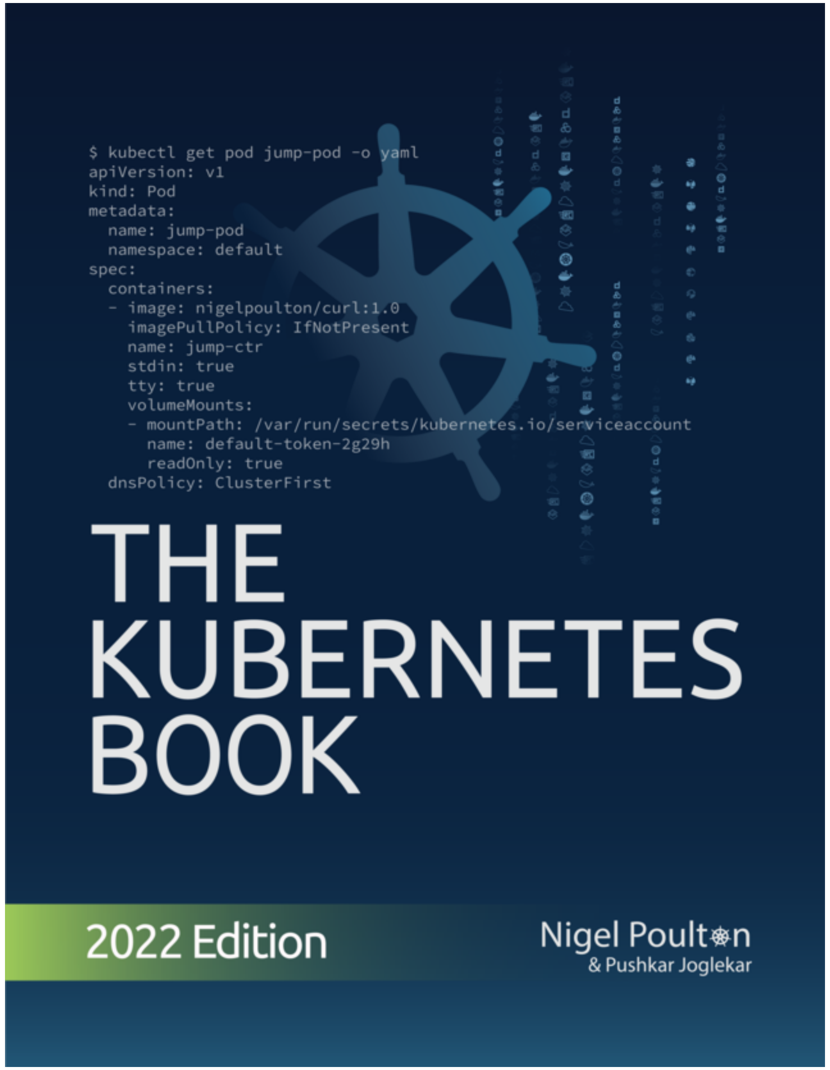

```
$ kubectl get pod jump-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  name: jump-pod
  namespace: default
spec:
  containers:
  - image: nigelpoulton/curl:1.0
    imagePullPolicy: IfNotPresent
    name: jump-ctr
    stdin: true
    tty: true
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-2g29h
      readOnly: true
  dnsPolicy: ClusterFirst
```

theme: gaia

_class: lead

paginate: true

backgroundColor: #fff

[//]: # (Slide author: Fred Gras)

The Kubernetes Book by Nigel Poulton *2021 Edition*

Chapter 4, Working w/ Pods

Feb 10, 2022 Austin OWASP Study Group

What is Pod -- the atomic unit of scheduling

You cannot run a "container" directly in Kubernetes, needs to be wrapped in **Pod**

- Pods augment containers in the following ways
  - Labels and annotations
  - Restart policies
  - Probes (ie: startup, readiness, liveness, and more)
  - Affinity and anti-affinity rules
  - Termination controls
  - Resource sharing (requests and w/ limits)
- Pods can be scheduled
- Pods enable resource sharing

