUNUei9NWkcKdXhhN1BPdEtOVUQ2V09WdWZC
UXRqN2ovR2lVaW5FNGkvTDNjVjJjNnBhYUY2TFZlQWF2VlpseWMvbERTYjZ4RQpoZFRkbFdWOG41WXFzR21pcjE3b0ZhUjlEbi9HNUs5bDlzUmhLWDAz
MUsxMUwwTW1SbnJLZ0llY2p6QUY4MFZmClhiazhpZkdFdC85cnp6OFovWjdibTFhOFRpZ1JHN2ZRVmtZUWdBV2pabjFBWkRkVFlvcUNsZHVTTVRGUkF4
RVQKTnBWZlB4bnBHZzBQU0llZ2xkYkxkOHE4VHBWOU1qUHpFN2tPendlT3ZDQ3hVbjhnLyt0ZHZadi9kZ3MxMmlScwpGL3dJUi9qdWxlakhWRUQyNEo2
UmR0aStJVURZaXZXNDQzZlNxaERpdDI4UlVoVFY5MWxUT3pyM3hRRT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    server: https://172.16.151.229:6443
  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:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM4akNDQWRxZ0F3SUJBZ0lJVjR1Q056a2RSUEV3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJF
R0ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB4TnpFeU1EUXhPVEV4TURKYUZ3MHhPREV5TURReE9URXhNRE5hTURReApGekFWQmdOVkJBb1REbk41
YzNSbGJUcHRZWE4wWlhKek1Sa3dGd1lEVlFRREV4QnJkV0psY201bGRHVnpMV0ZrCmJXbHVNSUlCSWpBTkJna3Foa2lHOXcwQkFRUZBQU9DQVE4QU1J
SUJDZ0tDQVFFQXdzZ0FUQzdLa0szUlhMRWEKeWp1eG43S2pVR21LV3RwNlVTTnJnWEk5WWlkNXdWZWRNMCt2amttSGM3WjgzYTJrUFg1SExGOEdwcEFi
S2FNTgpQSlVCY205ekRKZk10Q1FKWFBhaVluQkFET3hRMTVPL0ZIWGdLTG85ZDh1ay9Mcm5jZFUybnBhclh3dnFmdnlxCitvVVdFQ3kvaTJHL0V0NkRk
NlJUcmIybGk4MDlYSXpQZW4xczQxUkZ1Yko5T01vendBa29FdkJROUR3RC9EUUwKaWxVeGwxOFY5WVhMakVFQmFES2MxR3NSOThqUlZJRmVKT29TaWUx
QmYveTJaMHphVHRHbzhKc3Mrb1JKdmVrbAorMGZLYU1UcGxRYUQrWmR6OFcrZWRwVUZsd1h3RTFTbi9JcDVxNVJMdlh3TVB0Qkx4Tk1GdXc5bjZzcXM0
Mm9vCnB5MlNjd0lEQVFBQm95Y3dKVEFPQmdOVkhROEJBZjhFQkFNQ0JhQXdFd1lEVlIwbEJBd3dDZ1lJS3dZQkJRVUgKQXdJd0RRWUpLb1pJaHZjTkFR
RUxCUUFEZ2dFQkFJQk1qV3lnbUtlVkloMk5MRkxVQnh5UDRYVExwRmZWNnpaeQp2Ylc1SmcxQXNUUGZFQmcxSEhKRk8xb2g2L21rQ05PTnZESXN3bDR2
bi9UWWlzZlJBQ09pUjNHeDkrYnZLcUl4CkYydDltaitnbXFvc0Nqa0ZDZGFudjE3RzNwVFJIcjhrQndrVnpKeStIL1dzWUtpMkIxcHdId3hCazJkcmlX
ZlIKN0ErTFd5VU5uamNnNWdiam83d1FObWIzTG9zMVJMMys4UU1scFlCTjMrdFZVL1pON3l3NG9LVGtFeDNOR1RlSApXaE9kT0xaTzAxSzhiWWtqQ2hh
SXRITDlJa0I4eW5MS0lRcXU2UndYclJobkRrdlF1MDZTY0Q4dW4xS0VkdUs0CkVrMnFmdW0vdHpRN0U2aEFBcEhtMldaLzBxcm5jU0h0Y2U1cE93Q2Rq
MHRJT2s5Y1NRUT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
    client-key-data:
LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcEFJQkFBS0NBUUVBd3NnQVRDN0trSzNSWExFYXlqdXhuN0tqVUdtS1d0cDZVU05yZ1hJ
