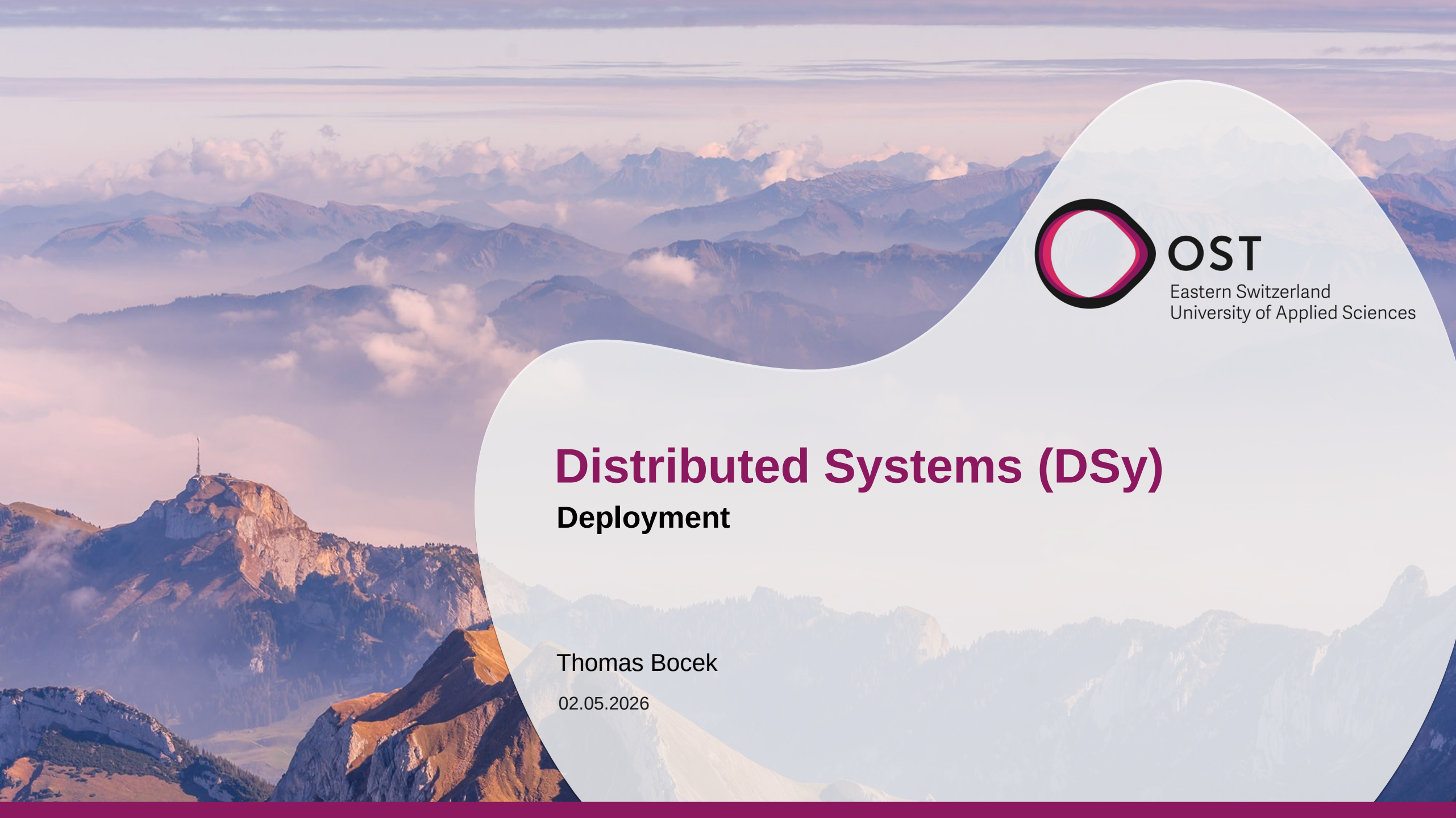


OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems (DSy)

Deployment

Thomas Bocek

02.05.2026

Learning Goals

- Lecture 11 (Deployment)
  - Different ways to deploy your service
    - High-level overview
  - More details
    - Cloud Infrastructure [link] - Norwin Schnyder
    - Cloud Operations [link] - Jan Untersander
    - Cloud Solutions [link] - Prof. Mirko Stocker

Back in the old days…

- OTS: apt-get / yum / pacman install package, e.g., Apache – configure – run
- Custom SW in the old days: Java WAR with Application Server, today: Fat JAR (e.g., Spring Boot)
- Custom /etc/init.d script with binary or script
- Problem:
  - It runs on my machine, who installs Java in the right version?
  - Dependency hell, two apps, different library versions
  - Configuration drift, what was changed manually?
  - What happens on crashes? HW defect?
  - Scaling?
  - Misconfiguration - access to complete PC?

