YAML Review

Let's discuss YAML, using simpler examples, but reflecting on what this means about the more complex example at the right.

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <some rule>
  egress:
  - <some rule>
```

YAML Review: Dictionaries

Items at the same level of indention, unless preceded by a hyphen (-), are like dictionary items:

```
kind: duck
age: juvenile
gender: male
```

represents an animal object, like a dictionary in Python.

Here, we have an anonymous object, whose "kind" key is "NetworkPolicy" and "apiVersion" is "networking...."

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <some rule>
  egress:
  - <some rule>
```

YAML Review: Nested Dictionaries

Items indented under another item, without hyphens (-), represent a dictionary nested within the item:

Under metadata, we have name and namespace. Here's what that creates:

```
metadata['name'] = 'networkpolicy'
metadata['namespace'] = 'default'
```

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <some rule>
  egress:
  - <some rule>
```

YAML Review: Lists

If we have an item indented under another item by a hypen and space, the top item is a list and the bottom item is a list item.

The policyTypes item is a list, with items "Ingress" and "Egress," like so:

```
policyTypes = ["Ingress","Egress"]
```

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <some rule>
  egress:
  - <some rule>
```

YAML Review: Complex Objects Level One

The first five lines at the right create a dictionary. The value for "kind" is "NetworkPolicy," while the value for "apiVersion" is "networking.k8s.io/v1," but the value for "metadata" is a nested dictionary:

```
object['kind'] = 'NetworkPolicy'
object['apiVersion'] = 'networking.k8s.io/v1'
object['metadata']['name'] = 'networkpolicy'
object['metadata']['namespace'] = 'default'
```

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <some rule>
  egress:
  - <some rule>
```

YAML Review: Complex Objects Level Two (1 of 6)

The spec key's value is a dictionary, with keys:

```
podSelector
policyTypes
ingress
egress
```

The podSelector's value is a single-item dictionary, with only a "matchLabel" key.

The matchLabel's value is a single item dictionary, with the key "label" set to "value".

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <in rule 1>
  - <in rule 2>
  egress:
  - <some rule>
```

YAML Review: Complex Objects Level Two (2 of 6)

Translating these two statements into pseudo-code:

1. The podSelector's value is a single-item dictionary, with only a "matchLabel" key.
2. The matchLabel's value is a single item dictionary, with the key "label" set to "value".

We see:

```
podSelector['matchLabels']['label'] = 'value'
```

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <in rule 1>
  - <in rule 2>
  egress:
  - <some rule>
```

YAML Review: Complex Objects Level Two (3 of 6)

policyTypes' value is a two-item list:

```
policyTypes = ['Ingress','Egress']
```

Since policyTypes is a key of the spec dictionary, we can see this as:

```
spec['policyTypes'] = ['Ingress','Egress']
```

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <in rule 1>
  - <in rule 2>
  egress:
  - <some rule>
```

YAML Review: Complex Objects Level Two (4 of 6)

The ingress key's value is also a list:

```
spec['ingress']=['<in rule 1>','<in rule 2>']
```

So we can put that in the context of the object itself, like so:

```
object['spec']['ingress'] =
               ['<in rule 1>','<in rule 2>']
```

The egress key's value is a single-item list:

```
spec['egress']=['<some rule>']
```

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <in rule 1>
  - <in rule 2>
  egress:
  - <some rule>
```

YAML Review: Complex Objects Level Two (5 of 6)

Putting this all together - the file on the right describes a dictionary with keys:

kind	a string
apiVersion	a string
metadata	a dictionary, with two keys
spec	a dictionary, with four keys

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - <in rule 1>
  - <in rule 2>
  egress:
  - <some rule>
```

YAML Review: Complex Objects Level Two (6 of 6)

object['spec'] is a dictionary (dict) with four keys:

podSelector	a dict, containing a dict
policyTypes	a list
ingress	a list
egress	a list

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - <in rule 1>
  - <in rule 2>
  egress:
  - <some rule>
```

Network Policies

- Network policies are Kubernetes' built-in firewall capabilities for pods.
- We'll discuss another method of traffic control later: service meshes.

Network Policy Introduction and Structure

Network policies create firewall rules, using label selection.

You create one or more policies.

Each policy names pods that it applies to via a label-based `podSelector`.

Once you create a network policy for a pod, you have a default deny for traffic for that pod in that direction.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: networkpolicy
  namespace: default

spec:
  podSelector:
    matchLabels:
      label: value

  policyTypes:
  - Ingress
  - Egress

  ingress:
  - <some rule>
  egress:
  - <some rule>
```

Network Policy Example

This example describes what kind of traffic is allowed inbound to the pods whose labels match:

```
app = bookstore
role = backend
```

It permits traffic ONLY from pods whose **`app`** label matches **`bookstore`**.

These labels have no inherent meaning, except conventional.

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: bookstore-backend
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: backend

  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: bookstore
```

Port-specific Rules

The policy can name ports, rather than simply allowing all traffic.

This policy permits incoming traffic to the bookstore backend pods' TCP ports 80 and 443.

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: bookstore-backend-web-ports
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: backend

ingress:
  - ports:
    - protocol: TCP
      port: 80
    - protocol: TCP
      port: 443
```

Combining Source Labels and Destination Ports

The policy can name ports, rather than simply allowing all traffic.

This policy permits traffic to port 80 on the bookstore backend pods, when it originates from other bookstore pods.

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: bookstore-backend-from-bookstore-pods
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: backends

ingress:
  - from:
      - podSelector:
          matchLabels:
            app: bookstore
    ports:
    - protocol: TCP
      port: 80
```

Network Policy Wildcards and Allow-All

To make something apply to all pods or all ports or so on, use {}.

- In a podSelector, {} means “all pods”
- In an ingress or egress list, {} means "everything".

The example on the right permits all pods to receive inbound traffic without restriction.

```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  ingress:
  - {}
```

Network Policy by IP Address

The podSelector, which indicates what pods this policy controls traffic into (ingress) or out of (egress), uses only labels. It does not use pod names or IP addresses.

On the other hand, the ingress and egress rules can use labels but can also use IP addresses.

```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress-to-bookstore
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: bookstore
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
        - 172.17.3.0/24
```

Network Policy Methodology

- Unless a pod has a single network policy, the pod can receive or send without restriction.
- Once a pod has one or more network policies in a given direction (ingress or egress), traffic is governed by default-deny.
- Multiple network policies can be in place for a pod. If any network policy would allow the traffic, it is allowed.
- If a pod has an ingress policy, but no egress policy:
  - Incoming (ingress) traffic is default-deny.
  - Outbound (egress) traffic is default-allow.
- If a pod has an egress policy, but no ingress policy:
  - Outbound (egress) traffic is default-deny.
  - Incoming (ingress) traffic is default-allow.

Exercise: Network Policies

Please, open the Firefox browser on the class machine to:

http://localhost:10000/exercises/kubernetes-network-policies