Kubernetes
Motivation

- Modern containerized applications can have thousands of containers.
- It is impossible to manage them manually.
- We want to be able to:
  - Deploy large clusters of containers automatically,
  - Divide the workload between them,
  - Return the system back to the normal state in case of malfunctions,
  - Update the containers and the whole cluster seamlessly.
- This set of tasks is called orchestration. There is a solution to these problems called ...
Master and Nodes

• Kubernetes Master:
  – API Server: interface to the master
  – Cluster Store: storing cluster configuration
  – Controller Manager: ensuring the correct state
  – Scheduler: assigning workloads to nodes

• Kubernetes Node:
  – Kubelet: the main Kubernetes agent
  – Container Runtime (e.g. Docker)
  – Network Proxy: assigning IP addresses to Pods
Declarative Model

• Kubernetes ensures that the current state of the system corresponds to the so-called *desired state*.

• The desired state is declared in a manifest file, which is submitted to the API server.

• Kubernetes deploys the system according to the received specification of the desired state.

• After the deployment, Kubernetes constantly monitors the state of the system and detects any variations from the desired state.

• If the current state differs from the desired state, Kubernetes automatically performs all the necessary actions to return the system back to the desired state.
Pods

- Pod is the basic unit of deployment.
- Each container has to run inside a pod.
- A pod can contain more than one container.
  - E.g. if two containers need to share a storage volume.
  - You should not scale your app by adding more containers to a single pod!
- Every pod gets its own unique IP address in the pod network, which allows pods to communicate with each other directly.
• Pods are declared in manifest files written in YAML or JSON format.
• In order to deploy a pod, its manifest file has to be POSTed to the API Server.
• The control plane will schedule the pod on one of the available nodes.
• Pods are mortal: when a pod dies, it cannot be brought back to life.

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-world-pod
labels:
  app: my-app
  version: v1
spec:
  containers:
  - name: hello-container
    image: myapp/helloworld:v1
    ports:
    - containerPort: 8080
```
ReplicaSets

- Pods by themselves are not self-healing or scalable.
- For this reason, pods should be deployed with higher-level objects, like ReplicaSets or Deployments.
- ReplicaSets define two important things: a template for pods and the number of replicas.
- With ReplicaSets, your app can be scaled simply by updating a ReplicaSet manifest file.

```yaml
apiVersion: apps/v1
class: ReplicaSet
metadata:
  name: hello-world-rs
spec:
  replicas: 20
  selector:
    matchLabels:
      app: my-app
template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: hello-container
        image: myapp/helloworld:v1
        ports:
        - containerPort: 8080
```
Services

- Every pod has a unique IP address
- If a pod dies, a new one is created in its place, with a new IP address.
  - We cannot rely on pod IP addresses
- We can solve this problem with services: a service provides stable IP address, DNS name, and port.
- Services are dynamically associated with a set of pods based on label selector.
- Services redirect external connections to associated pods.

```yaml
apiVersion: v1
class: Service
metadata:
  name: hello-world-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 7777
```
Deployments

- Deployments are high-level objects that manage ReplicaSets.
- The main purpose of Deployments is to allow rolling updates and rollbacks – without causing any downtime.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  labels:
    app: my-app
spec:
  replicas: 5
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app-container
        image: my-app-container:v1
        ports:
        - containerPort: 8080
```