# Introduction to Kubernetes

# About me

- George Paraskevas
- Linux Systems Engineer
- DevOps @ Yodeck
- Automation driven (Lazy)
- Terraform/Ansible/Helm/Python/bash
- Openstack/k8s/Networking/IPv6

# Presentation Topics

- What is Kubernetes
- High level Architecture
- k8s Toolset
- Basic resources definitions
- Advanced concepts
- The good the bad and the ugly

# What is Kubernetes (k8s)

- Container orchestration platform.
- Written in Go.
- Made by Google.
- Opensourced in 2014, seeding the CNCF.
- Current version 1.28.2
- Concepts
  - Schedules containers across nodes.
  - Automates operational tasks.
  - Container Loadbalancing.
  - Self-healing.
  - Horizontally scalable.

# Architecture

- Odd number of nodes acting as Masters (control plane)
    - API
- X amount of Nodes acting as workers (data plane)
    - Container runtime
- Components
    - API Server
        - The frontend used for interacting with the Kubernetes cluster
    - Etcd
        - Stores all information about nodes and workload
    - Scheduler
        - Assigns containers to nodes
    - Controller
        - Ensures that the system converges to the desired state (thermostat of the cluster)
    - Container runtime
        - The software used to run containers (containerd)
    - Kubelet
        - Receives commands from the API and instructs the container runtime

# Kubectl

**Kubectl** is the necessary **cli** to interact with a k8s cluster.

```
$ kubectl get nodes

$ kubectl get pods

$ kubectl get service

$ kubectl describe pod pod-0

$ kubectl logs –follow –tail=100 pod-0
```

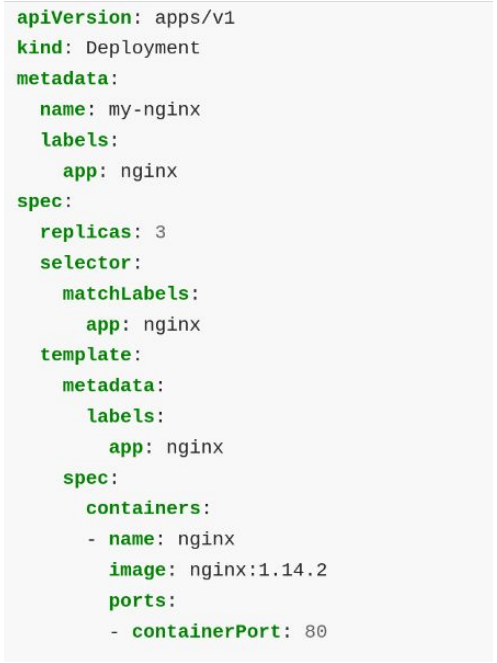
# Manifests

**Manifests**:

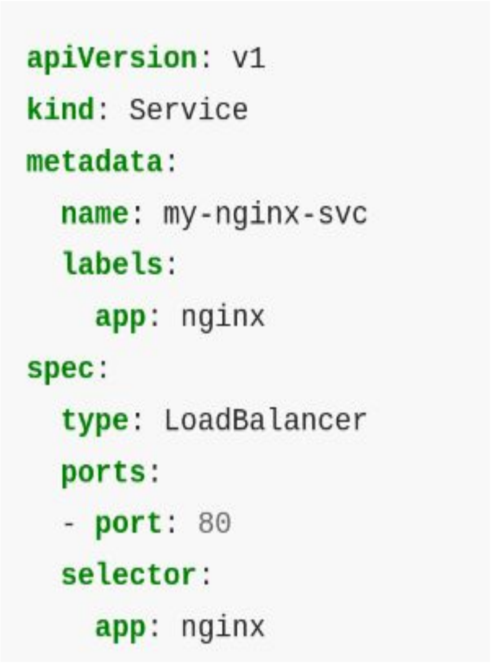
Yaml representations of k8s **resources*** that can be applied with **kubectl**

**resources:**

K8s objects that translate to containers/configurations/loadbalancers



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```



```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx-svc
  labels:
    app: nginx
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: nginx
```

# Basic resources

- Pod
- Deployment
- Service
- Deployment Strategies
- Probes
- Resources Limits/Requests
- job/cronjob
- Namespace
- configMap