OVlpZDV3VmVkCk0wK3Zqa21IYzdaODNhMmtQWDVITEY4R3BwQWJLYU1OUEpVQmNtOXpESmZNdENRSlhQYWlZbkJBRE94UTE1Ty8KRkhYZ0tMbzlkOHVr
L0xybmNkVTJucGFyWHd2cWZ2eXErb1VXRUN5L2kyRy9FdDZEZDZSVHJiMmxpODA5WEl6UAplbjFzNDFSRnViSjlPTW96d0Frb0V2QlE5RHdEL0RRTGls
VXhsMThWOVlYTGpFRUJhREtjMUdzUjk4alJWSUZlCkpPb1NpZTFCZi95MlowemFUdEdvOEpzcytvUkp2ZWtsKzBmS2FNVHBsUWFEK1pkejhXK2VkcFVG
bHdYd0UxU24KL0lwNXE1Ukx2WHdNUHRCTHhOTUZ1dzluNnNxczQyb29weTJTY3dJREFRQUJBb0lCQVFDTWFudjBiNkx0NjdCTApQdzJPRHJ4ODROM2s0
VUN4UUdEL1R5WjRHZS93YTM3Vmg2QzIxZEk4SS91MnlpNTM3RzdET2Q1N09VSGxIMnZGCmQxcmFlcHloYjRDajZtYlMwa3ZCcmJiZ2VnVmlpNVczNHpY
VHZTcE1rZjR2OXluSVc4RHZpZ0luRDMwWk50RGsKbktkeGxER1VsWDI4TUVuN2cxZUpEM2lYZm1qeVAwNExFR1BkbFNYRVMxbExlNHN1R0RJUTlVTHVv
anpzRkFqYgpneUdEUzIwVTZ5N2FEc0lZdVdEQ2ZhdEx2Ym5mQy9VZSsvYnN6b1Q3dEdtRzlzYlZXL013S21UQmI0MmlaeFhECnN2RUNXYlJ2c2FPaVBm
ckJ2bk41a3M4enhRQkllWlVYRnlzRmUxc09kb3Q0Tms5Z29MU2RZRklCaDBRV3drKy8KZnRUSGZUa0JBb0dCQU1oWGI0cG8xZGhubUw1VnpxWlZIbUFS
cEU3STRGR1ZHY0FlVXNCczUvaFI5UGhhNE5yKwp6MlFWSktBK05ZYlJYZThPdWxtdzRzeUpaUFN2RVBFYjlmVG1pY091clZoMHBCSHlzYnZEY0R5Tjdh
clNLajMzCksyQzVObGhFTFo2YmJETzlLamU0cFNWazZwUTdVeHlQUHJyRE4rdGhvRmNDdWRsTzBlcFhRQUtCQW9HQkFQamwKSENZR3NyMklMd3RFb1pW
Qy9MS3orU2t1WU0wekRkZkhtVmNXY1VUem90bG5uVWhLN2FvYzZscjRjVEYxTkZBSQpBc0NMS3k3elJzY05LYlM1UFdwQ1VMTjlMSVhHZEsrSGFtT1FB
N0NCSUJBK1V4SHU0eHA4WEZFNGRKR3dlUThHCmoyM1p0aXlJSjJQUjd1VVUvNDBWTWI5cjJEdEk4b3pid09pQjdqTHpBb0dCQUxKdExxPb1lkRnhINThH
Y3FtOTUKWDVRd0lpWUl3Yi9uQ3dnUCthTm5XekhOekY5a2tONTZFanNRbVk0ZDJZNDQ2Vy8vcmZnemtjcVlrMUZZbWI0agpLT3IwWjczZjFJThpYeXlj
K2E1QVliV01zaDl4RGk0aDlJQXdkRFlvZ25pLzg1NGcyM2pFK2xCVGtKaDBQWkV5CkFxeFRNWHB4ZUZ3R0VYOTRzM3dDT1FBQkFvR0FNS0ZyVjQ2MWU4
elJERTJUbUx0bTdtKzF2ak1lbk5sZDJneFkKektoSTUyVUhLNTFRSU9EckFQTDNZMkRwbFBWR2pHQ1VVUlNnRW1OY0wrWkZnTmMweGI5QlQyQ2t3MXFj
VC9PUQplUFdaa3ZJWDFyU212SGxGakZaQ0gyaDlkajNaMlhLNXNZZjVUVWdwRWhyaHA0YnJ5NkFaZ1VKTUZJRlRTdXhoCkM4emZRcnNDZ1lCTExVTjRi
M0xFakVORyt5K2tFcTFmcEp4MDJhNk4zeUN5SmFyNWIzbWVjS0YvMG5nTm9PZlcKUFkRWE3ZzFDaVVkMVJibGtZZHhxdFNiYUZEUzVERDI5MHVLOEZj
WEVRdWRlcEJ1YklDQ0ozc3ZWdndIRElHegphdm8zc3lSd3lVdGt0MmZmYkorbHJETE8zTmg3RnJiS3NOWG5zTjgrYjVBeDBZSHhkMXRsUUE9PQotLS0t
LUVORCBSU0EgUFJJVkFURSBLRVktLS0tLQo=
user@ubuntu:~$
```

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.

