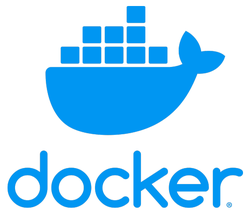
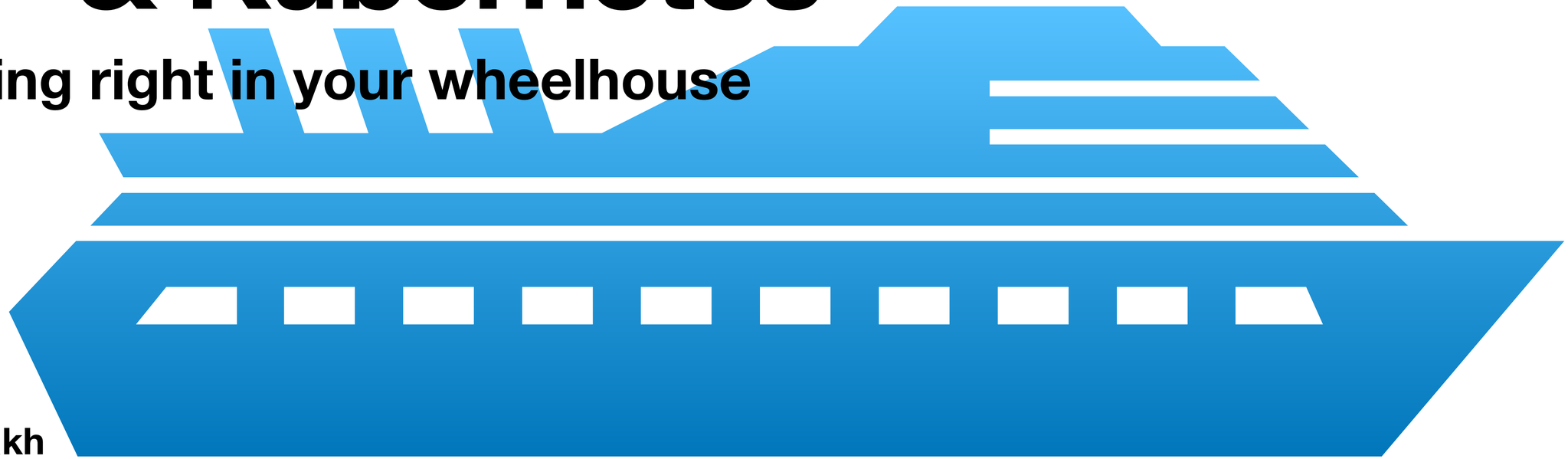


PHP & Kubernetes

Deploying right in your wheelhouse



Denys Bulakh

Denys Bulakh
Introduction

- Programmer
- Web Software Developer
- CTO at Regiondo
- Principal Engineer at Jochen Schweizer Mydays Group

REGIONDO

JOCHEN SCHWEIZER
mydays GROUP

JOCHEN SCHWEIZER · JOCHEN SCHWEIZER NOW! · mydays · ERLEBNISWERK · HIP TRIPS · REGIONDO · spontacts

What will we talk about?

Application deployment

Deployment automation

Application scaling

Local development

Logging

How to deploy **web application**?

