

Middleware Architectures 1

Lecture 5: Cloud Native and Kubernetes

doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • <https://vitvar.com>



Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • <https://vitvar.com/lectures>



Modified: Mon Oct 27 2025, 06:03:23
Humla v1.0

Overview

- Cloud Native
- Kubernetes

Overview

- The Cloud Native Computing Foundation (CNCF)
 - *Motto: Building sustainable ecosystems for cloud native software*
 - *CNCF is part of the nonprofit Linux Foundation*
- Cloud Native = scalable apps running in modern cloud environments
 - *containers, service meshes, microservices*
 - *Apps must be usually re-built from scratch or refactored*
 - *Benefits:*
 - *loosely coupled systems that are resilient, manageable, and observable*
 - *automation allowing for predictable and frequent changes with minimal effort*
 - *Trail Map*
 - *provides an overview for enterprises starting their cloud native journey*
- Lift and Shift
 - *Cloud transition program in organizations*
 - *Move app from on-premise to the cloud*
 - *Benefits*
 - *Infrastructure cost cutting (OPEX vs. CAPEX)*
 - *Improved operations (scaling up/down if possible can be faster)*

CNCF Trail Map

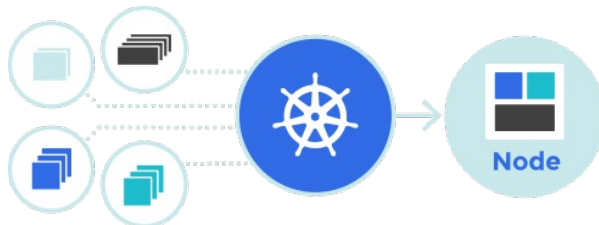


Overview

- Cloud Native
- Kubernetes
 - *Basic Concepts*
 - *Core Concepts and Architecture*
 - *Workloads*
 - *Services*
 - *Beyond the Basics*

Overview

- In your architecture...
 - *Containers are atomic pieces of application architecture*
 - *Containers can be linked (e.g. web server, DB)*
 - *Containers access shared resources (e.g. disk volumes)*
- Kubernetes
 - *Automation of deployments, scaling, management of containerized applications across number of nodes*
 - *Based on Borg, a parent project from Google*

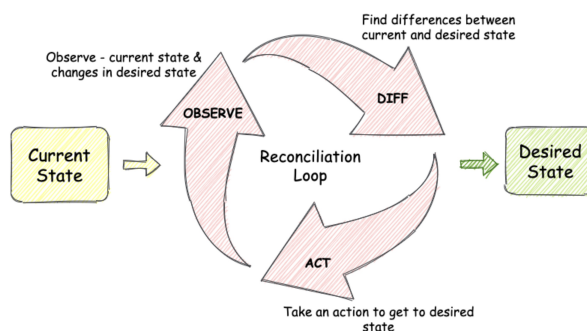


Key Design Principles

- Kubernetes provides abstractions that separate application deployment from the underlying infrastructure details
- Application workloads and infrastructure decoupling
 - **Compute:** Define what to run without specifying where it runs
 - **Storage:** Applications request storage independent of storage backend
 - **Networking:** Stable access to applications regardless of IPs or location
- Benefits
 - Portability across on-prem and cloud environments
 - Scalability and resilience through dynamic scheduling
 - Consistency and standardization of deployment model
 - Reduced vendor lock-in thanks to open standards

Desired State and Reconciliation

- Kubernetes operates on a **desired state** model
 - Users define the state they want through object specifications (YAML)
 - Example: “there should be 3 replicas of this application”
- Actual State vs. Desired State
 - Kubernetes constantly monitors the cluster
 - If the actual state drifts from the desired state, it takes action to fix it
- Reconciliation Loop
 - Controllers continuously compare desired vs. actual state
 - Automatically performs actions such as restarting, rescheduling, or scaling Pods



Features

- Automatic binpacking
 - *Automatically places containers onto nodes based on their resource requirements and other constraints.*
- Horizontal scaling
 - *Scales your application up and down with a simple command, with a UI, or automatically based on CPU usage.*
- Automated rollouts and rollbacks
 - *Progressive rollout out of changes to application/configuration, monitoring application health and rollback when something goes wrong.*
- Storage orchestration
 - *Automatically mounts the storage system (local or in the cloud)*
- Self-healing
 - *Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to user-defined health checks.*
- Service discovery and load balancing
 - *Gives containers their own IP addresses and a single DNS name for a set of containers, and can load-balance across them.*