Example of: $ kubectl explain pods --recursive

```
KIND:        Pod
VERSION:     v1

DESCRITPION:
    Pod is a collection of containers that can be run on a host.
    This resource is created by clients and scheduled onto hosts.

FIELDS:
    apiVersion       <string>
    kind             <string>
    metadata         <Object>
        annotations  <map[string]string>
        labels       <map[string]string>
        name         <string>
        namespace    <string>
etc...
```

- notes: see example in two slides

Example of: $ kubectl explain spec.restartPolicy

KIND: Pod

VERSION: v1

FIELD: restartPolicy <string>

DESCRIPTION:

Restart policy for all containers within the pod.

One of **Always, OnFailure, Never.**

Default to **Always**.

More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/

Example of: labels, pods, affinities, policies, and dependencies

```
kind:       Pod
apiVersion: v1
metadata:            // this is the wrapper
    name:   hello-pod
    labels:
        zone:     prod
        version:  v1
spec:
    containers:       // this is a container (that is wrapped)
    - name: hello-ctr
      image: nigelpoulton/k8sbook:1.0
      ports:
      - containerPort: 8080
```

- this pod is a wrapper around one or more containers
- You will see this again many more times :)

Pods enable resource sharing

- Shared filesystem
- Shared network stack (IP address , ports ...)
- Shared memory
- Shared volumes

note: ... will cover these details throught the book (ie: not here :)

Static vs controller Pods

Two ways to deploy pods

- Directly w/ a Pod manifest **Called static**
- Indirectly w/ a controller

Pods compared to cattle, when they die, they get replaced (ie: not a pet :)

Single-container or multi-container Pods

(see later description)

Deploying Pods

steps as follows:

- Define it in a YAML manifest file
- Post the YAML to the API server
- The API server authenticates and authorizes the request
- The configuration (YAML) is validated
- The scheduler deploys the Pod to a healthy node with enough available resources
- The load kubelet monitors it

TODO: find Reference to confirm same steps

the Guts of a Pod

a Pod is actually a special type of container called a pause container

pods are a collection of resources

these resources are actual Linux kernel namespaces, and have following

- **net namespace:** IP address, port range, routing tables...
- **pid namespace:** isolated process tree
- **UTS namespace:** Hostname
- **IPC namespace:** Unix domain sockets and shared memory

(not shown) Diagram showing a **private** pod network, do not need port mapping
similar image

Atomic deployment of Pods -- Meaning all-or-nothing

Pod lifecycle -- pending then running, then terminate after succeded state

Short or long-lived Pods

Pod immutability -- you can't modify them after deployed

Pods are ideal for scaling -- called horizontal scaling

Summary about pods before moving on

- Pods are an atomic unit of scheduling
- single pods are simple, multi-container pods ideal (and fundamental) for tightly coupled workloads
- Pods are scheduled on physical nodes which do not span multiple nodes
- Pods are declared in a **manifest file** you post to the API server
- Pods deployed by higher-level controllers

note: leads into multi-container Pods

Multi-container Pods -- each pod should have a clear defined responsibility

sometimes important to couple two or more functions

co-locating (in the same Pod) allows containers to be designed for single responsibility such as:

- **Sidecar** pattern -- the secondary task augments the main application container
- **Adapter** pattern -- the secondary takes main container output and reformats for external system
- **Ambassador** pattern -- the helper container **brokers** connectivity to external system
- **Init** pattern -- the special **init** container guarantees start/complete before run main application

Hands-on with Pods -- the second part of Chapter 4

access the following repository for the book:

```
$ git clone https://github.com/nigelpoulton/TheK8sBook.git
```

Pod manifest files explained -- under pods (folder) called pod.yml

same example as before, add // comment

```
kind:       Pod    // object
apiVersion: v1     // normally <api-group>/<version>  v1 example StorageClass
metadata:          // where you attach things
    name:   hello-pod
    labels:        // default namespace
        zone:     prod
        version:  v1
spec:              // spec will define the containers (desired state)
    containers:    // just one -- this is single container pod
    - name:  hello-ctr
      image: nigelpoulton/k8sbook:1.0
      ports:
      - containerPort: 8080
```