1st challenge

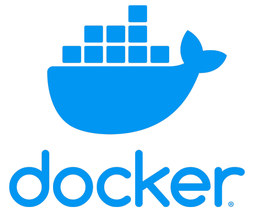
It runs permanently on web server

2nd challenge

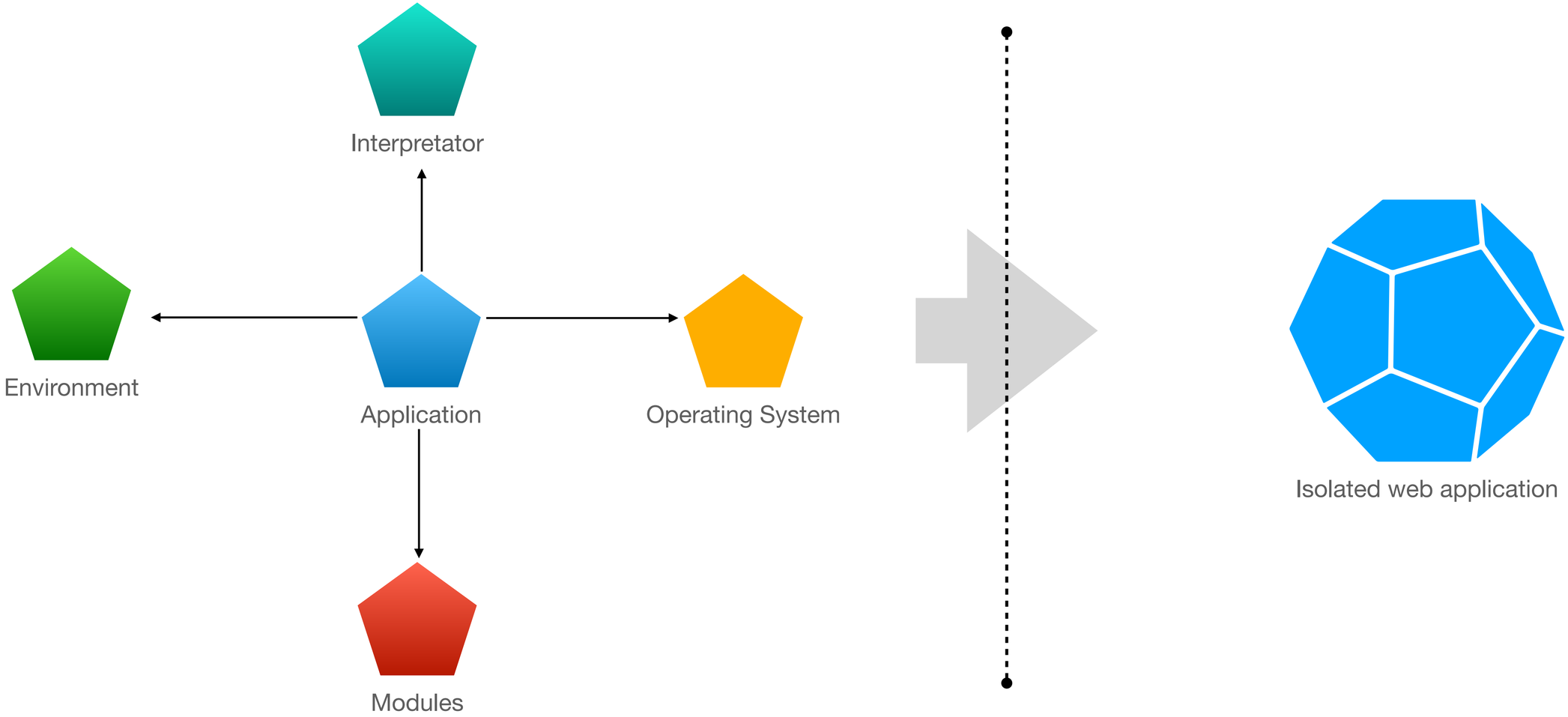
Application versioning

3rd challenge

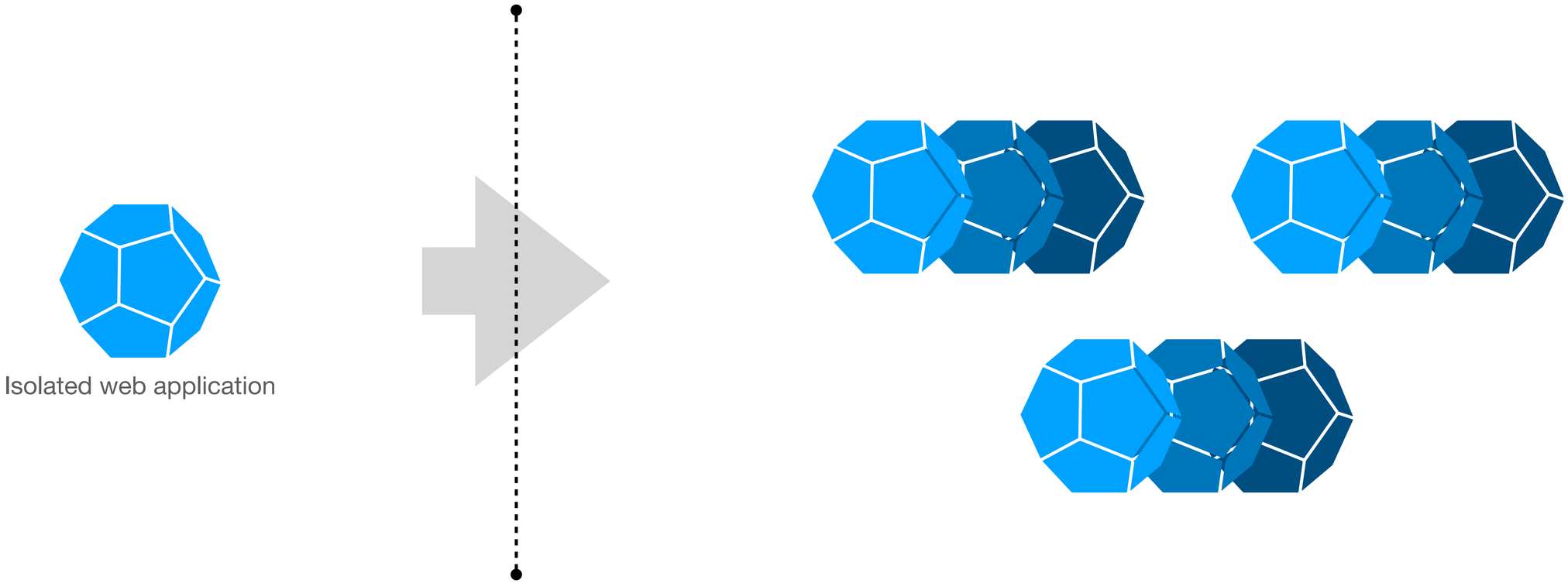
State management



Application containerisation helps to make stateless isolated web application

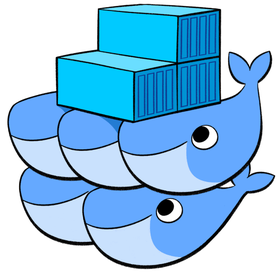


Instead of managing state and dependencies, we can include and isolate them in the application



We can run as many isolated web applications, as we need, without need to take care about state and dependencies