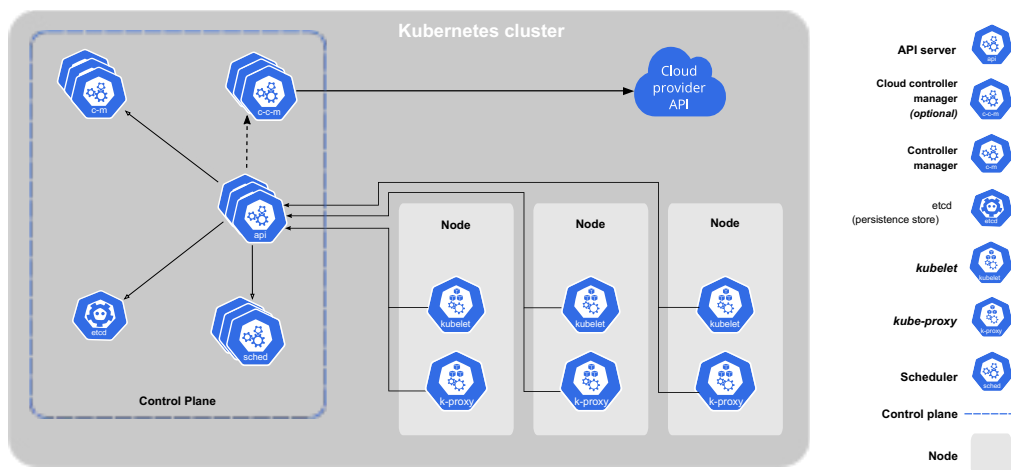
Overview

- Cloud Native
- Kubernetes
 - *Basic Concepts*
 - *Core Concepts and Architecture*
 - *Workloads*
 - *Services*
 - *Beyond the Basics*

Core Building Blocks

- **Cluster**
 - A set of worker nodes and a control plane
 - Runs and manages containerized applications
- **Node**
 - A worker machine in Kubernetes (VM or physical)
 - Runs Pods scheduled by the control plane
- **Control Plane**
 - Manages the overall state of the cluster
 - Schedules workloads and responds to cluster events
- **Pod**
 - The smallest deployable unit in Kubernetes
 - One or more tightly-coupled containers
 - Containers share networking and storage within a Pod

Architecture



Control Plane Components (Part 1)

- Global decisions about the cluster
 - *Scheduling*
 - *Detecting and responding to cluster events, starting up new pods*
- kube-apiserver
 - *exposes the Kubernetes API*
 - *The API server is the front end for the Kubernetes control plane.*
- etcd
 - *highly-available key value store used to store all cluster data*
- kube-scheduler
 - *watches for newly created Pods with no assigned node*
 - *selects a node for Pods to run on.*
 - *Decision factors: resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications*

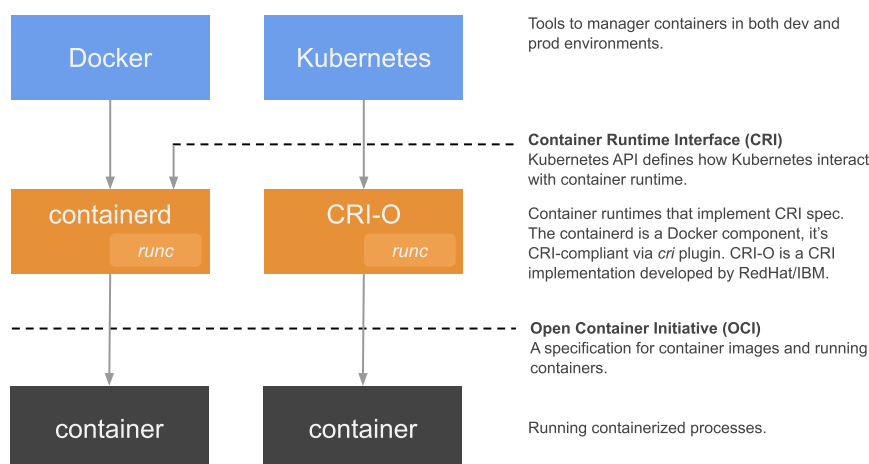
Control Plane Components (Part 2)

