

# Modul CNT

## Continuous Integration / Delivery mit Kubernetes

September 2021

Marcel Bernet

Dieses Werk ist lizenziert unter einer  
[Creative Commons Namensnennung - Nicht-kommerziell -  
Weitergabe unter gleichen Bedingungen 3.0 Schweiz Lizenz](https://creativecommons.org/licenses/by-nc-sa/3.0/de/) /



# Lernziele

- ★ Sie haben einen Überblick über Continuous Integration / Delivery, mit gängigen Tools und Kubernetes.



# Zeitlicher Ablauf

- ★ Continuous Integration (CI) und Continuous Delivery (CD) Definition
- ★ Continuous Integration (CI)
  - Docker
  - GitHub/GitLab
- ★ Continuous Delivery (CD)
  - Kubernetes
- ★ Auftrag für Praktische Arbeit (**zählt für die Modulnote**)

# Continuous Integration und Delivery - Definition

- ★ **Continuous Integration (CI)** (auch *fortlaufende* oder *permanente Integration*) ist ein Begriff aus der Software-Entwicklung, der den Prozess des fortlaufenden Zusammenfügens von Komponenten zu einer Anwendung beschreibt.
- ★ **Continuous Delivery (CD)** bezeichnet eine Sammlung von Techniken, Prozessen und Werkzeugen, die den Softwareauslieferungsprozess (englisch: Delivery) verbessern.

# Continuous Integration mit Docker


**Dockerfile**

 208 Bytes

```

1  FROM golang:1.16-alpine AS build
2
3  WORKDIR /src/
4  COPY *.go go.* /src/
5  RUN CGO_ENABLED=0 go build -o /bin/birdpedia
6
7  FROM scratch
8  COPY --from=build /bin/birdpedia /bin/birdpedia
9  COPY assets/index.html /assets/
    
```

```

1  FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-buster-slim AS base
2  WORKDIR /app
3  EXPOSE 80
4  FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build
5  WORKDIR /src
6  COPY ["DockerDemo.csproj", ""]
7  RUN dotnet restore "./DockerDemo.csproj"
8  COPY . .
9  WORKDIR "/src/."
10 RUN dotnet build "DockerDemo.csproj" -c Release -o /app/build
11 FROM build AS publish
12 RUN dotnet publish "DockerDemo.csproj" -c Release -o /app/publish
13 FROM base AS final
14 WORKDIR /app
15 COPY --from=publish /app/publish .
16 ENTRYPOINT ["dotnet", "DockerDemo.dll"]
    
```

- ★ Im Dockerfile sind mehrere **FROM** Einträge möglich, sogenannte multi-stage builds.
- ★ Das kann benutzt werden um:
  - Ein Base Image mit Compiler, Build Tools etc. zu starten und die Sourcen zu builden
  - Von einem neuen Base Image abzuleiten, z.B. mit dem Runtime der entsprechenden Sprache.
  - Das gebildete Programm vom ersten Container Image in das zweite Container Image zu kopieren.
- ★ <https://gitlab.com/mc-b/birdpedia/-/tree/master>
- ★ <https://codefresh.io/docker-tutorial/docker-images-net-core/>
- ★ **Hinweis:** Visual Studio Enterprise verwendet diese Variante auch.

# GitHub und CI Beispiele

Aktueller Status CI

Prüfung der Test-  
Abdeckung

Prüfung Code  
Qualität

build passing coverage unknown codefactor A

## README.md (Markdown Code)

```
[[Build Status](https://travis-ci.org/mc-b/SCS-ESI.svg?branch=master)](https://travis-ci.org/mc-b/SCS-ESI)
[[Coverage Status](https://coveralls.io/repos/github/mc-b/SCS-ESI/badge.svg)](https://coveralls.io/github/mc-b/SCS-ESI)
[[CodeFactor](https://www.codefactor.io/repository/github/mc-b/scs-esi/badge)](https://www.codefactor.io/repository/github/mc-b/scs-esi)
```

## SCS ESI Sample

<https://github.com/mc-b/SCS-ESI>

CI Konfiguration: `.travis.yml`

## Visual Studio Code - Open Source ("Code - OSS")

Azure Pipelines failed feature-request issues 2.6k open bug issues 869 open chat on gitter

