



OST

Eastern Switzerland
University of Applied Sciences

Distributed Systems (DSy)

Deployment

Thomas Bocek

10.05.2025

Learning Goals

- Lecture 12 (Deployment)
 - Different ways to deploy your service
 - High-level overview
 - More details
 - Cloud Infrastructure [[link](#)] - Norwin Schnyder
 - Cloud Operations [[link](#)] - Dr. Josef Hammer
 - 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**, provide custom /etc/init.d script with binary or script
- Problem:
 - It runs on my machine, who installs Java in the right version?
 - What happens on crashes?
 - Scaling?
 - HW defect?
 - 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
- The new way: based on containers
- How to deploy?
 - Just copy container to prod, done?
 - Many, many strategies...

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
 - Which to use? [\[link\]](#)
 - Kubernetes
 - Widespread
 - Plain docker / podman
 - Simple

- 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
```

Docker / Podman

- Use docker --context to run/maintain containers on other machines
 - One of my super simple deployment scripts

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

export DOCKER_HOST="ssh://xyz@192.168.1.2"

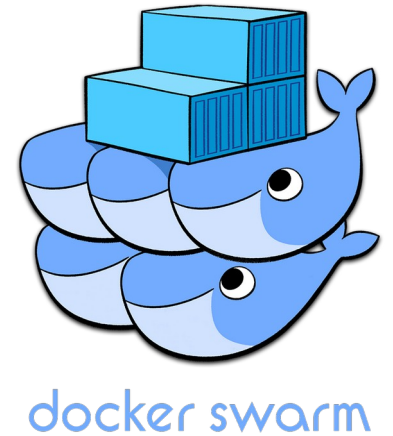
docker compose build
docker-compose up -d
```

- Copy files into image works as docker sends this file from local machine to the remote. Mount files does not do this
- No scaling, no logging, no resource monitoring, but simple

- Podman is daemonless
 - Simpler, but deployment needs more work [\[link\]](#)
 - **Quadlet**
 - Run container under systemd in a declarative way
 - Use another container config file to create a systemd config file
 - Use another project to create a container config from a podman command [\[link\]](#)
 - But, running upgrading images works seamless [\[link\]](#)
- 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

- 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...



- [Kubernetes vs. Docker Swarm](#)
- “Docker Swarm has already lost the battle against Kubernetes for supremacy in the container orchestration space” [[link](#)]
- “Kubernetes supports higher demands with more complexity while Docker Swarm offers a simple solution that is quick to get started with.” [[link](#)]

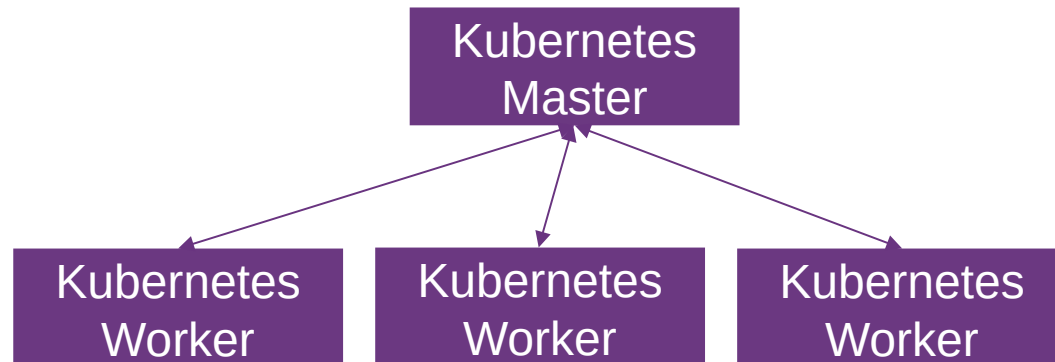


Kubernetes

- What is **Kubernetes** (K8s)
 - Container orchestration
 - Automates deployment, scaling, and management of containerized applications
 - Started by Google in 2014, now with **CNCF**
 - Widely adopted in the industry for managing complex applications
- Kubernetes-based PaaS
 - **Google, Amazon, Azure (book), Digital Ocean, ...**
 - **Difficult pricing schemes**
- Why Kubernetes?
 - Simplifies application deployment and management
 - Development: run on one machine, deployment how and where to distribute?
 - Ensures high availability and fault tolerance
 - Containers can crash, machine that runs container can crash (e.g., out of memory)
 - Supports auto-scaling based on demand
 - Facilitates rolling updates and rollbacks
 - Rollbacks are hard, especially with state, stateless rollback is easier
 - Provides a powerful ecosystem of tools and services
 - Package manager **Helm** released in 2016 (**convert docker-compose**)

Kubernetes

- Design principles
 - Configuration is declarative – declare state with YAML/JSON
 - Immutable containers
 - Don't store state in a container. If a health check fails, Kubernetes removes the container and starts a new one
 - Rollback applications, use older version of container – may need to change schema



- Architecture
 - Master Node: Controls the overall state of the cluster
 - API Server: Manages communication within the cluster
 - etcd: Stores configuration data for the cluster
 - Controller Manager: Ensures the desired state of the cluster
 - Scheduler: Assigns workloads to worker nodes
 - Worker Node: Runs application containers
 - kubelet: Communicates with the master node and manages containers
 - kube-proxy: Handles network routing and load balancing
 - Container runtime: Executes containers (containerd, CRI-O, etc.)