```
user@ubuntu:~$ kubectl cluster-info

Kubernetes master is running at https://172.16.151.229:6443
KubeDNS is running at https://172.16.151.229:6443/api/v1/namespaces/kube-system/services/kube-dns/proxy
```

```
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
user@ubuntu:~$
```

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

```
user@ubuntu:~$ kubectl cluster-info dump |& wc -l

5215
user@ubuntu:~$
```

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

```
user@ubuntu:~$ kubectl describe node ubuntu

Name:               ubuntu
Roles:              master
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/hostname=ubuntu
                    node-role.kubernetes.io/master=
Annotations:        node.alpha.kubernetes.io/ttl=0
                    volumes.kubernetes.io/controller-managed-attach-detach=true
CreationTimestamp:  Wed, 28 Mar 2018 10:14:15 -0700
Taints:             <none>
Unschedulable:      false
Conditions:
  Type             Status  LastHeartbeatTime                 LastTransitionTime                Reason
Message
  ----             ------  -----------------                 ------------------                ------
-------
  OutOfDisk        False   Wed, 28 Mar 2018 11:44:40 -0700   Wed, 28 Mar 2018 10:14:12 -0700
KubeletHasSufficientDisk     kubelet has sufficient disk space available
  MemoryPressure   False   Wed, 28 Mar 2018 11:44:40 -0700   Wed, 28 Mar 2018 10:14:12 -0700
KubeletHasSufficientMemory   kubelet has sufficient memory available
  DiskPressure     False   Wed, 28 Mar 2018 11:44:40 -0700   Wed, 28 Mar 2018 10:14:12 -0700
KubeletHasNoDiskPressure     kubelet has no disk pressure
  PIDPressure      False   Wed, 28 Mar 2018 11:44:40 -0700   Wed, 28 Mar 2018 10:14:12 -0700
KubeletHasSufficientPID      kubelet has sufficient PID available
  Ready            True    Wed, 28 Mar 2018 11:44:40 -0700   Wed, 28 Mar 2018 11:07:18 -0700   KubeletReady
kubelet is posting ready status. AppArmor enabled
Addresses:
  InternalIP:  192.168.225.210
  Hostname:    ubuntu
Capacity:
 cpu:                2
 ephemeral-storage:  18447100Ki
 hugepages-1Gi:      0
 hugepages-2Mi:      0
 memory:             2029876Ki
 pods:               110
Allocatable:
 cpu:                2
 ephemeral-storage:  17000847332
 hugepages-1Gi:      0
 hugepages-2Mi:      0
 memory:             1927476Ki
 pods:               110
System Info:
 Machine ID:                 6e883acc04fc7db3713776be57a3dac9
 System UUID:                B2344D56-FE13-D736-51F0-31DF1E2E8A04
 Boot ID:                    9eda422d-14ef-43aa-94d7-5da65fbc2dfa
 Kernel Version:             4.4.0-116-generic
 OS Image:                   Ubuntu 16.04.1 LTS
 Operating System:           linux
 Architecture:               amd64
 Container Runtime Version:  docker://18.3.0
 Kubelet Version:            v1.10.0
 Kube-Proxy Version:         v1.10.0
ExternalID:                  ubuntu
Non-terminated Pods:         (7 in total)
  Namespace                  Name                              CPU Requests  CPU Limits  Memory Requests  Memory
Limits
  ---------                  ----                              ------------  ----------  ---------------  ----------
---
```

```
  kube-system                etcd-ubuntu                       0 (0%)        0 (0%)      0 (0%)           0 (0%)
  kube-system                kube-apiserver-ubuntu             250m (12%)    0 (0%)      0 (0%)           0 (0%)
  kube-system                kube-controller-manager-ubuntu    200m (10%)    0 (0%)      0 (0%)           0 (0%)
  kube-system                kube-dns-86f4d74b45-dz9gl         260m (13%)    0 (0%)      110Mi (5%)       170Mi (9%)
  kube-system                kube-proxy-n9x4z                  0 (0%)        0 (0%)      0 (0%)           0 (0%)
  kube-system                kube-scheduler-ubuntu             100m (5%)     0 (0%)      0 (0%)           0 (0%)
  kube-system                weave-net-kq9db                   20m (1%)      0 (0%)      0 (0%)           0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests  CPU Limits  Memory Requests  Memory Limits
  ------------  ----------  ---------------  -------------
  830m (41%)    0 (0%)      110Mi (5%)       170Mi (9%)
Events:
  Type    Reason     Age   From             Message
  ----    ------     ----  ----             -------
  Normal  NodeReady  37m   kubelet, ubuntu  Node ubuntu status is now: NodeReady
user@ubuntu:~$
```

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.

```
user@ubuntu:~$ kubectl version

Client Version: version.Info{Major:"1", Minor:"10", GitVersion:"v1.10.0",
GitCommit:"fc32d2f3698e36b93322a3465f63a14e9f0eaead", GitTreeState:"clean", BuildDate:"2018-03-26T16:55:54Z",
GoVersion:"go1.9.3", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"10", GitVersion:"v1.10.0",
GitCommit:"fc32d2f3698e36b93322a3465f63a14e9f0eaead", GitTreeState:"clean", BuildDate:"2018-03-26T16:44:10Z",
GoVersion:"go1.9.3", Compiler:"gc", Platform:"linux/amd64"}
user@ubuntu:~$
```

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.