Manifest files: Empathy as Code

Nigel mentioned *Nirmal Mehta* @ `2017 dockercon talk` title A Strong Belief, Loosely Held: Bringing Empathy to IT

ie: the manifest(s) s/b excellent sources of documentation

Deploying Pods from a manifest file -- kubectl apply

```
$ kubectl apply -f pod.yml
# // wait some time ...
$ kubectl get pods
NAME       READY   STATUS             RESTARTS   AGE
hello-pod  0/1     ContainerCreating 0          9s
```

Introspecting running Pods -- kubectl get

- **-o wide** gives more columns
- **-o yaml** return full copy of the Pod from the cluster store

  Output defined in two parts: **desired state** (.spec) and **observed state** (.status)

```
$ kubectl get pods hello-pod -o yaml
#... // yaml-snip
spec:                     // desired state
    containers:
    - image: nigelpoulton/k8sbook:1.0
      imagePullPolicy: IfNotPresent
      name: hello-ctr
#... // yaml-snip
status:                   // current observed state
    conditions:
    - lastProbeTime:      null
      lastTransitionTime: "2022-02-08T18:21:51Z"
      status: "True"
      type:   Initialized
```

- put full listing in appendix TODO

kubectl describe

```
$ kubectl describe pods hello-pod
Name:        hello-pod
Namespace:   default
Start Time: Mon, 09 Feb 2022 18:21:51 +0000
Labels:      version=v1
             zone=prod
Status:      Running
IP:          10.42.1.28
Containers:
    hello-ctr:
        Container ID:   containerd://<blah>
        Image:          nigelpoulton/k8sbook:1.0
        Port:           8080/TCP
# ... // snip
Conditions:
    Type            Status
    Initialized     True
    Ready           True
    ContainersReady True
# ... // snip
Events:
    Type    Reason     Age     Message
    ----    ------     ----    -------
    Normal  Scheduled  5m30s   Successfully assigned ...
    Normal  Pulling    5m30s   Pulling image "nigelpoulton/k8sbook:1.0"
    Normal  Pulled     5m8s    Successfully pulled image ...
    Normal  Created    5m8s    Created container hello-ctr
    Normal  Started    5m8s    Started container hello-ctr
```

- put full listing in appendix TODO

kubectl logs

```
$ kubectl logs <multipod>
# ... // yaml-snip
spec:
    containers:
    - name: app                  // first container
      image: nginx
          ports:
            - containerPort: 8080
    - name: syncer               // second container
      image: k8s.ger.io/gi-sync:v3.1.6
      volumeMounts:
      - name: html
# ... // etc
```

$ kubectl logs multipod --container syncer // Example for specific container

kubectl exec: running commands in Pods

execute commands within the pod

```
$ kubectl exec hello-pod -- ps aux
PID    USER    TIME   COMMAND
 -1    root    0.00   node ./app.js
 11    root    0.00   ps aux
```

get shell access

```
$ kubectl exec -it hello-pod -- sh  // add --container if multi-container

// interactive in shell
# apk add curl
// then
# curl localhost:8080
```

Pod hostnames -- command is case sensitive

container in Pod will inherit hostname from name of the pod

valid DNS names (a-z 0-9, the minus-sign, and the period-sign)

Pod immutability

```
$ kubectl edit <name>
```

Kubernetes will prevent the change

Example multi-containers

init container

sidecar container

The Chaper ending and Summary

Cleanup

```
$ kubectl delete pod <name>
```

Chapter 4 Summary -- whats next ...

REF: eBook: TheUltimate Guide to Kubernetes Security

- mention in Austin-OWASP slack channel #kubernetes Jan27th (TODO URL)
- mention like the MARP-TEAM -- nice conversion from markdown to PDF

Appendix -- More detailed information

TODO