



Container orchestration: Kubernetes

Corso di Sistemi Distribuiti e Cloud Computing A.A. 2024/25

Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

Container orchestration

- Platforms for managing deployment of **multi-container packaged applications** in large-scale clusters
- Allow to configure, provision, deploy, monitor, and dynamically control containerized apps
 - Used to integrate and manage *containers at scale*
- Kubernetes is the most popular one
- Alternatives
 - [Docker Swarm](#)
 - [Nomad](#)

Container management systems at Google

- Application-oriented shift
 - “Containerization transforms the data center from being machine-oriented to being application-oriented”
- Goal: let container technology operate at Google scale
 - Everything at Google runs as a container
 - Google launches several billions of containers per week
- Predecessors of Kubernetes at Google
 - Borg and Omega: purely Google-internal systems, precede Kubernetes, see [Borg, Omega, and Kubernetes](#) paper

Kubernetes



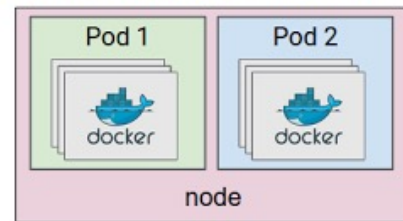
- Google's open-source platform for automating deployment, scaling, and management of containerized apps across clusters of hosts
 - <http://kubernetes.io>
 - **K8s**: name abbreviation
- Features:
 - **Portable**: public, private, hybrid, multi-cloud
 - **Extensible**: modular, pluggable, hookable, composable
 - **Self-healing** and **self-optimizing**: auto-placement, auto-restart, auto-scaling of containers
- Can run on public or private cloud platforms (e.g., AWS, OpenStack), and also on bare metal machines
- Offered as Cloud service by main Cloud providers
 - Kubernetes management and deployment on underlying infrastructure is up to Cloud provider

Pod

- **Pod**: smallest deployable compute object in Kubernetes
 - Set of (tightly coupled) containers with shared storage/network, and a specification for how to run the containers
 - Pod containers are bundled and scheduled together, and run in a shared context
 - K8s gives pods their own IP addresses and a single DNS name for a set of pods, and can load-balance across them
- Users organize pods using **labels**
 - Label: arbitrary key/value pair attached to pod
 - E.g., role=frontend and stage=production

Kubernetes Pods

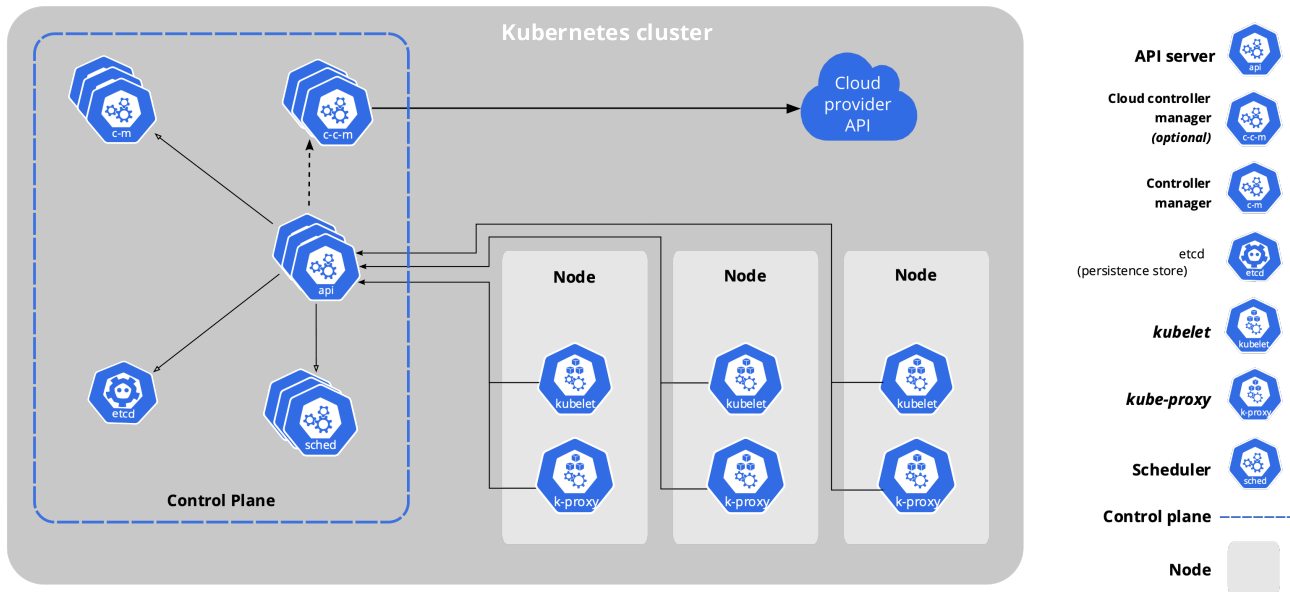
collections of containers that are co-scheduled