- VMs / Containers help a lot
  - No access to complete PC, can scale, move to another machine, pre-install the right Java version
  - Each app brings its own libraries → no dependency hell
  - Immutable images → no configuration drift
- The new way: based on containers
  - How to deploy?
  - Just copy container to prod, done?
  - Many, many strategies...

OST

Deployment Strategies

- Many strategies and variations [link, link, link]
- Rolling Deployment
  - New version is gradually deployed to replace the old version - without taking the entire system down at once

    + Minimal downtime, low risk

    - Complexity, longer deployment times
- Blue-Green Deployment
  - 2 environments, current prod (blue), current prod with new release (green). Test, then switch

    + Instant rollback, 0 downtime

    - 2 prod environments, keep data in sync
- Canary Releases
  - Canary in a coal mine - new version to a small group of users or servers first, if all goes well, more users

    + Risk reduction, user feedback

    - Complexity, inconsistencies
- Feature Toggle
  - Fine grained canary, set feature for specific users

    + More risk reduction, specific user feedback

    - Increase complexity of codebase, config management
- Big Bang
  - Deploy everything at once

    + Simple

    - High risk, limited rollback

Practical Deployment

- Containerization as basis
  - Ansible (Progress Chef, Puppet) - and more
    - Playbooks with ssh host list – your host should run the same OS (apt/yum)
  - Docker Swarm
    - Works with docker-compose.yml – with docker you package your application the same way on any platform - simple
  - Kubernetes
    - Widespread
  - Plain docker (compose) / podman
    - Simple

```
#!/usr/bin/env bash

export DOCKER_HOST="ssh://user@host"
docker compose up -d --build
```

- Ansible (intro)
  - No agents running (unlike Progress Chef, Puppet)
  - Push-based system
  - ssh host list
  - Playbook
  - Run it: `ansible-playbook playbook.yml`

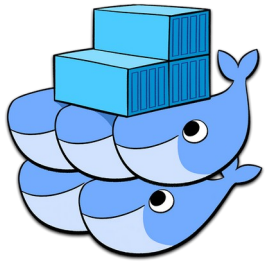
```
[webservers]
mwivmweb01
mwivmweb02
```

- More basic:
  - pssh / ssh

```
---
- name: Playbook
  hosts: webservers
  become: yes
  become_user: root
  tasks:
    - name: ensure apache is at the latest version
      yum:
        name: httpd
        state: latest
    - name: ensure apache is running
      service:
        name: httpd
        state: started
```

Podman / Docker Swarm

- Podman is daemonless
  - Simpler, but deployment needs more work [link]
  - Quadlet
    - Run container under systemd in a declarative way
    - Tools to convert podman/compose [link]
    - Auto-update from registry built-in [link]
- Many variations, tools, helpers: podman-compose [link]
- Opinion: I'm still using docker / docker-compose: daemon is awesome for deployments, docker-compose for local development works quite well

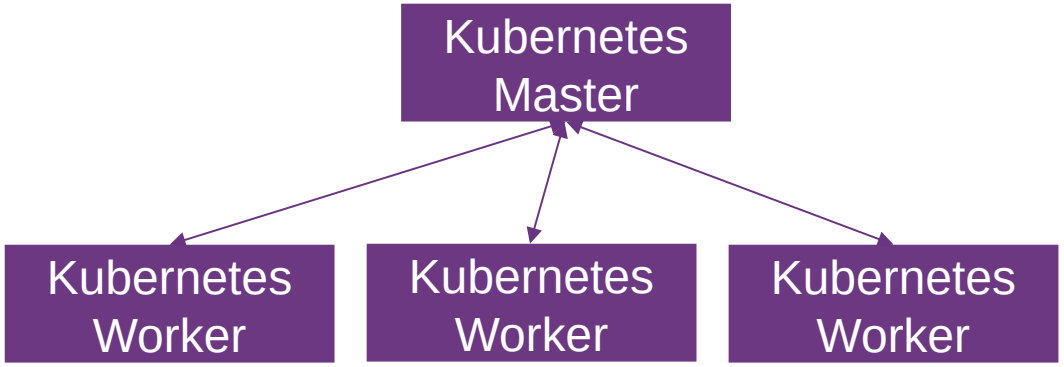


- Docker Swarm
  - Deploy with docker-compose.yml (deploy)
  - Built into docker
    - docker swarm – manage swarm
    - docker stack – manage deployments
    - docker node – manage nodes
  - Scheduler is responsible for placement of containers to nodes
  - Can use the same files, easy to setup?
    - Azure, Google cloud, Amazon
    - Deprecated...