Orchestrators - tools to manage, scale, and maintain containerised applications



kubernetes

Features

- Automated rollouts and rollbacks
- Service discovery and load balancing
- Storage orchestration
- Secret and configuration management
- Horizontal scaling
- Self-healing
- IPv4/IPv6 dual-stack
- Managing resources for containers

https://kubernetes.io/

Kubernetes does not deploy source code and does not build your application

https://kubernetes.io/

Kubernetes does not provide application level services, such as databases, caches, middleware as build-in services

https://kubernetes.io/

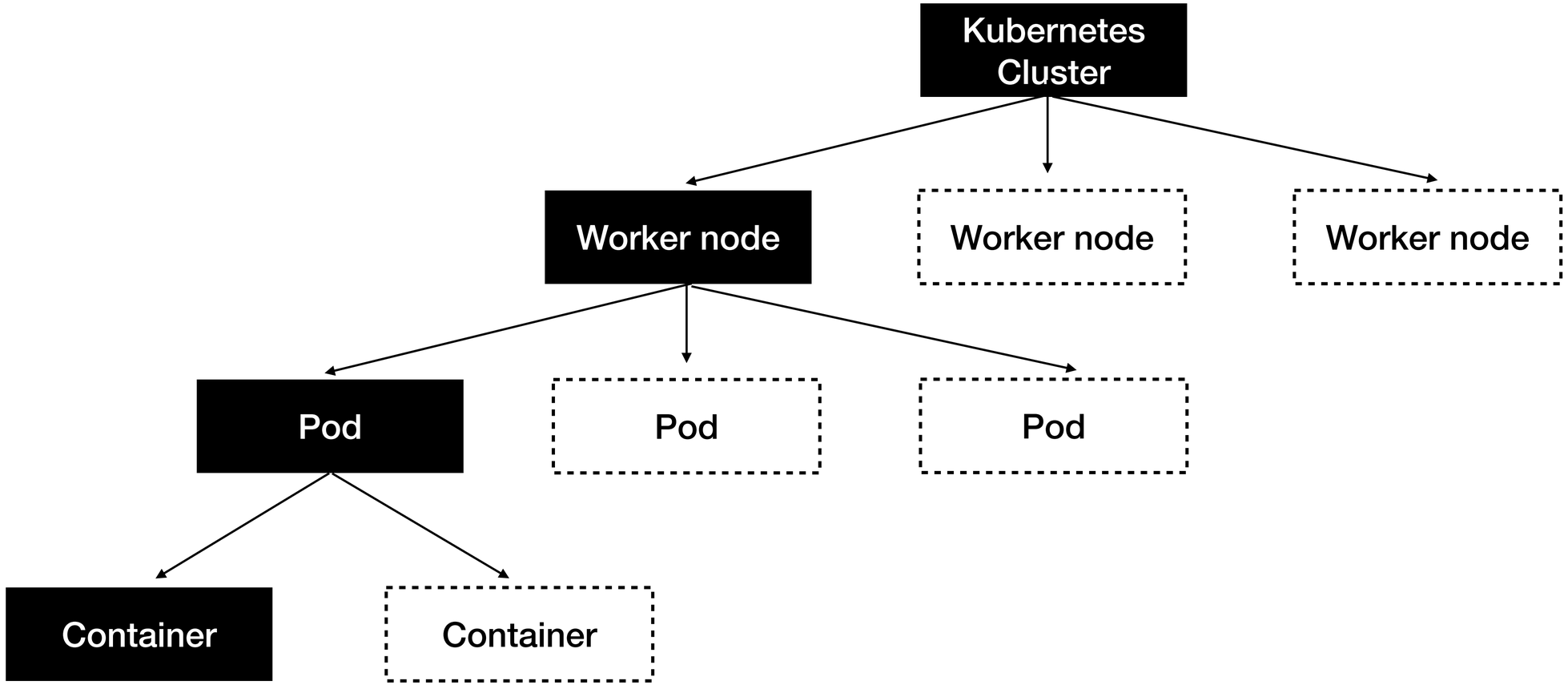
If an application can run in a container, it should run great on Kubernetes