Architecture

- When you deploy Kubernetes, you get a **cluster**
- Kubernetes cluster: set of worker machines, called **nodes**, that run containerized applications
- Architecture organized according to **master-worker pattern**
- Every cluster has at least one worker node
- Worker nodes host pods
- Kubernetes supports multiple container runtimes, including **containerd** (used by Docker Engine)

Architecture



<https://kubernetes.io/docs/concepts/overview/components/>

Architecture: control plane

- **Kubernetes control plane:** cluster's **master**, takes **global decision** about cluster (e.g., scheduling) as well as detects and responds to cluster events (e.g., starting up a new pod)
 - In production environments, control plane usually runs across multiple machines for failover and high availability
- **Main components of control plane**
 - **kube-apiserver:** API server that exposes Kubernetes API, front end for Kubernetes' control plane
 - **kube-scheduler:** decides how to assign newly created pods to nodes using a **placement** (or **scheduling**) policy
 - **kube-controller-manager:** runs controller processes (e.g., **Node controller** which is responsible for noticing and responding when nodes go down)

Architecture: control plane

- Main components of control plane
 - [etcd](#): distributed key-value data store with strong consistency and high availability
 - Written in Go
 - Uses [Raft](#) consensus algorithm to manage a highly-available replicated log
 - How used in Kubernetes? Backing store for all cluster data



Architecture: nodes

- [Kubernetes nodes](#): [worker](#) nodes (can be VM or physical machines) that maintain multiple running pods and provide Kubernetes' runtime environment
- Main components on worker nodes
 - [kubelet](#): agent ensuring that pods running on node are healthy and running
 - [kube-proxy](#): network proxy that maintains network rules on nodes

Some Kubernetes terminology

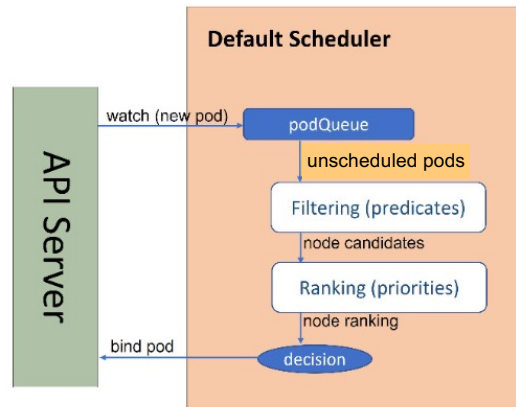
- **Workload**: containerized application running on Kubernetes
- **Deployment**: tells Kubernetes how to create or modify ipod nstances that hold a workload; deployments can help to:
 - efficiently scale number of replica pods
 - enable rollout of updated code in a controlled manner
 - roll back to earlier deployment version if necessary
- Kubernetes deployment:
 - described in YAML file
 - created using `kubectl apply`
 - `kubectl` is Kubernetes' CLI tool
 - Example:
kubernetes.io/docs/concepts/workloads/controllers/deployment/

Some Kubernetes terminology

- Each pod gets its own unique cluster-wide IP address, but in a deployment the set of pods can change at runtime (e.g., due to node crash or pod replication)
 - In a containerized application running with multiple pods, how to keep track of which IP address to connect to?
- **Service**: abstraction that defines a **logical set of pods** and a **policy** by which to access them to expose application running on a set of pods as a network service
 - Pod IPs are not exposed outside the cluster without a service

Pod placement

- *kube-scheduler*: default Kubernetes scheduler
kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/
- Scheduler watches for newly created pods and finds best node for each new pod
- Kubernetes' placement policy selects node in 2 steps: *filtering* and *scoring*



Valeria Cardellini - SDCC 2024/25

12

Pod placement

- Kubernetes' placement policy selects node in 2 steps: *filtering* and *scoring*
 1. *Filtering*: kube-scheduler pre-selects nodes that satisfy certain pod requirements based on predicates (defined by using node labels in pod definition, e.g., `disktype:ssd`)
 2. *Scoring*: kube-scheduler assigns a score to the remaining nodes that survived filtering using some rule (e.g., amount of free CPU and memory). The pod is bound to node with highest score (if more than one node, kube-scheduler selects one of these at random)
- Filtering and ranking stages can be configured
kubernetes.io/docs/concepts/scheduling-eviction/

Valeria Cardellini - SDCC 2024/25

13

Pod placement

- Other placement policies can be easily integrated into Kubernetes; two examples of heuristic policies:
 - *Round robin*: organize nodes in a circular list, saving the latest node used for scheduling; allocate each pod on next node with enough resources, starting from current position on list
 - *Greedy first fit*: popular heuristic used to solve [bin packing](#) problem; consider pods as items to be greedily allocated in bins, representing nodes; add nodes to a list and sort them in ascending order of available resources; place each pod into *the first bin* in which it will fit

Pod placement

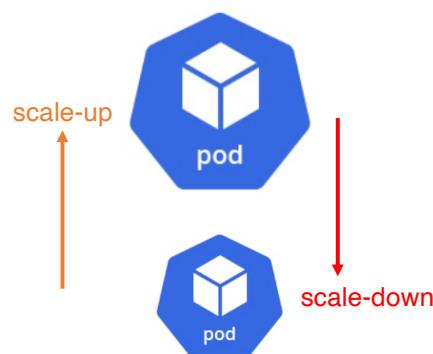
- Limitations of default placement policy and other heuristics
 - Not well-suited to place pods in geo-distributed environments with non-negligible network delays
 - Do not consider energy consumption
 - Do not take into account security requirements
- Opportunity for thesis topics!