OST

Kubernetes

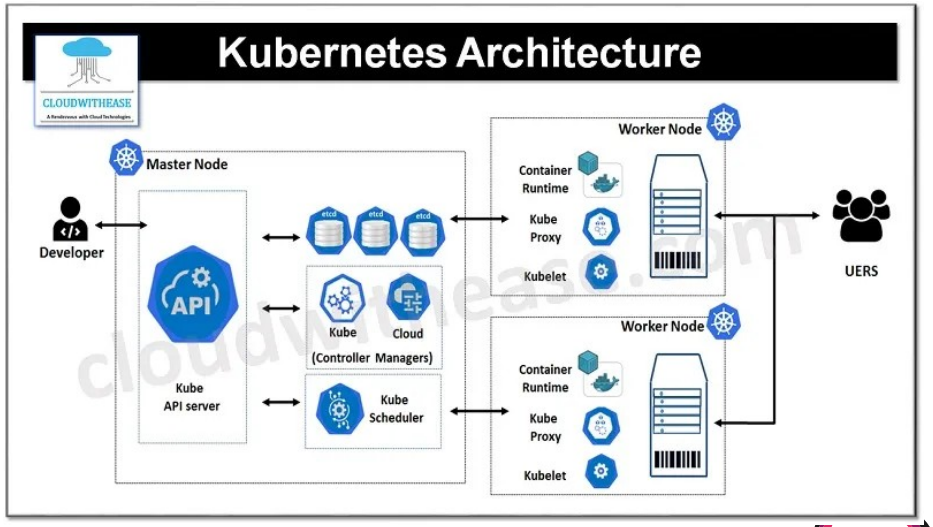
- What is Kubernetes (K8s)?
  - Container orchestration: deployment, scaling, management
  - Started by Google 2014, now CNCF
  - Industry standard for complex applications
- Kubernetes-based PaaS: Google, Amazon, Azure (book), Digital Ocean, ... – pricing schemes can be tricky
- Why Kubernetes?
  - High availability, fault tolerance (containers/nodes can crash)
  - Auto-scaling based on demand
  - Rolling updates and rollbacks built-in
  - Ecosystem: Helm (package manager), operators, …
- Design principles
  - Declarative config (YAML/JSON) – describe desired state
  - Immutable containers – don't store state, restart on health check fail
  - Rollbacks possible, but tricky with schema changes
- Architecture: Master + Worker nodes
  - Master: API Server, etcd, Controller Manager, Scheduler
  - Worker: kubelet, kube-proxy, container runtime (containerd, CRI-O)



Kubernetes

- Key Concepts
  - Pod: smallest deployable unit, one or more containers
  - Service: stable network endpoint for a set of pods
  - Deployment: manages desired state, scale, HW limits
  - ConfigMap: non-sensitive configuration data
  - Secret: sensitive data (passwords, API keys), encrypted in etcd
  - Volume: persistent storage, decoupled from pod lifecycle
  - Namespace: segment cluster, multiple projects/teams isolated
  - Full list: [link]

- Getting Started
  - Local development: Minikube, k3s
  - kubectl: command-line tool
  - GUI: Kubernetes Dashboard, Lens
  - Docs: kubernetes.io/docs, kubernetes.io/training, Youtube course
  - Foundation for any deployment: containerization
  - Deeper at OST: Cloud Infrastructure, Cloud Operations, Cloud Solutions



Source: https://cloudwithease.com/what-is-kubernetes/

OST

Services, Terraform

- Services
  - AWS, App Runner [link]
  - Digital Ocean, App Platform [link]
  - Railway [link]
    - Git based deployment
    - Zero config and HTTPS support
    - Integrated Dbs
    - Pull Request → Preview
    - Logs / Monitoring

- Ideally change a single line to deploy your application with another provider
- Terraform
  - IaaS description
- Deployment Best Practices
  - Automate the deployment as much as possible
  - Infrastructure as Code
  - Immutable Infrastructure
  - Health Checks / Monitoring
  - Centralized Logging
  - Rollback