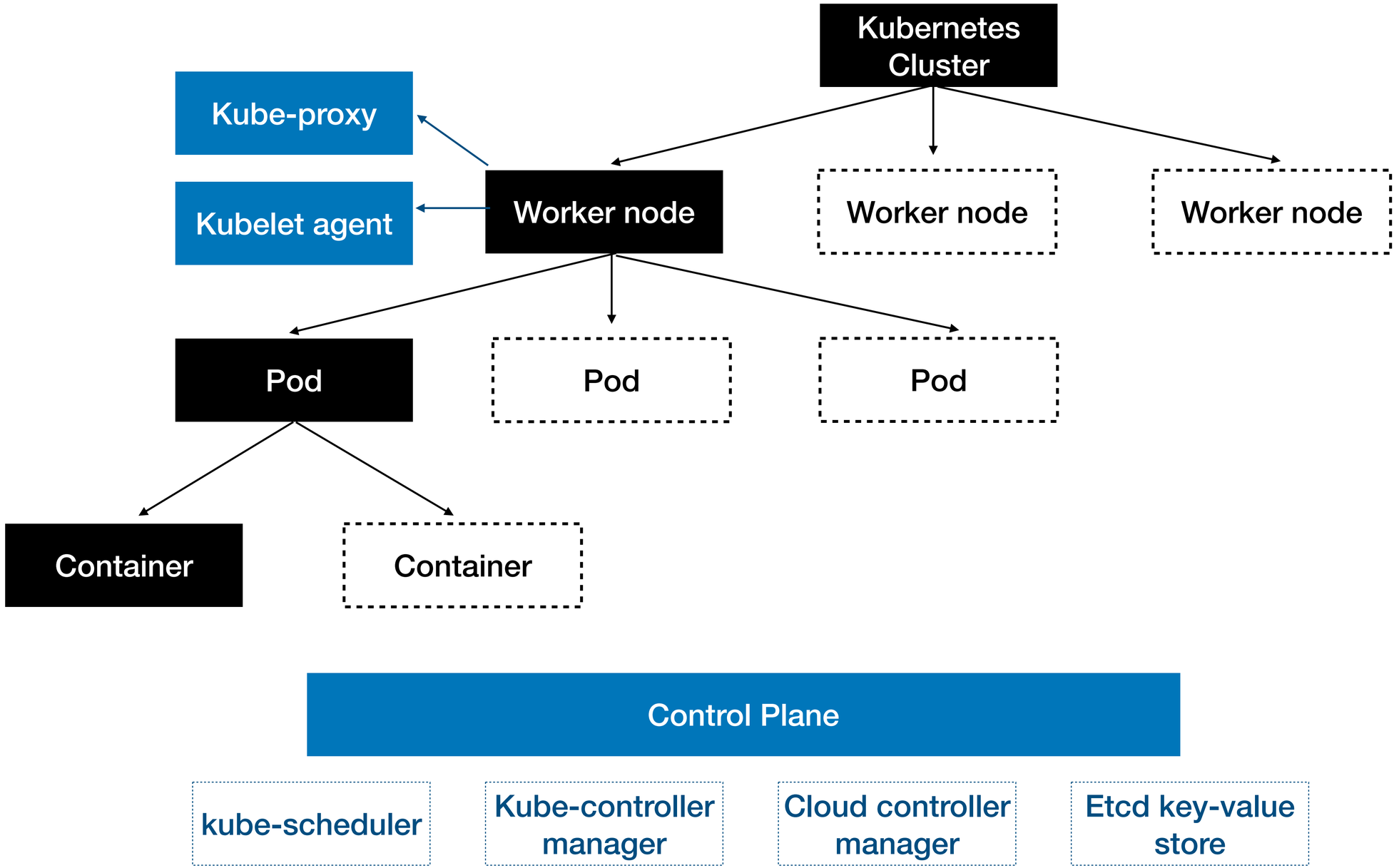
https://kubernetes.io/

PHP applications can be easily containerised and, therefore, deployed with Kubernetes