- kube-controller-manager
 - *runs controller to ensure the desired state of cluster objects*
 - **Node controller**
 - *noticing and responding when nodes go down*
 - **Job controller**
 - *creates Pods to run one-off tasks to completion.*
 - **Endpoints controller**
 - *Populates the Endpoints object (that is, joins Services, Pods).*
- cloud-controller-manager
 - *Integration with cloud services (when the cluster is running in a cloud)*
 - **Node controller**
 - *checks if a node has been deleted in the cloud after it stops responding*
 - **Route controller**
 - *For setting up routes in the underlying cloud infrastructure*
 - **Service controller**
 - *For creating, updating and deleting cloud provider load balancers*

Node

- Kubernetes runtime environment
 - *Run on every node*
 - *Maintaining running pods*
- kubelet
 - *An agent that runs on each node in the cluster*
 - *It makes sure that containers are running in a Pod.*
- kube-proxy
 - *maintains network rules on nodes*
 - *network rules allow network communication to Pods from inside or outside of the cluster*
 - *uses the operating system packet filtering layer or forwards the traffic itself.*
- Container runtime
 - *Responsible for running containers*
 - *Kubernetes supports several container runtimes (containerd, CRI-O)*
 - *Any implementation of the Kubernetes CRI (Container Runtime Interface)*

Container Stack



Overview

- Cloud Native
- Kubernetes
 - *Basic Concepts*
 - *Core Concepts and Architecture*
 - *Workloads*
 - *Services*
 - *Beyond the Basics*

Namespaces

- Logical grouping of cluster resources
 - *Allow you to organize and separate objects within a Kubernetes cluster*
 - *Useful when multiple teams, environments, or projects share the same cluster*
- Rationale
 - *Provide isolation and boundaries between workloads*
 - *Prevent name collisions*
 - *Objects can have the same name if in different namespaces*
 - *Enable resource limits and access control per namespace*
- Usage
 - *Common namespaces: default, kube-system, kube-public, kube-node-lease*
 - *Create separate namespaces for e.g. dev, test, prod*
 - *Commands run in a namespace unless another is specified*

Pod

- Pod
 - A group of one or more tightly-coupled containers.
 - Containers share storage and network resources.
 - A Pod runs a single instance of a given application
 - Pod's containers are always co-located and co-scheduled
 - Pod's containers run in a shared context, i.e. in a set of Linux namespaces
- Pods are created using workload resources
 - You do not create them directly
- Pods in a Kubernetes cluster are used in two main ways
 - Run a single container; the most common Kubernetes use case
 - Run multiple containers that need to work together

Workloads

- An application running on Kubernetes
- Workloads run in a set of Pods
- Pre-defined workload resources to manage lifecycle of Pods
 - **Deployment** and **ReplicaSet**
 - managing a stateless application workload
 - any Pod in the Deployment is interchangeable and can be replaced if needed
 - **StatefulSet**
 - one or more related Pods that track state
 - For example, if a workload records data persistently, run a StatefulSet that matches each Pod with a persistent volume.
 - **DaemonSet**
 - Ensures that all (or some) Nodes run a copy of a Pod
 - Such as a cluster storage daemon, logs collection, node monitoring running on every node
 - **Job** and **CronJob**
 - Define tasks that run to completion and then stop.
 - Jobs represent one-off tasks, whereas CronJobs recur according to a schedule.