```
user@ubuntu:~$ kubectl get

Desktop/                  .kube/                    Public/
...

user@ubuntu:~$ kubectl get
```

You can enable temporary kubectl bash completion with:

```
user@ubuntu:~$ source <(kubectl completion bash)

user@ubuntu:~$
```

And after.

```
user@ubuntu:~$ kubectl get

certificatesigningrequest  deployment                networkpolicy             replicaset
statefulset
...

user@ubuntu:~$ kubectl get
```

That is much better!

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

```
user@ubuntu:~$ kubectl get service,deployments,rs,pods

NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
svc/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   1h
user@ubuntu:~$
```

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.

```
user@ubuntu:~$ kubectl get service,deployments,rs,pods --namespace=kube-system

NAME       TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP   1h

NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kube-dns   1         1         1            1           1h

NAME                  DESIRED   CURRENT   READY     AGE
kube-dns-86f4d74b45   1         1         1         1h

NAME                              READY     STATUS    RESTARTS   AGE
etcd-ubuntu                       1/1       Running   0          1h
kube-apiserver-ubuntu             1/1       Running   0          1h
kube-controller-manager-ubuntu    1/1       Running   0          1h
kube-dns-86f4d74b45-dz9gl         3/3       Running   0          1h
kube-proxy-n9x4z                  1/1       Running   0          1h
kube-scheduler-ubuntu             1/1       Running   0          1h
weave-net-kq9db                   2/2       Running   0          48m
user@ubuntu:~$
```

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

To test our cluster lets run a single container pod with a replication factor of 2. 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.

```
user@ubuntu:~$ kubectl run my-nginx --image=nginx:1.11 --replicas=2 --port=80

deployment.apps "my-nginx" created
user@ubuntu:~$
```

This creates two copies (replicas) of a pod running nginx. The deployment 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 deployments running on the cluster:

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

NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-nginx   2         2         2            0           7s
user@ubuntu:~$
```

This shows that our pods are deployed and up to date. It may take a bit to pull the Docker images (Available might be 0). Let's look at the pod listing from the Kubernetes side:

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

NAME                        READY     STATUS    RESTARTS   AGE
my-nginx-87464966f-7c6xj    1/1       Running   0          9s
my-nginx-87464966f-kdqqz    1/1       Running   0          9s
user@ubuntu:~$
```

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

```
user@ubuntu:~$ docker container ls --filter "name=nginx"

CONTAINER ID        IMAGE                     COMMAND                  CREATED             STATUS
PORTS               NAMES
```

```
d91b69f1cb49        5766334bdaa0                "nginx -g 'daemon of…"   32 seconds ago      Up 32 seconds
k8s_my-nginx_my-nginx-87464966f-7c6xj_default_c00d0c6e-32d6-11e8-885a-000c292e8a04_0
292e9faf782e        5766334bdaa0                "nginx -g 'daemon of…"   32 seconds ago      Up 32 seconds
k8s_my-nginx_my-nginx-87464966f-kdqqz_default_c00da53b-32d6-11e8-885a-000c292e8a04_0
3cd2b943db6b        k8s.gcr.io/pause-amd64:3.1   "/pause"                 33 seconds ago      Up 32 seconds
k8s_POD_my-nginx-87464966f-kdqqz_default_c00da53b-32d6-11e8-885a-000c292e8a04_0
3362db7d71a2        k8s.gcr.io/pause-amd64:3.1   "/pause"                 33 seconds ago      Up 32 seconds
k8s_POD_my-nginx-87464966f-7c6xj_default_c00d0c6e-32d6-11e8-885a-000c292e8a04_
user@ubuntu:~$
```

As you can see, while our *run* subcommand requested that Kubernetes run nginx with 2 replicas, 4 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 "gcr.io/google_containers/pause-amd64:3.1" 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 4 containers, each 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:

```
user@ubuntu:~$ docker container ls --filter "name=nginx" --format "{{.Names}}"

k8s_my-nginx_my-nginx-87464966f-7c6xj_default_c00d0c6e-32d6-11e8-885a-000c292e8a04_0
k8s_my-nginx_my-nginx-87464966f-kdqqz_default_c00da53b-32d6-11e8-885a-000c292e8a04_0
k8s_POD_my-nginx-87464966f-kdqqz_default_c00da53b-32d6-11e8-885a-000c292e8a04_0
k8s_POD_my-nginx-87464966f-7c6xj_default_c00d0c6e-32d6-11e8-885a-000c292e8a04_0
user@ubuntu:~$
```

Next let's ask Kubernetes to display the replica sets at work:

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

NAME                 DESIRED   CURRENT   READY     AGE
my-nginx-87464966f   2         2         2         7m
user@ubuntu:~$
```

As you can see, the deployment generated a replica set to create and manage our two pods. The replica set has the name we provided on the run command line above (plus numerical extension) and shows the number of replicas of the pod it is maintaining.