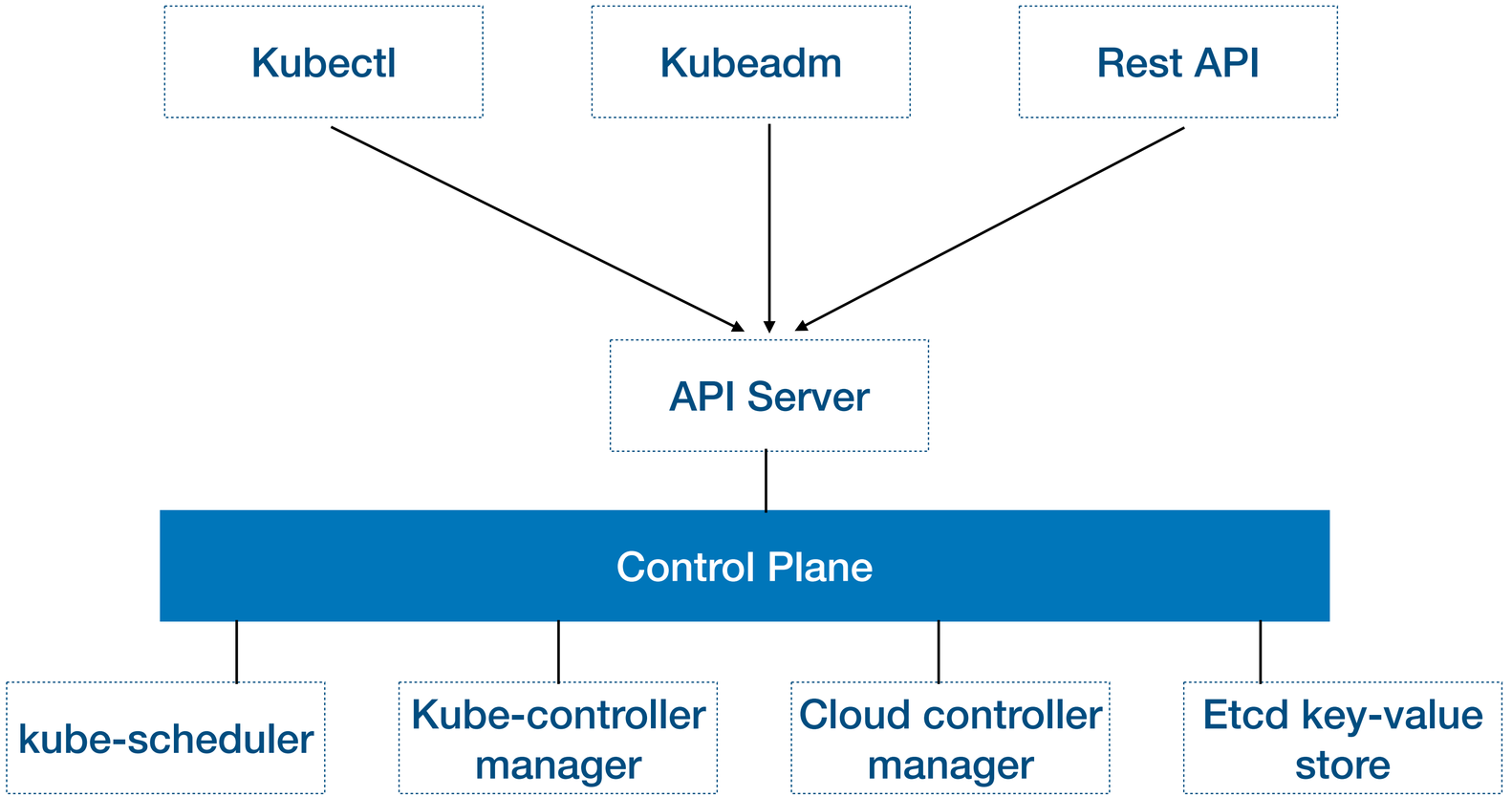
+

kubernetes


Kubernetes Cluster
Worker node
Worker node
Worker node
Pod
Pod
Pod
Container
Container


Kubernetes Cluster
Kube-proxy
Kubelet agent
Worker node
Worker node
Worker node
Pod
Pod
Pod
Container
Container
Control Plane
kube-scheduler
Kube-controller manager
Cloud controller manager
Etcd key-value store

```
kubectl [command] [TYPE] [NAME] [flags]
```



Kubernetes is using objects, as persistent entities in the system, to represent the state of the cluster

Kubernetes Object

What and where is running?

Which resources are available?

Restart, upgrade, fault-tolerance policies

By creating an object, you're telling the Kubernetes system what do you want your cluster's workload to look like, this is your cluster's desired state

https://kubernetes.io/

Declare Kubernetes object's desired configuration

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

A. **version:** can be v1, v1beta and v2.
B. **kind:** Pod, DaemonSet, Deployment, Service.
C. **metadata:** name of the pod, namespace, labels, annotations.
D. **spec:** desired state of the pod, including container image, replicas count, environment variables and volumes.

```
kubectl apply -f https://k8s.io/examples/application/deployment.yaml --record
```

https://kubernetes.io/

Other management techniques

Objects can be managed imperatively, by operating directly on live objects in cluster, using kubectl commands

```
kubectl create deployment nginx --image nginx
```

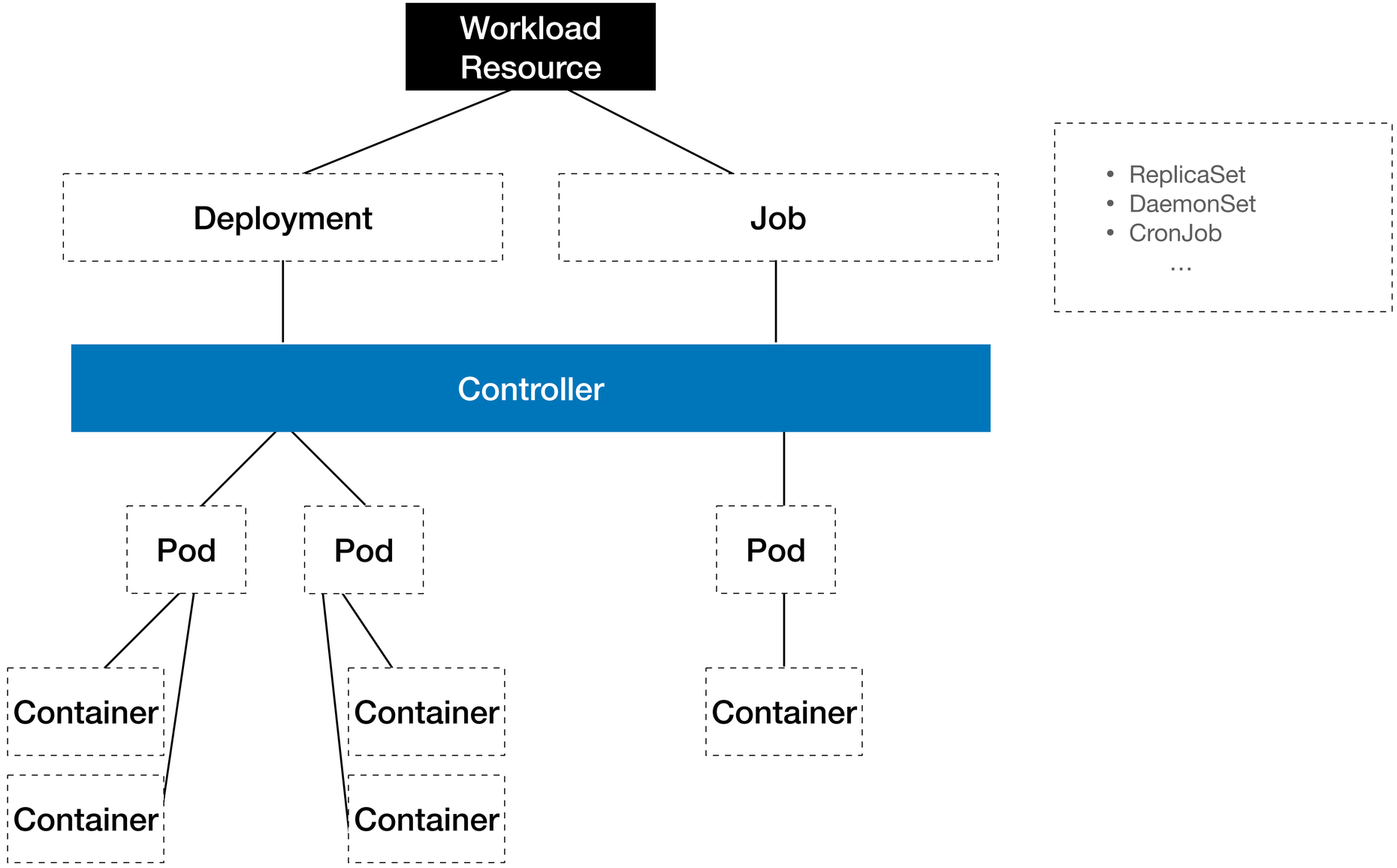
Or using kubectl command to specify the operation (create, replace, etc.), optional flags and at least one file name. The file specified must contain a full definition of the object in YAML or JSON format.

```
kubectl create -f nginx.yaml
```

```
kubectl delete -f nginx.yaml -f redis.yaml
```