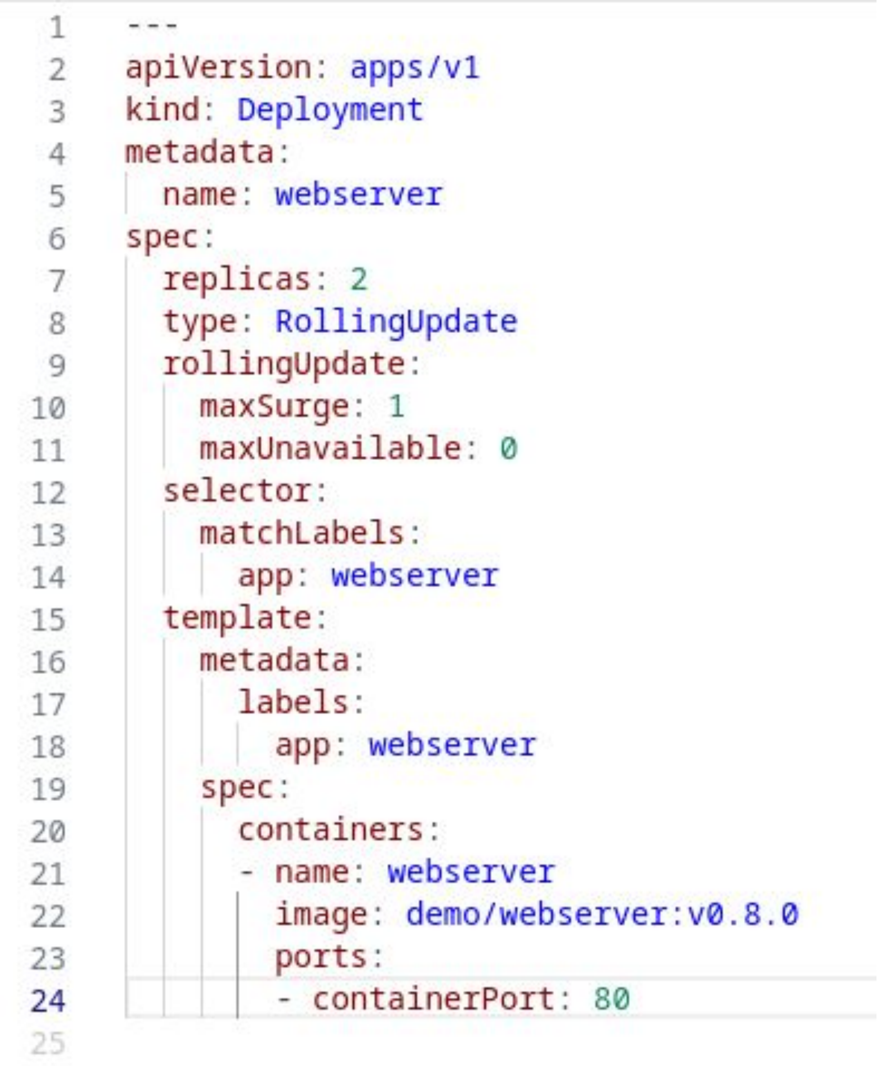
# Pod

- A Pod is the smallest object you can create in k8s
- One or more containers can live inside the same pod
  - They share the same network
  - They also share the same fate
    - Created and destroyed together
  - Used for helper applications
    - Collecting logs
    - Monitoring adapters
    - Reverse proxies
- Pods are designed to be ephemeral
  - There is no expectation that a specific, individual pod will persist for a long lifetime.

# Deployment

- A collection of pods defined by a template
  - Replicas
  - Deployment strategy
  - Healthchecks
- Kubernetes can’t guarantee the life of a Pod
  - But can guarantee the deployment replicas
- Best suited for stateless applications
  - Pods can be replaced any time without breaking things

# Deployment manifest



```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver
spec:
  replicas: 2
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 0
  selector:
    matchLabels:
      app: webserver
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: webserver
        image: demo/webserver:v0.8.0
        ports:
        - containerPort: 80