Deployment Spec Example

- Deployment spec

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5  spec:
6    selector:
7      matchLabels:
8        app: nginx
9    replicas: 3 # tells deployment to run 3 pods matching the template
10   template:
11     metadata:
12       labels:
13         app: nginx
14     spec:
15       containers:
16         - name: nginx
17           image: nginx:1.14.2
18           ports:
19             - containerPort: 80
```

- *A desired state of an application running in the cluster*
- *Kubernetes reads the Deployment spec and starts three app instances*
- *If an instance fails, Kubernetes starts a replacement app instance*

Overview

- Cloud Native
- Kubernetes
 - *Basic Concepts*
 - *Core Concepts and Architecture*
 - *Workloads*
 - *Services*
 - *Beyond the Basics*

What is a Service?

- A Kubernetes **Service** is an abstraction that defines
 - A logical set of Pods
 - A policy to access them.
- Pods are ephemeral – their IPs change when recreated
- A Service provides a stable virtual endpoint for a set of Pods
- Services enable reliable communication between components:
 - Internal pods communication
 - External access to cluster workloads
- Each Service gets
 - A DNS name and
 - virtual IP (ClusterIP) inside the cluster.
- Kubernetes component **kube-proxy** manage routing to backend Pods.

Service Types

- **ClusterIP**
 - Exposes the Service on an internal IP in the cluster only.
 - Used for internal communication between Pods.
- **NodePort**
 - Exposes the Service on each Node's IP at a static port (e.g. 30080).
 - Accessible externally via **NodeIP:NodePort**.
- **LoadBalancer**
 - Provisions an external load balancer (e.g. in cloud environments).
 - Routes external traffic to the Service.
- **ExternalName**
 - Maps the Service to an external DNS name.
 - No proxying — pure DNS CNAME redirection.

How Services Work

- **Selector**

- A Service usually defines a **selector** — a label query used to find matching Pods.
- Example: **selector: app=nginx** matches all Pods with label **app=nginx**.
- Kubernetes monitors Pods that match this selector and updates Service backends

- **Endpoints / EndpointSlice**

- For every Service with a selector, Kubernetes creates an **Endpoints** (or **EndpointSlice**) object listing all healthy Pod IPs and ports.
- This list changes dynamically as Pods are added, removed, or become unhealthy.

- **kube-proxy**

- Runs on every Node and watches Service and Endpoint objects.
- Programs **iptables** or **IPVS** rules to forward traffic from the Service's virtual IP (**ClusterIP**) to one of the backend Pod IPs.
- Load balancing is done using round-robin or IPVS algorithms.

- **DNS Integration**

- **CoreDNS** automatically creates a DNS record for each Service:
 - **<service>.<namespace>.svc.cluster.local**
- Pods can reach the Service via DNS without knowing Pod IPs

Packet Forwarding and Load Balancing

- **iptables Mode**

- **kube-proxy** creates NAT rules in the **nat** table to redirect Service traffic.
- Packets sent to the Service's **ClusterIP:Port** are sent (via **DNAT**) to a Pod IP.
- Example

```
1  # Match packets to the Service ClusterIP:Port
2  -A KUBE-SERVICES -d 10.96.0.10/32 -p tcp -m tcp --dport 80 \
3    -m comment --comment "default/my-service: cluster IP" \
4    -j KUBE-SVC-XYZ123
5
6  # Choose a backend Pod pseudo-randomly (rough round robin)
7  -A KUBE-SVC-XYZ123 -m statistic --mode random --probability 0.5 \
8    -j KUBE-SEP-A1B2C3
9  -A KUBE-SVC-XYZ123 -j KUBE-SEP-D4E5F6
10
11 # DNAT to actual Pod IP and port
12 -A KUBE-SEP-A1B2C3 -s 10.42.0.0/16 -j KUBE-MARK-MASQ
13 -A KUBE-SEP-A1B2C3 -p tcp -m tcp -j DNAT --to-destination 10.42.0.12:8080
14
15 -A KUBE-SEP-D4E5F6 -s 10.42.0.0/16 -j KUBE-MARK-MASQ
16 -A KUBE-SEP-D4E5F6 -p tcp -m tcp -j DNAT --to-destination 10.42.1.7:8080
```

Example: ClusterIP Service

- Example configuration exposing an NGINX Deployment internally:

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-svc
5  spec:
6    selector:
7      app: nginx
8    ports:
9      - protocol: TCP
10       port: 80
11       targetPort: 8080
12     type: ClusterIP
```

- Pods with `app=nginx` receive traffic through `ClusterIP`.
- DNS name: `nginx-svc.default.svc.cluster.local`
- Used by other Pods to connect via `http://nginx-svc:80`.

Overview

- Cloud Native
- Kubernetes
 - *Basic Concepts*
 - *Core Concepts and Architecture*
 - *Workloads*
 - *Services*
 - *Beyond the Basics*

Advanced Topics

- Custom APIs and Controllers
 - CRDs, Operators, reconciliation loops
 - Admission webhooks (mutating/validating)
- Security
 - RBAC, Namespaces, Pod Security (seccomp, capabilities, rootless)
 - Image signing and supply chain (SBOM, cosign), Secret management (Vault/CSI)
 - Policy engines: OPA Gatekeeper, Kyverno
- Networking
 - CNI, eBPF (Cilium), NetworkPolicies, Ingress
 - Gateway API, Service Mesh (mTLS, traffic shaping)
- Storage
 - CSI drivers, snapshots, expansion, topology-aware PVs
 - Backup/DR (e.g., Velero), StatefulSet patterns
- Scaling and Scheduling
 - HPA/VPA/KEDA (event-driven), Cluster Autoscaler
 - Affinity/anti-affinity, taints/tolerations, topology spread
- Ops and Delivery