https://kubernetes.io/

Pod is a smallest deployable compute unit that you can create and manage in Kubernetes


Workload Resource
Deployment
Job
ReplicaSet
DaemonSet
CronJob
...
Controller
Pod
Pod
Pod
Container
Container
Container
Container
Container

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  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
```

ReplicaSet to be created

Pod template:

Job

Pod template:

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

Number of retries

CronJob

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

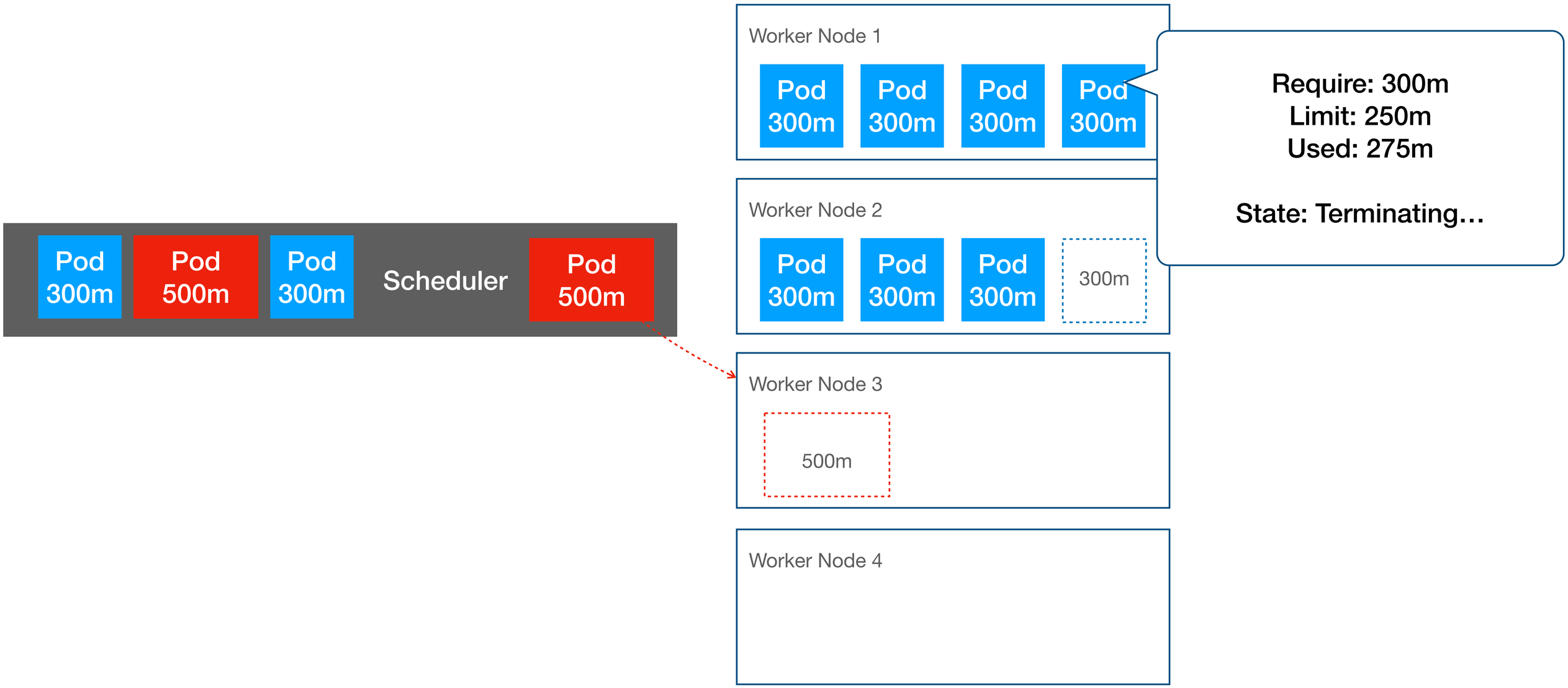
Use resources requests and limits available for containers on every Pod right in your Kubernetes Deployment config

Resource limits