Auto-scaling in Kubernetes

- **Multiple auto-scalers** at different control layers
 - Cluster auto-scaling with node granularity
 - Horizontal auto-scaling with pod granularity
 - Vertical auto-scaling with pod granularity
- Organized as **MAPE loops** github.com/kubernetes/autoscaler
- **Cluster Autoscaler**: adjusts size of Kubernetes cluster when one of the following conditions is true:
 - There are pods that failed to run in the cluster due to insufficient resources
 - There are nodes in the cluster that have been underutilized for an extended period of time and their pods can be placed on other existing nodes

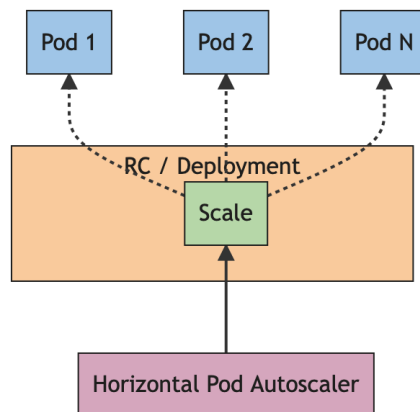
Auto-scaling: VPA

- **Vertical Pod Autoscaler (VPA)**: scales amount of pod resources (CPU, memory)
 - Based on historical resource usage of pods: decaying histogram of weighted CPU and memory usage
 - ✗ Whenever VPA updates pod resources, the pod is recreated: this disrupts the application availability



Auto-scaling: HPA

- **Horizontal Pod Autoscaler (HPA)**: scales number of pods in a deployment, replica set or stateful set
kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
 - Based on observed CPU utilization (or, with custom metrics support, on some other application-provided metrics)
 - Creates new pods without affecting existing ones



Valeria Cardellini - SDCC 2024/25

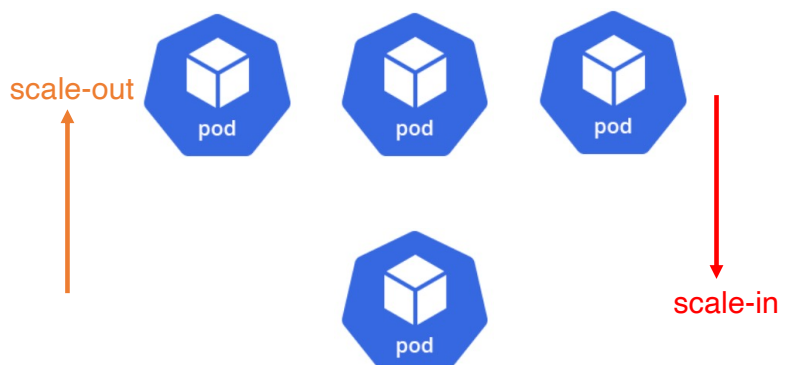
18

Auto-scaling: HPA

- HPA policy: heuristic policy, variant of threshold-based policy
 - Scales number of pods according to ratio between observed value and target value

$$desiredReplicas = \left\lceil currentReplicas \frac{currentMetricValue}{desiredMetricValue} \right\rceil$$

- Stabilization time window to limit fluctuations in replicas number



Valeria Cardellini - SDCC 2024/25

19

Distributions

- Multiple options
 1. [Vanilla Kubernetes](#): install K8s from source code
 2. [Pure distributions](#): pre-built K8s
 - E.g., [Charmed Kubernetes](#)
 3. [Plus distributions](#): platforms that integrate K8s with other specific technologies (e.g., container runtimes, host OSs or control-plane add-ons)
 - E.g., [Red Hat OpenShift](#), [Rancher](#)
 4. [Limited-purpose distributions](#): intended for specific and limited purpose (e.g., single-node, DevOps, edge & IoT)
 - E.g., [kind](#), [minikube](#), [MicroK8s](#), [K3S](#)
 5. [Kubernetes-as-a-service](#): K8s fully managed by Cloud provider
 - E.g., [AWS EKS](#), [Azure AKS](#), [Google GKE](#)

20

Some useful tools

- [kubectl](#): command-line tool to run commands against Kubernetes cluster
 - Use `kubectl` to deploy applications, inspect and manage cluster resources, and view logs
- [Metrics Server](#): scalable, efficient source of container resource metrics
 - Collects resource metrics from kubelets and exposes them in kube-apiserver through Metrics API for use by HPA and VPA
- [Helm](#): package manager for Kubernetes
 - Helps to automate Kubernetes applications lifecycle