```

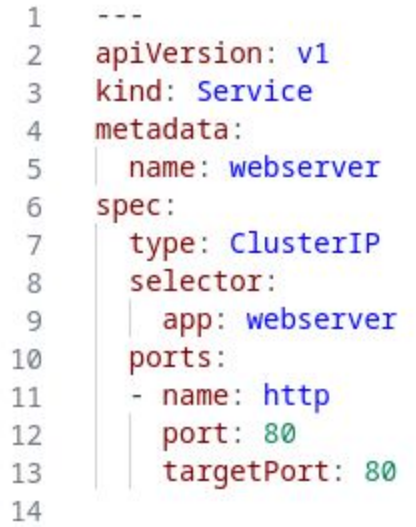
# Service

- Each pod has an ephemeral IP address
    - When the pod gets rescheduled or recreated, the ip changes.
- Services create as a stable endpoint in the form of a DNS record
    - **service-name.namespace**.svc.cluster.local
    - Directs traffic to a pod, or a set of pods based on their labels
    - Loadbalances traffic based on pod health
    - So pods can scale up or down and traffic head towards them
    - No changes in endpoints to other services

# Service types

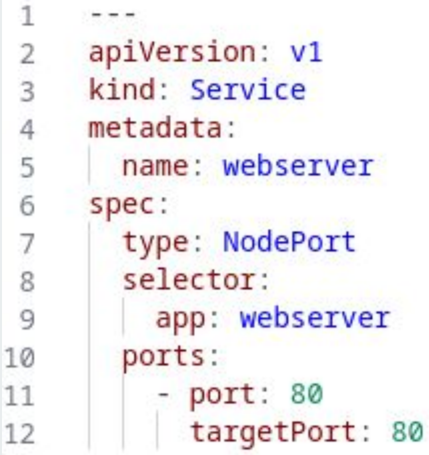
- ClusterIP
  - Allocates a static internal IP from the cluster
- NodePort
  - Exposes a port in the node that runs the container
    - Default nodePort range 30000-32767
- LoadBalancer
  - Allocates an external IP using a service
    - This service is responsible for the lifecycle of the IP
    - aws/gcp/metallb
- External name
  - Maps a service to a DNS name
  - Returns a CNAME record to that name
- Headless
  - A service that can point to specific pods
  - Pod-name.servicename.namespace.svc

# Service manifest



```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: webserver
spec:
  type: ClusterIP
  selector:
    app: webserver
  ports:
  - name: http
    port: 80
    targetPort: 80

```



```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: webserver
spec:
  type: NodePort
  selector:
    app: webserver
  ports:
    - port: 80
      targetPort: 80
```

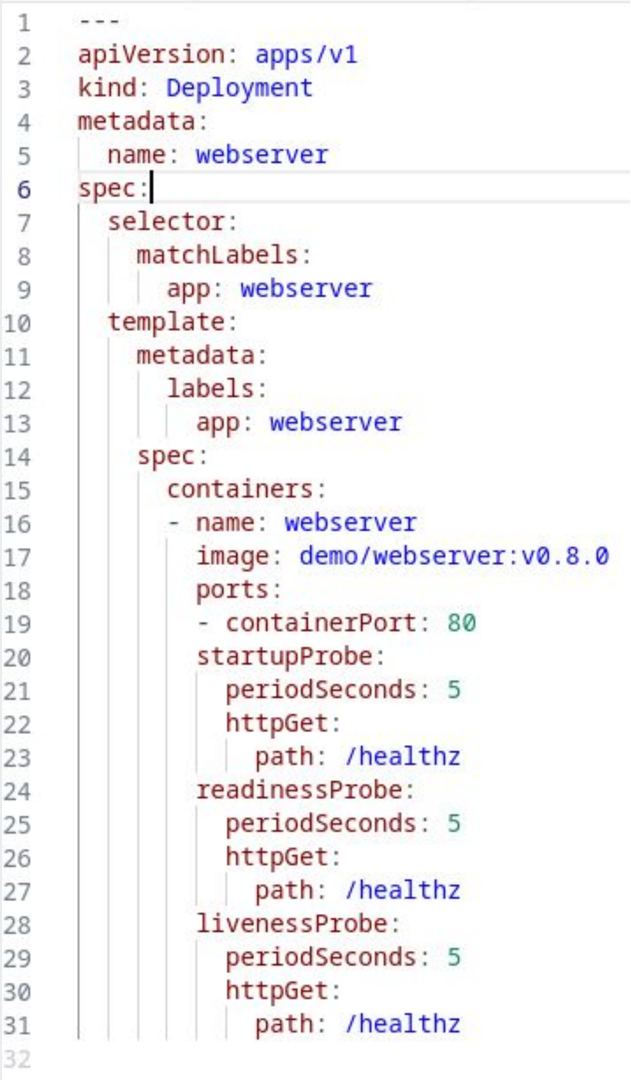
# Deployment strategies

- Rolling
  - Replaces slowly all older pods with new ones
  - Rollout can take time
  - No control over traffic
  - Good for client facing apps (appservers, webservers)
- Recreate
  - Terminates all running pods and replaces them with new ones
  - Comes with downtime
  - Good for worker type applications(Kafka consumers, celery workers)