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

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

d91b69f1cb49
user@ubuntu:~$
```

```
user@ubuntu:~$ docker container ls --filter "name=nginx"

CONTAINER ID        IMAGE                       COMMAND                  CREATED              STATUS
PORTS               NAMES
244e9e3849af        5766334bdaa0                "nginx -g 'daemon of…"   22 seconds ago       Up 21 seconds
k8s_my-nginx_my-nginx-87464966f-7c6xj_default_c00d0c6e-32d6-11e8-885a-000c292e8a04_1
292e9faf782e        5766334bdaa0                "nginx -g 'daemon of…"   About a minute ago   Up About a minute
k8s_my-nginx_my-nginx-87464966f-kdqqz_default_c00da53b-32d6-11e8-885a-000c292e8a04_0
3cd2b943db6b        k8s.gcr.io/pause-amd64:3.1   "/pause"                 2 minutes ago        Up About a minute
k8s_POD_my-nginx-87464966f-kdqqz_default_c00da53b-32d6-11e8-885a-000c292e8a04_0
3362db7d71a2        k8s.gcr.io/pause-amd64:3.1   "/pause"                 2 minutes ago        Up About a minute
k8s_POD_my-nginx-87464966f-7c6xj_default_c00d0c6e-32d6-11e8-885a-000c292e8a04_0
user@ubuntu:~$
```

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.

```
user@ubuntu:~$ docker container ls -a --filter "name=nginx"

CONTAINER ID        IMAGE                     COMMAND                  CREATED             STATUS
PORTS               NAMES
244e9e3849af        5766334bdaa0              "nginx -g 'daemon of…"   43 seconds ago      Up 42 seconds
k8s_my-nginx_my-nginx-87464966f-7c6xj_default_c00d0c6e-32d6-11e8-885a-000c292e8a04_1
d91b69f1cb49        5766334bdaa0              "nginx -g 'daemon of…"   2 minutes ago       Exited (137) 43
seconds ago                       k8s_my-nginx_my-nginx-87464966f-7c6xj_default_c00d0c6e-32d6-11e8-885a-
000c292e8a04_0
292e9faf782e        5766334bdaa0              "nginx -g 'daemon of…"   2 minutes ago       Up 2 minutes
k8s_my-nginx_my-nginx-87464966f-kdqqz_default_c00da53b-32d6-11e8-885a-000c292e8a04_0
3cd2b943db6b        k8s.gcr.io/pause-amd64:3.1   "/pause"                 2 minutes ago       Up 2 minutes
k8s_POD_my-nginx-87464966f-kdqqz_default_c00da53b-32d6-11e8-885a-000c292e8a04_0
3362db7d71a2        k8s.gcr.io/pause-amd64:3.1   "/pause"                 2 minutes ago       Up 2 minutes
k8s_POD_my-nginx-87464966f-7c6xj_default_c00d0c6e-32d6-11e8-885a-000c292e8a04_0
user@ubuntu:~$
```

Notice that we killed container *d91b69f1cb49* in the example but the new container *244e9e3849af* 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 it is replicated and 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 dynamically replicated set of pods as a Service. The `kubectl` expose command does this.

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

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

NAME                  DESIRED   CURRENT   READY     AGE
my-nginx-87464966f   2         2         2         8m
user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl expose $(kubectl get rs -o=name) --port=80

service "my-nginx-87464966f" exposed
user@ubuntu:~$
```

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.

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

NAME                  TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes            ClusterIP   10.96.0.1      <none>        443/TCP   2h
my-nginx-7cb86fff46   ClusterIP   10.102.46.131  <none>        80/TCP    21s
user@ubuntu:~$
```

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:

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

user@ubuntu:~$ echo $NX_CIP
10.102.46.131

user@ubuntu:~$ 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
user@ubuntu:~$
```

Which pod did you connect to? Does it matter?

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.

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

NAME                        READY     STATUS    RESTARTS   AGE
my-nginx-87464966f-7c6xj    1/1       Running   1          9m
my-nginx-87464966f-kdqqz    1/1       Running   0          9m
user@ubuntu:~$
```

Here we select the first one listed.

```
user@ubuntu:~$ NX_POD_NAME=$(kubectl get pods --no-headers | awk '{print $1}' | head -1)

user@ubuntu:~$ echo $NX_POD_NAME

my-nginx-87464966f-7c6xj
user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl exec $NX_POD_NAME 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/"
user@ubuntu:~$
```

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:

```
user@ubuntu:~$ kubectl get pod $NX_POD_NAME -o json | jq ".spec.containers[].name" -r

my-nginx
user@ubuntu:~$
```

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

```
user@ubuntu:~$ kubectl exec -c my-nginx $NX_POD_NAME 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/"
user@ubuntu:~$
```

Do not forget about tab completion!

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`)

```
user@ubuntu:~$ journalctl -n 400 --no-pager -u kubelet.service | grep -v "no observation"