Kubernetes

- Key Concepts [[link](#)]

- **Pod**: Smallest deployable unit, contains one or more containers
- **Service**: Stable network endpoint to expose a set of Pods
- **Deployment**: Manages the desired state of an application, define scale, HW limits
- **ConfigMap**: Stores non-sensitive configuration data for an application
- **Secret**: Stores sensitive configuration data, like passwords and API keys

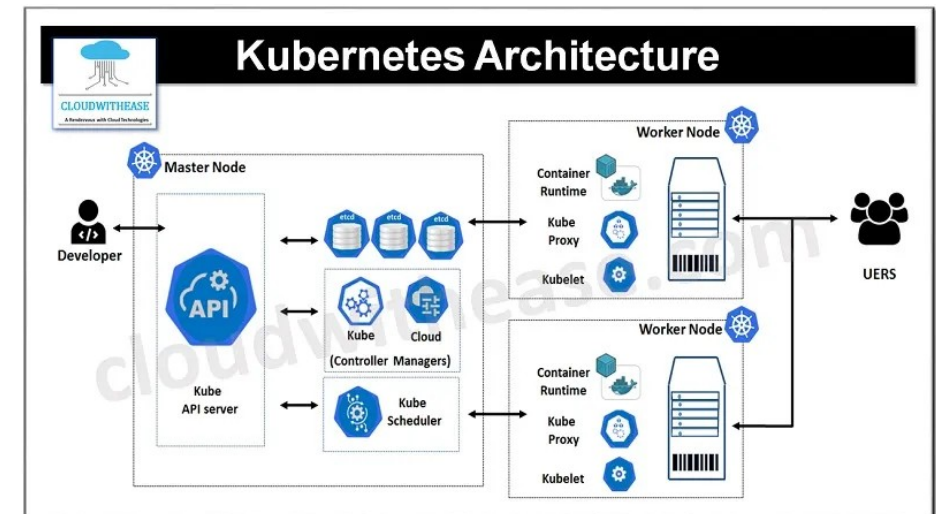
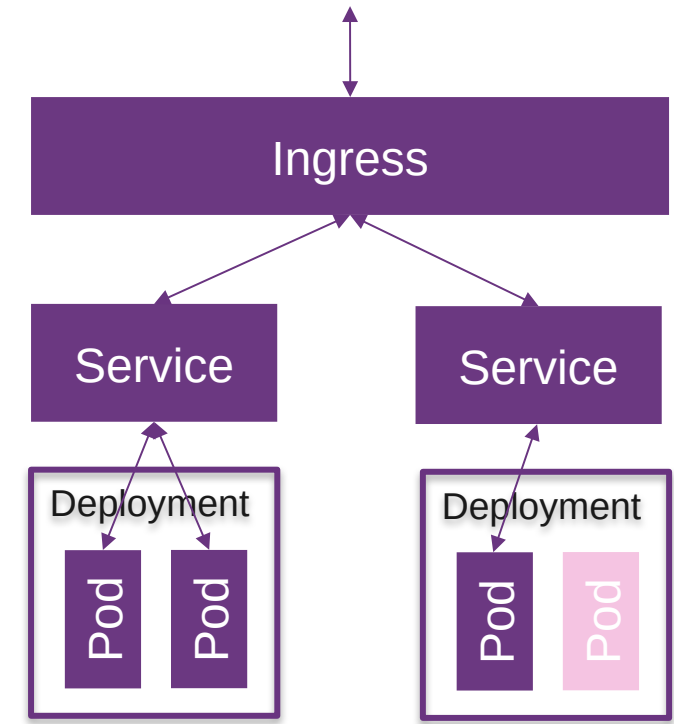
- **Volume**: Persistent storage for data generated by a container
- **Namespaces** – run multiple projects on one cluster, separate with namespaces

▼ Concepts

- ▶ Overview
- ▶ Cluster Architecture
- ▶ Containers
- ▶ Windows in Kubernetes
- ▶ Workloads
- ▶ Services, Load Balancing, and Networking
- ▶ Storage
- ▶ Configuration
- ▶ Security
- ▶ Policies
- ▶ Scheduling, Preemption and Eviction
- ▶ Cluster Administration
- ▶ Extending Kubernetes

Kubernetes

- Getting Started with Kubernetes: [Minikube](#), [k3s](#)
 - Minikube: Run a single-node Kubernetes cluster locally
 - kubectl: Command-line tool for managing a Kubernetes cluster
 - Kubernetes Dashboard: Web-based user interface for managing a cluster, [Lens](#)
- Official documentation: <https://kubernetes.io/docs>
- Kubernetes tutorials: <https://kubernetes.io/training>
- [Youtube course](#)



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
 - Rollback