# Probes (Healthchecks)

- Startup probe allows kubelet to know if a container has started
  - If configured, liveness and readiness probes are disabled until container is up
    - Makes sure that those probes don’t interfere with the startup
    - Ensures that pod with slow startup won’t get restarted constantly
- Liveness probe knows when to restart a container
- Readiness probe knows when a container is ready to accept traffic
  - A Pod is ready when all containers are ready
  - When a Pod is not ready, it is removed from the Service load balancer
    - But it is not restarted

# Probe types



```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver
spec:
  selector:
    matchLabels:
      app: webserver
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: webserver
        image: demo/webserver:v0.8.0
        ports:
        - containerPort: 80
        startupProbe:
          periodSeconds: 5
          httpGet:
            path: /healthz
        readinessProbe:
          periodSeconds: 5
          httpGet:
            path: /healthz
        livenessProbe:
          periodSeconds: 5
          httpGet:
            path: /healthz
```

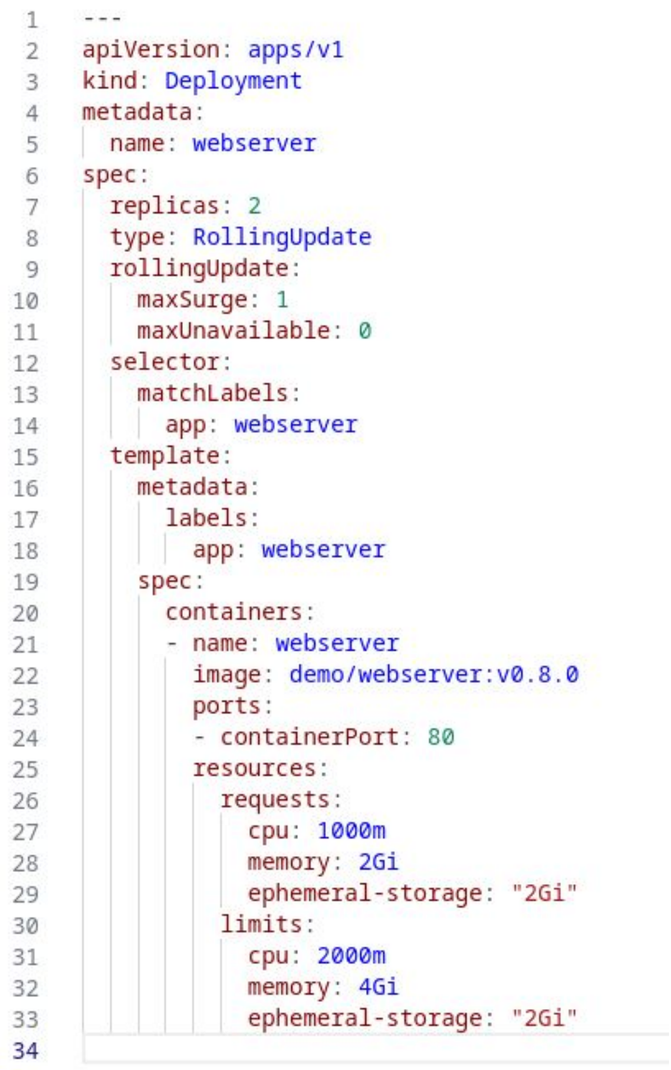
# Probe types

- Command inside the container
  - Must return exit code 0
- HTTP Request
  - Code 200 <= X < 400
- TCP
  - If connection establishes, probe is successful
- gRPC
  - New feature, do your own research :P

# Resources limits and requests

- Limits
  - Limits the amount of memory, cpu and storage a pod can use.
  - Exceeding this limit will result in pod restart
- Requests
  - Specifies the amount of memory, cpu and storage a pod requires upon scheduling.
  - If the requested resources are available in a single node then the pod will be scheduled.