-- Logs begin at Mon 2017-12-04 02:29:14 PST, end at Mon 2017-12-04 13:27:39 PST. --
Dec 04 13:04:27 ubuntu kubelet[74364]: E1204 13:04:27.551559   74364 helpers.go:468] PercpuUsage had 0 cpus, but the
actual number is 2; ignoring extra CPUs
Dec 04 13:04:27 ubuntu kubelet[74364]: E1204 13:04:27.566946   74364 helpers.go:468] PercpuUsage had 0 cpus, but the
actual number is 2; ignoring extra CPUs
Dec 04 13:04:27 ubuntu kubelet[74364]: I1204 13:04:27.609826   74364 reconciler.go:212]
operationExecutor.VerifyControllerAttachedVolume started for volume "default-token-2kmhb" (UniqueName:
"kubernetes.io/secret/b6bc1fa0-d936-11e7-a277-000c29ae8ddc-default-token-2kmhb") pod "my-nginx-7cb86fff46-b47g4"
(UID: "b6bc1fa0-d936-11e7-a277-000c29ae8ddc")
Dec 04 13:04:27 ubuntu kubelet[74364]: I1204 13:04:27.710464   74364 reconciler.go:212]
operationExecutor.VerifyControllerAttachedVolume started for volume "default-token-2kmhb" (UniqueName:
"kubernetes.io/secret/b6be06ff-d936-11e7-a277-000c29ae8ddc-default-token-2kmhb") pod "my-nginx-7cb86fff46-pl8dc"
(UID: "b6be06ff-d936-11e7-a277-000c29ae8ddc")
Dec 04 13:04:28 ubuntu kubelet[74364]: W1204 13:04:28.195920   74364 pod_container_deletor.go:77] Container
"9b38a47a31f8755eeb5c75e6b6c75fb735070adc862a6e76d80621f723011cc4" not found in pod's containers
Dec 04 13:04:28 ubuntu kubelet[74364]: W1204 13:04:28.228414   74364 pod_container_deletor.go:77] Container
"cf955e40c16f5f60d83b5663bfe5292c19f20297df543efed51e051c1c18db7c" not found in pod's containers
Dec 04 13:04:47 ubuntu kubelet[74364]: W1204 13:04:47.165440   74364 conversion.go:110] Could not get instant cpu
stats: different number of cpus
Dec 04 13:04:47 ubuntu kubelet[74364]: W1204 13:04:47.166876   74364 conversion.go:110] Could not get instant cpu
stats: different number of cpus
Dec 04 13:08:21 ubuntu kubelet[74364]: I1204 13:08:21.173299   74364 kuberuntime_manager.go:500] Container {Name:my-
nginx Image:nginx:1.11 Command:[] Args:[] WorkingDir: Ports:[{Name: HostPort:0 ContainerPort:80 Protocol:TCP
HostIP:}] EnvFrom:[] Env:[] Resources:{Limits:map[] Requests:map[]} VolumeMounts:[{Name:default-token-2kmhb
ReadOnly:true MountPath:/var/run/secrets/kubernetes.io/serviceaccount SubPath: MountPropagation:<nil>}]
LivenessProbe:nil ReadinessProbe:nil Lifecycle:nil TerminationMessagePath:/dev/termination-log
TerminationMessagePolicy:File ImagePullPolicy:IfNotPresent SecurityContext:nil Stdin:false StdinOnce:false
TTY:false} is dead, but RestartPolicy says that we should restart it.
Dec 04 13:08:21 ubuntu kubelet[74364]: I1204 13:08:21.173483   74364 kuberuntime_manager.go:739] checking backoff
for container "my-nginx" in pod "my-nginx-7cb86fff46-pl8dc_default(b6be06ff-d936-11e7-a277-000c29ae8ddc)"
user@ubuntu:~$
```

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:

```
user@ubuntu:~$ kubectl get pods --namespace=kube-system

NAME                                READY     STATUS    RESTARTS   AGE
etcd-ubuntu                         1/1       Running   0          2h
kube-apiserver-ubuntu               1/1       Running   0          2h
kube-controller-manager-ubuntu      1/1       Running   0          2h
kube-dns-545bc4bfd4-957v6           3/3       Running   0          2h
kube-proxy-l9jv6                    1/1       Running   0          2h
kube-scheduler-ubuntu               1/1       Running   0          2h
kubernetes-dashboard-7486b894c6-4bp24   1/1       Running   1          1h
weave-net-q2gds                     2/2       Running   0          1h
user@ubuntu:~$
```

Now display the last 10 lines from the API service:

```
user@ubuntu:~$ kubectl logs --namespace=kube-system --tail=10 kube-apiserver-ubuntu

I1204 19:12:25.744624       1 storage_rbac.go:257] created role.rbac.authorization.k8s.io/system::leader-locking-
kube-scheduler in kube-system
I1204 19:12:25.785495       1 storage_rbac.go:257] created
role.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-public
I1204 19:12:25.825925       1 storage_rbac.go:287] created rolebinding.rbac.authorization.k8s.io/system::leader-
locking-kube-controller-manager in kube-system
I1204 19:12:25.864200       1 storage_rbac.go:287] created rolebinding.rbac.authorization.k8s.io/system::leader-
locking-kube-scheduler in kube-system
I1204 19:12:25.904522       1 storage_rbac.go:287] created
rolebinding.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-system
```