```yaml
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

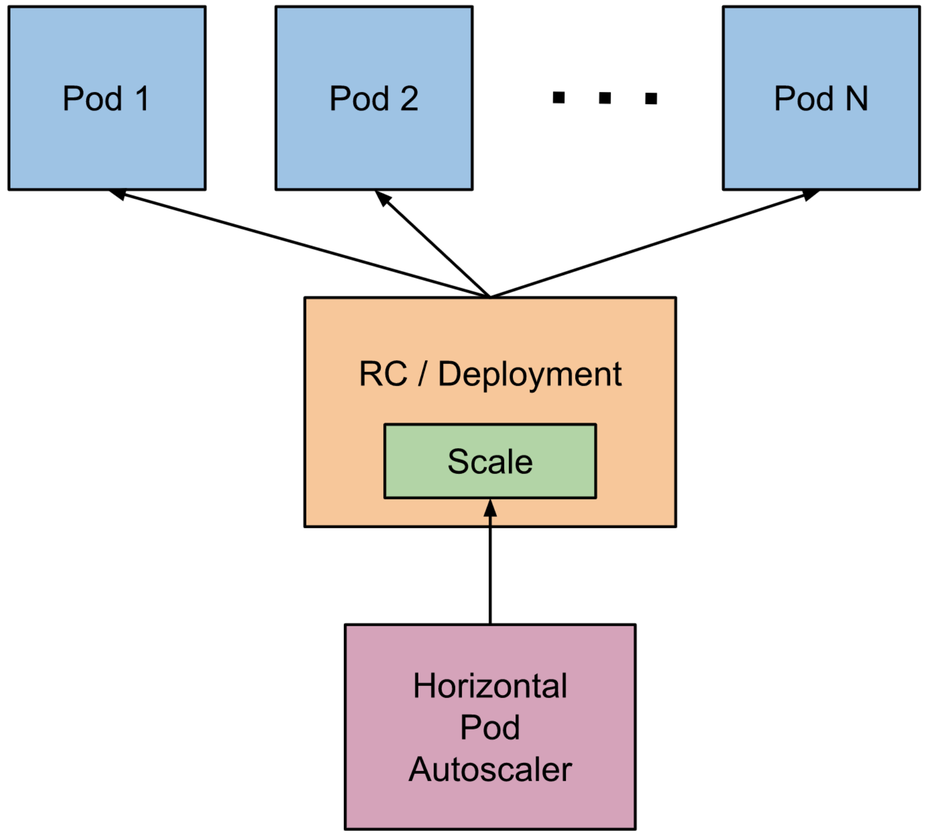
- *request* value is used by scheduler to decide which node to place the Pod on
- *limit* value shows how much resources container is allowed to use
- CPU is specified in units of Kubernetes CPU, where 1 CPU is equivalent of 1 vCPU/ core on cloud providers
- 250m (millicpus) = 0.25 CPU

Split between nodes by *require* value



Resource limitation helps scheduler to manage containers automatically

Horizontal Pod Autoscaler automatically scales the number of Pods in your deployment, based on CPU utilisation (or application-provided metrics)

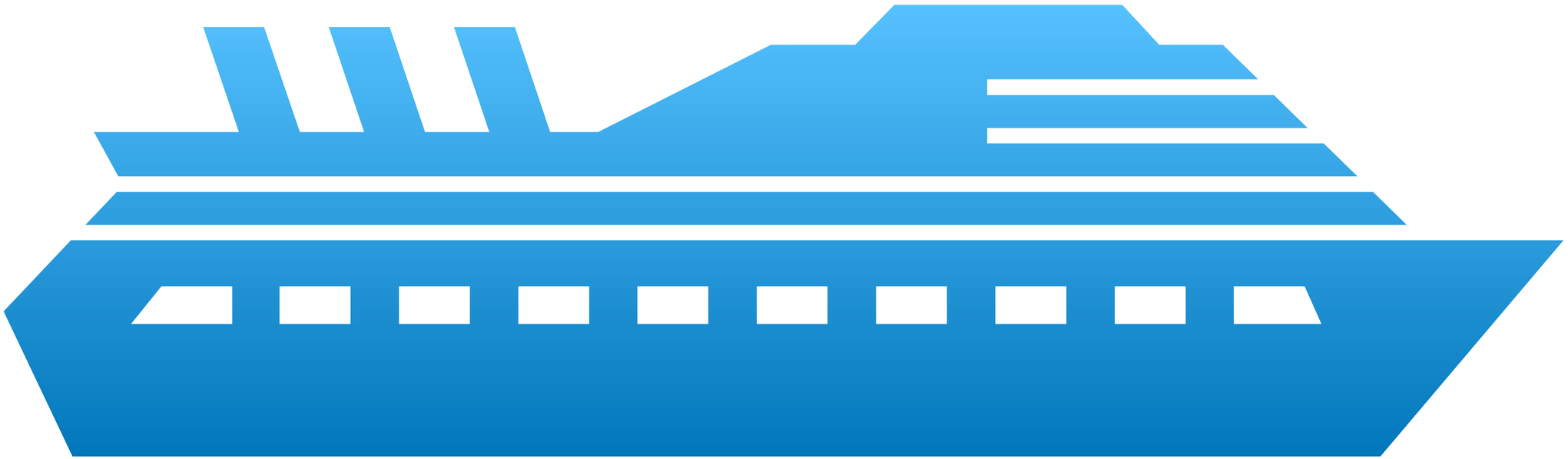


The controller manager periodically queries the resource utilisation and adjusts the number of replicas in a deployment to match the observed average CPU utilisation to the target specified by user.

