This course provides insight for non-technical stakeholders into Cloud-native computing, Microservices Architecture, Containerization (such as Docker), and Kubernetes. Its goal is to make you understand what they entail, what necessitated them, and the value they can add to your company.

Containerization is one of the most important developments in the industry, and its impact is far-reaching. It is hard to imagine what we, as an industry, would have done if it had not come at the right time.

After its inception, containerization presented a new set of problems that needed to be addressed. This is where, as an IT community, we introduced solutions such as Docker Compose, Docker Swarm, and eventually Kubernetes. Later on, the term cloud-native computing was coined to cover all the above technologies.

The Problem Definition

The long road to MSA

In this first part, we look at the current state of our IT Architecture and dive into how we got here by laying out the gradual evolutionary process. We started this road at the tail-end of the eighties, just before application-centric. This section will cover the following:

- The benefits and challenges of **client/server** architecture
- The move to **application centric** architectures
- the birth of **services** and **components**
- Values and problems of **Component-Based Design**
- The force of **low coupling** (one of the drivers for architecture)
- The era of **Enterprise Application Integration** (EAI) and **Integration Patterns** (EIP)
- **Event-Driven** and **Message-Oriented** architectures
- The rise of the **Service-Oriented Architecture**
- Promises made by **Enterprise Service Bus** (ESB) and the fall of SOA
- the dawn of **Micro Service Architecture** (MSA)
- Values and characteristic of the MSA

Technical revolutions and eventual freedom

In this section, we take a look at a second driver/dimension that led us to where we are and how it led to the birth as well as the immense popularity of Docker. This journey starts during the height of application-centric architecture in the mid-nineties. Here, we'll cover the following:

- From **CORBA** (Common Request Broker Architecture) to the rule of Java Enterprise
- The ages of the (Java) **Application Server** (WebSphere, WebLogic, JBoss, Glassfish, Tomcat, OC4J)
- The introduction of **deployable components**
- Finally, competition, Microsoft joins with **.Net**
- The value for **IT and Operations**

- The stifling effect on IT Projects
- The rise of (multiple) **technology stacks**
- The **technology matrix hell**
- The need for more agility and **constrained freedom**
- Redefinition of "separation of concerns"

Containerisation

In this part, we explain what containerization is and how it solves the problems we have discussed so far. Here is what we'll cover:

- Recap of the **problems** solved by Containerisation
- What **embodies** a container ("what's in it"?)
- The role of **Virtualization** and its role in containerised environments
- The definition of **containers** and **images** as well as the relation between the two
- **Docker-compose** and the depleted soil of **Docker Swarm**
- The required **infrastructure** for images
- The impact of the containerization on the **software delivery processes**
- **Risk mitigated** by Containerisation
- Containerisation, MSA, and a call for **DevOps**

The Crucial Role of Kubernetes

MSA and containerization have their own challenges when used with traditional IT application platforms and deployment strategies. This is where Kubernetes comes in. In this section, we'll cover the following:

- **Challenges** addressed by Kubernetes
- What is **Orchestration**?
- Choosing between **on-premises** or **cloud-based** Kubernetes clusters
- A discussion on **cluster sizes**
- **Security** in Kubernetes
- Cluster Observability and **monitoring**
- Cluster **logging** (ELK/Elastic Stack, Kibana, Loki, Grafana)
- Resource and **performance monitoring** (Prometheus, Grafana)
- A quick rundown of popular tools (**helm**, **Kustomize**, …)
- A discussion on DevOps and **Gitops**
- The possible need and role of a **Service Mesh**

Standardisation

Lastly, we'll cover the following crucial enablers:

- Appreciate the Open Container Initiative (**OCI**)
- The role of the **Linux Foundation**
- Container runtime abstraction: **CRI** (Container Runtime Interface)

- Container Network Interface (CNI)