```
I1204 19:12:25.945576       1 storage_rbac.go:287] created
rolebinding.rbac.authorization.k8s.io/system:controller:cloud-provider in kube-system
I1204 19:12:25.986616       1 storage_rbac.go:287] created
rolebinding.rbac.authorization.k8s.io/system:controller:token-cleaner in kube-system
I1204 19:12:26.026056       1 storage_rbac.go:287] created
rolebinding.rbac.authorization.k8s.io/system:controller:bootstrap-signer in kube-public
E1204 21:25:32.974602       1 upgradeaware.go:310] Error proxying data from client to backend: write tcp
172.16.151.229:52660->172.16.151.229:10250: write: broken pipe
E1204 21:26:16.015490       1 upgradeaware.go:310] Error proxying data from client to backend: write tcp
172.16.151.229:52770->172.16.151.229:10250: write: broken pipe
user@ubuntu:~$
```

The log displays the timestamp of each entry, the API call being made and the service making the call.

- Examine the API log to see how often the kubelet reports in.

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.

```
user@ubuntu:~$ kubectl run mylogtracker --image nginx:1.11

deployment.apps "mylogtracker" created
user@ubuntu:~$
```

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

```
user@ubuntu:~$ kubectl get pod --namespace=kube-system

NAME                                    READY     STATUS    RESTARTS   AGE
etcd-ubuntu                             1/1       Running   0          2h
kube-apiserver-ubuntu                   1/1       Running   0          2h
kube-controller-manager-ubuntu          1/1       Running   0          2h
kube-dns-545bc4bfd4-957v6               3/3       Running   0          2h
kube-proxy-l9jv6                        1/1       Running   0          2h
kube-scheduler-ubuntu                   1/1       Running   0          2h
kubernetes-dashboard-7486b894c6-4bp24   1/1       Running   1          1h
weave-net-q2gds                         2/2       Running   0          1h
user@ubuntu:~$
```

Try the controller manager server first:

```
user@ubuntu:~$ kubectl logs --namespace=kube-system kube-controller-manager-ubuntu | grep mylogtracker

I1204 21:28:34.996978       1 event.go:218] Event(v1.ObjectReference{Kind:"Deployment", Namespace:"default",
Name:"mylogtracker", UID:"157993cf-d93a-11e7-a277-000c29ae8ddc", APIVersion:"extensions", ResourceVersion:"9571",
FieldPath:""}): type: 'Normal' reason: 'ScalingReplicaSet' Scaled up replica set mylogtracker-6ff4ff6fd5 to 1
I1204 21:28:35.046040       1 event.go:218] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"default",
Name:"mylogtracker-6ff4ff6fd5", UID:"157bb1bc-d93a-11e7-a277-000c29ae8ddc", APIVersion:"extensions",
ResourceVersion:"9572", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: mylogtracker-
6ff4ff6fd5-k5qrq
user@ubuntu:~$
```

You can track the interaction with the deployment, replica set, and pods through the API calls.

- How long does it take for the pod to become fully operational?

Now take a look at the kubelet log:

```
user@ubuntu:~$ journalctl -u kubelet.service | grep mylogtracker

Dec 04 13:28:35 ubuntu 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")
user@ubuntu:~$
```

The first occurrence of our mylogtracker key shows the kubelet discovering the new pod during the `kubelet` sync loop with the API server (which you previously discovered runs every 10 seconds by default).

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

Try getting events with `kubectl`:

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