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

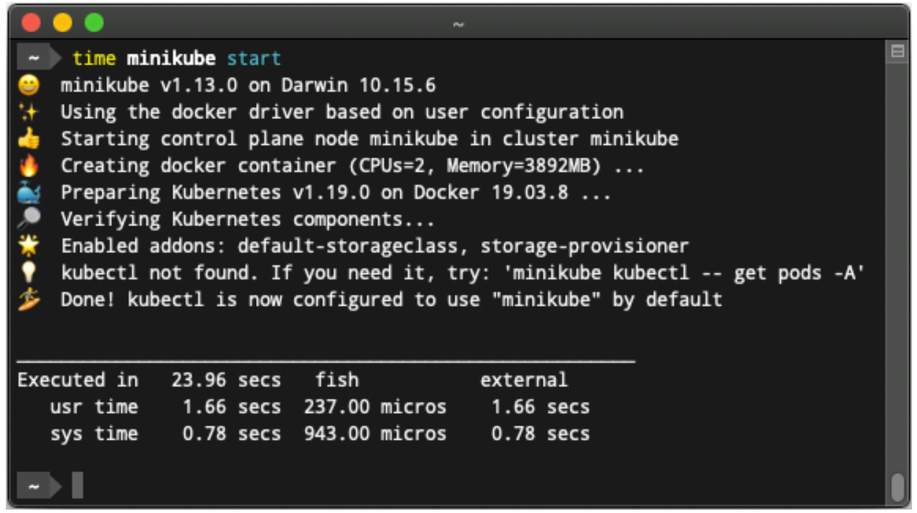
Autoscaler will aim to maintain 50% of average CPU utilisation on all Pods

Demo

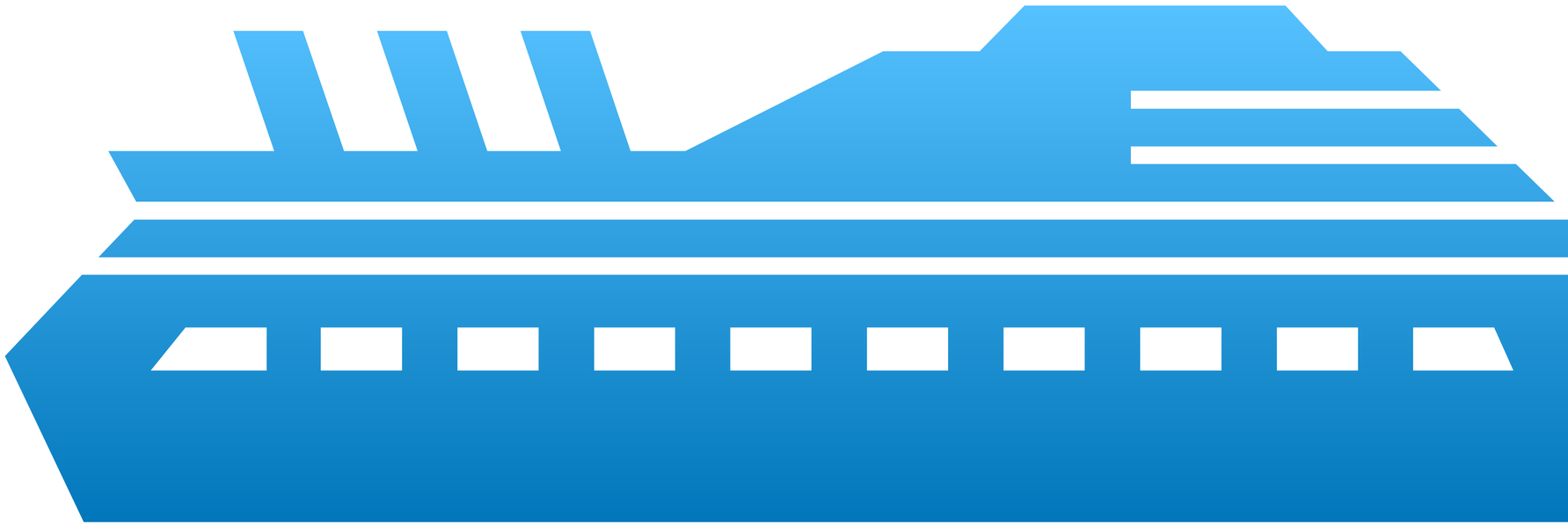


minikube

minikube is an easy way to quickly setup Kubernetes cluster locally



Questions?



Thank you!

Denys Bulakh
https://www.linkedin.com/in/denysbulakh/