# Limits/Requests

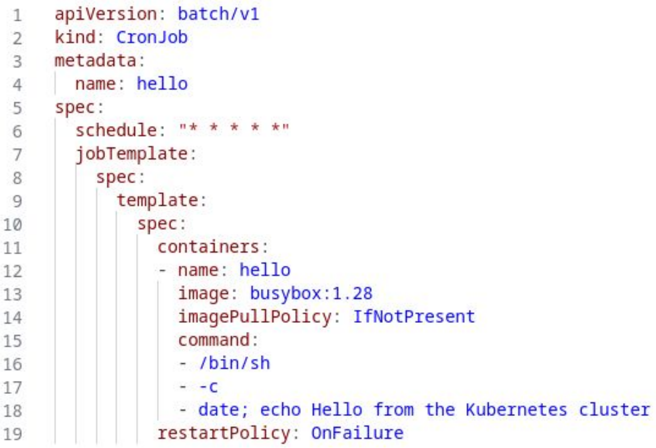


```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver
spec:
  replicas: 2
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 0
  selector:
    matchLabels:
      app: webserver
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: webserver
        image: demo/webserver:v0.8.0
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: 1000m
            memory: 2Gi
            ephemeral-storage: "2Gi"
          limits:
            cpu: 2000m
            memory: 4Gi
            ephemeral-storage: "2Gi"
```

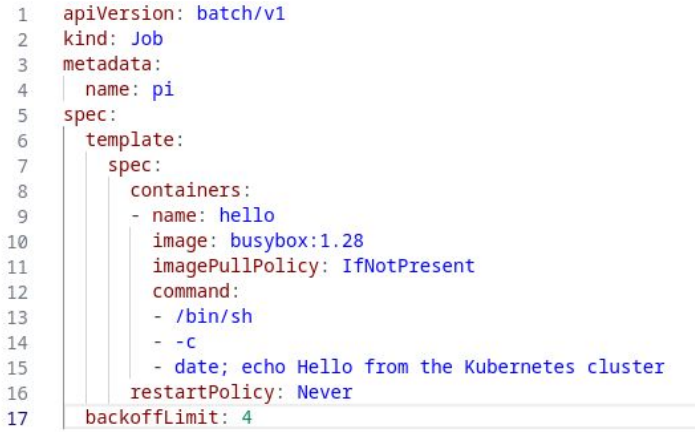
# Jobs and CronJobs

- Deployments are used to create reliable, long running services
- What if we want to execute a discrete task?
  - Let’s say to generate a report
- The desired state of Jobs is to complete the job
  - If pod crashes at some point during the execution of a job, a new one gets rescheduled
  - K8s allows us to set a back-off limit before quitting
- CronJob allows you to create Jobs on a repeating schedule
  - Uses the cron format

# Job / CronJob manifest



```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox:1.28
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```