```
LAST SEEN   FIRST SEEN   COUNT     NAME                                              KIND         SUBOBJECT
TYPE      REASON                  SOURCE                  MESSAGE
14m         14m          1         my-nginx-87464966f-7c6xj.152035d367cce61b        Pod
Normal    Scheduled               default-scheduler       Successfully assigned my-nginx-87464966f-7c6xj to ubuntu
14m         14m          1         my-nginx-87464966f-7c6xj.152035d37669e397        Pod
Normal    SuccessfulMountVolume   kubelet, ubuntu         MountVolume.SetUp succeeded for volume "default-token-
gggvw"
12m         14m          2         my-nginx-87464966f-7c6xj.152035d3a3239349        Pod          spec.containers{my-
nginx}       Normal    Pulled                  kubelet, ubuntu         Container image "nginx:1.11" already present
on machine
12m         14m          2         my-nginx-87464966f-7c6xj.152035d3a9a03fb4        Pod          spec.containers{my-
nginx}       Normal    Created                 kubelet, ubuntu         Created container
12m         14m          2         my-nginx-87464966f-7c6xj.152035d3b39fa51b        Pod          spec.containers{my-
nginx}       Normal    Started                 kubelet, ubuntu         Started container
14m         14m          1         my-nginx-87464966f-gkqvg.152035cd1356f265        Pod          spec.containers{my-
nginx}       Normal    Killing                 kubelet, ubuntu         Killing container with id docker://my-
nginx:Need to kill Pod
14m         14m          1         my-nginx-87464966f-kdqqz.152035d36b657a88        Pod
Normal    Scheduled               default-scheduler       Successfully assigned my-nginx-87464966f-kdqqz to ubuntu
14m         14m          1         my-nginx-87464966f-kdqqz.152035d37658cf0c        Pod
Normal    SuccessfulMountVolume   kubelet, ubuntu         MountVolume.SetUp succeeded for volume "default-token-
gggvw"
14m         14m          1         my-nginx-87464966f-kdqqz.152035d39ea7ffff        Pod          spec.containers{my-
nginx}       Normal    Pulled                  kubelet, ubuntu         Container image "nginx:1.11" already present
on machine
14m         14m          1         my-nginx-87464966f-kdqqz.152035d3a1f0507e        Pod          spec.containers{my-
nginx}       Normal    Created                 kubelet, ubuntu         Created container
14m         14m          1         my-nginx-87464966f-kdqqz.152035d3aee00023        Pod          spec.containers{my-
nginx}       Normal    Started                 kubelet, ubuntu         Started container
14m         14m          1         my-nginx-87464966f-tbpk7.152035cd113d702a        Pod          spec.containers{my-
nginx}       Normal    Killing                 kubelet, ubuntu         Killing container with id docker://my-
nginx:Need to kill Pod
14m         14m          1         my-nginx-87464966f.152035cd06b132bb              ReplicaSet
Normal    SuccessfulDelete        replicaset-controller   Deleted pod: my-nginx-87464966f-tbpk7
14m         14m          1         my-nginx-87464966f.152035cd0855e0b7              ReplicaSet
Normal    SuccessfulDelete        replicaset-controller   Deleted pod: my-nginx-87464966f-gkqvg
14m         14m          1         my-nginx-87464966f.152035d366ccaa19              ReplicaSet
Normal    SuccessfulCreate        replicaset-controller   Created pod: my-nginx-87464966f-7c6xj
14m         14m          1         my-nginx-87464966f.152035d367d79e18              ReplicaSet
Normal    SuccessfulCreate        replicaset-controller   Created pod: my-nginx-87464966f-kdqqz
14m         14m          1         my-nginx.152035cd05b19916                        Deployment
Normal    ScalingReplicaSet       deployment-controller   Scaled down replica set my-nginx-87464966f to 0
14m         14m          1         my-nginx.152035d364e263a4                        Deployment
Normal    ScalingReplicaSet       deployment-controller   Scaled up replica set my-nginx-87464966f to 2
1m          1m           1         mylogtracker-55d4cf8dfb-l676w.15203685a3eb9eb0   Pod
Normal    Scheduled               default-scheduler       Successfully assigned mylogtracker-55d4cf8dfb-l676w to
ubuntu
1m          1m           1         mylogtracker-55d4cf8dfb-l676w.15203685b130b0ae   Pod
Normal    SuccessfulMountVolume   kubelet, ubuntu         MountVolume.SetUp succeeded for volume "default-token-
gggvw"
1m          1m           1         mylogtracker-55d4cf8dfb-l676w.15203685cf91077b   Pod
spec.containers{mylogtracker}   Normal    Pulled                  kubelet, ubuntu         Container image
"nginx:1.11" already present on machine
1m          1m           1         mylogtracker-55d4cf8dfb-l676w.15203685d2a89a10   Pod
spec.containers{mylogtracker}   Normal    Created                 kubelet, ubuntu         Created container
1m          1m           1         mylogtracker-55d4cf8dfb-l676w.15203685ded39d14   Pod
spec.containers{mylogtracker}   Normal    Started                 kubelet, ubuntu         Started container
1m          1m           1         mylogtracker-55d4cf8dfb.15203685a24fbee5         ReplicaSet
Normal    SuccessfulCreate        replicaset-controller   Created pod: mylogtracker-55d4cf8dfb-l676w
1m          1m           1         mylogtracker.15203685a074f9a6                    Deployment
Normal    ScalingReplicaSet       deployment-controller   Scaled up replica set mylogtracker-55d4cf8dfb to 1
user@ubuntu:~$
```

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:

```
user@ubuntu:~$ kubectl get -o json events | tail

            },
            "type": "Normal"
        }
    ],
    "kind": "List",
```

```
    "metadata": {
        "resourceVersion": "",
        "selfLink": ""
    }
}
user@ubuntu:~$
```

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:

```
user@ubuntu:~$ 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
user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl delete service my-nginx-87464966f

service "my-nginx-87464966f" deleted
user@ubuntu:~$
```

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

NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
svc/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   2h

user@ubuntu:~$
```

Do not delete the kubernetes service.

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

```
user@ubuntu:~$ kubectl get deployments

NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
my-nginx       2         2         2            2           23m
mylogtracker   1         1         1            1           4m
user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl delete deployment my-nginx

deployment.extensions "my-nginx" deleted
user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl delete deployment mylogtracker

deployment.extensions "mylogtracker" deleted
user@ubuntu:~$
```

```
user@ubuntu:~$ kubectl get deployments

No resources found.
user@ubuntu:~$
```

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

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

NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
svc/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   2h
user@ubuntu:~$
```

Be sure to leave the `svc/kubernetes` services.

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 Kubernetes exploration lab!