```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: hello
        image: busybox:1.28
        imagePullPolicy: IfNotPresent
        command:
        - /bin/sh
        - -c
        - date; echo Hello from the Kubernetes cluster
      restartPolicy: Never
  backoffLimit: 4
```

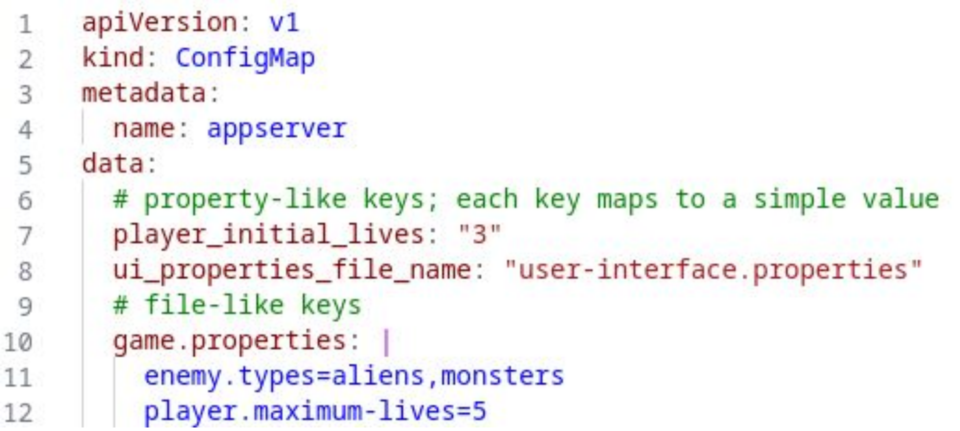
# Namespaces

- A logical grouping and isolation of resources
- A scope for names
  - Resource names
- A way to divide cluster resources between multiple users

# ConfigMaps

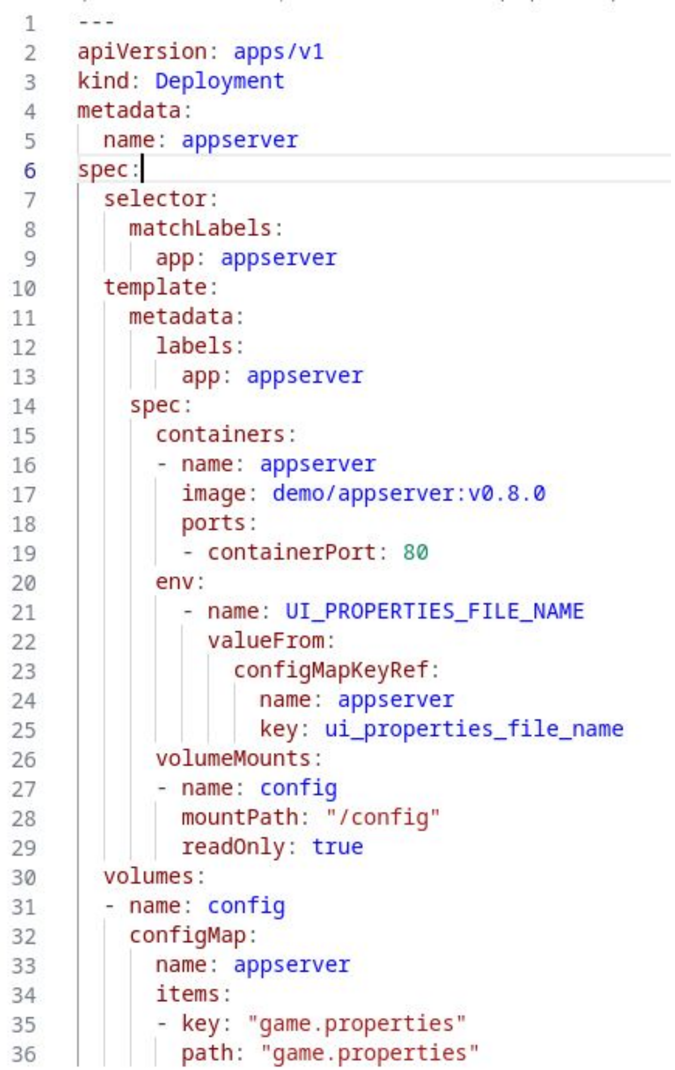
- non-confidential data in key-value pairs
- decouple environment-specific configuration from your container image
- as environment variables, command-line arguments, or as configuration files in a volume.
- For complex or big configuration files
- For apps that cannot use ENV VARIABLES

# ConfigMaps



```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: appserver
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"
  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
```

# ConfigMaps



```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appserver
spec:
  selector:
    matchLabels:
      app: appserver
  template:
    metadata:
      labels:
        app: appserver
    spec:
      containers:
      - name: appserver
        image: demo/appserver:v0.8.0
        ports:
        - containerPort: 80
      env:
        - name: UI_PROPERTIES_FILE_NAME
          valueFrom:
            configMapKeyRef:
              name: appserver
              key: ui_properties_file_name
      volumeMounts:
      - name: config
        mountPath: "/config"
        readOnly: true
  volumes:
  - name: config
    configMap:
      name: appserver
      items:
      - key: "game.properties"
        path: "game.properties"
```

# Advanced concepts

- Helm charts
  - A package manager for k8s apps
- Ingress
  - Layer 7 loadbalancing on k8s
- ArgoCD
  - Declarative gitops for k8s
  - A CD for k8s

# The good

- k8s is fun to use.
- k8s makes a lot of things easier, from an operators view.
- Managed k8s on all major platforms
- Vast catalog of Helm charts
- Everything is code.
- Argocd makes everything so easier.

# The bad

- k8s has a learning curve.
- Managed k8s have their own learning curve.
- No grpc loadbalancing by default.
- Not all applications are ready for k8s.
- A lot of moving parts.
- A lot of abstraction.

# The ugly

- Some managed k8s are not so managed afterall.
- A lot of differences between managed providers.
- One application with no limits can bring a node down.
- Stateful apps require extra care.

# Questions

CAN I ASK 1 QUESTION?
YOU JUST DID
HAHA.. OK CAN I ASK 2?
YOU ALREADY DID
OK CAN I ASK 4 QUESTIONS?
YOU JUST DID
WHEN ?
NOW

# Thank you