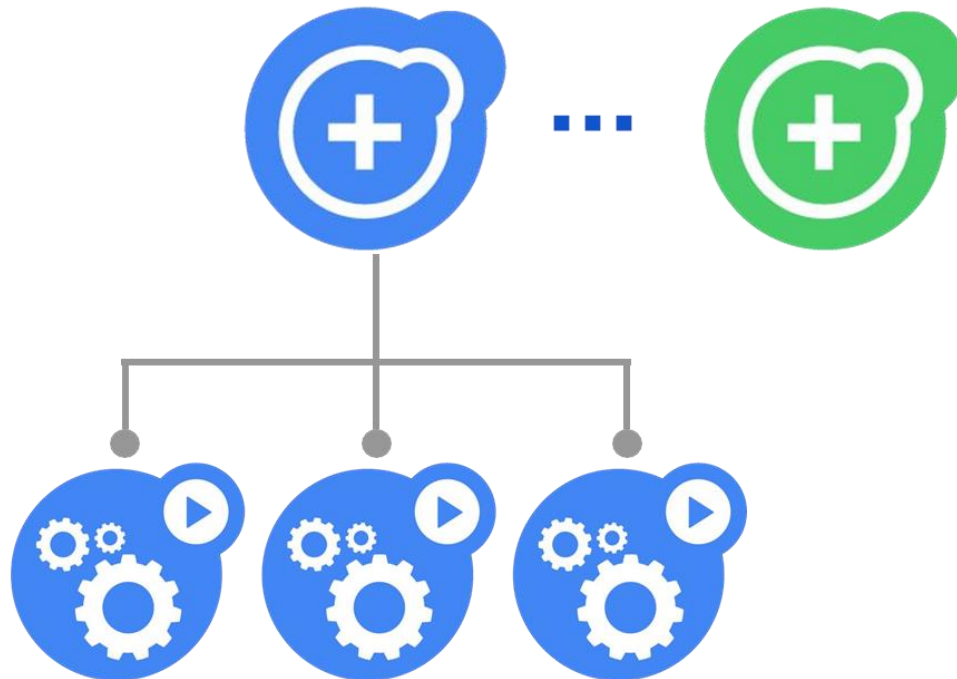
<https://github.com/microsoft/vscode>

CI Konfiguration: `azure-pipelines.yml`

# Deployment

## Deployment

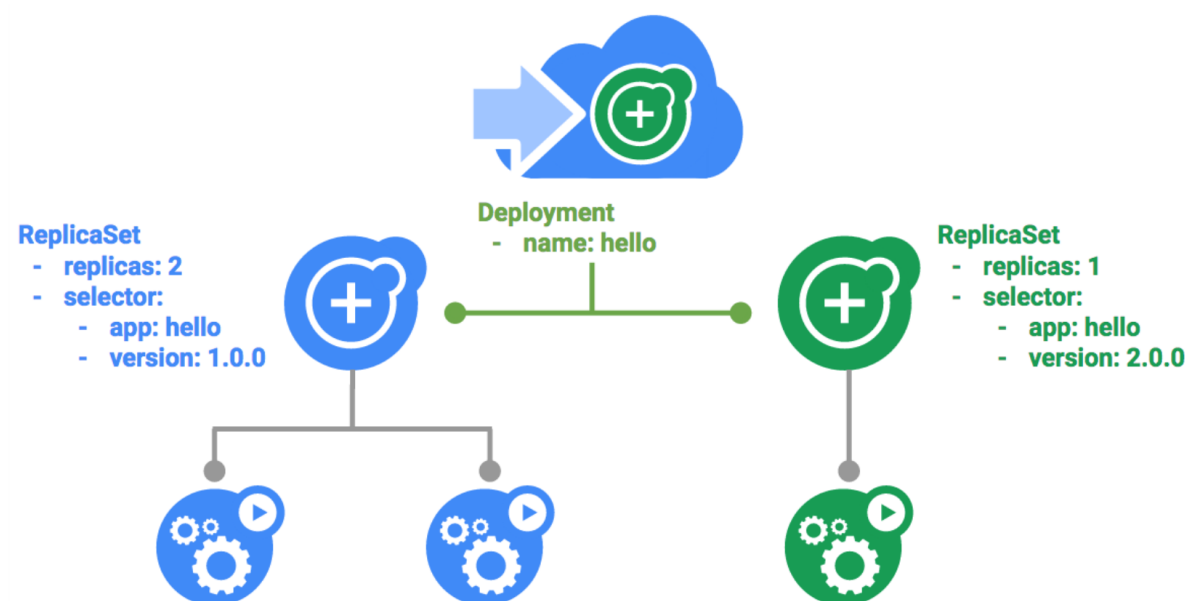
- **strategy: {type: RollingUpdate}**
- **replicas: 3**
- **selector:**
  - **app: my-app**



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bpmn-frontend
spec:
  replicas: 5
  selector:
    matchLabels:
      app: bpmn-frontend
  template:
    metadata:
      labels:
        app: bpmn-frontend
        group: web
        tier: frontend
    spec:
      containers:
        - name: bpmn-frontend
          image: misegr/bpmn-frontend:latest
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
              name: bpmn-frontend
```

- ★ Deployment eines Services (App)
  - Ersetzt ein Container Image im Pod
  - Ausgerollt durch ReplicaSet
- ★ Ermöglicht Deklarative Updates (Deployments)
  - Erzeugt und Löscht (nach Update) automatisch ReplicaSet und Pods.

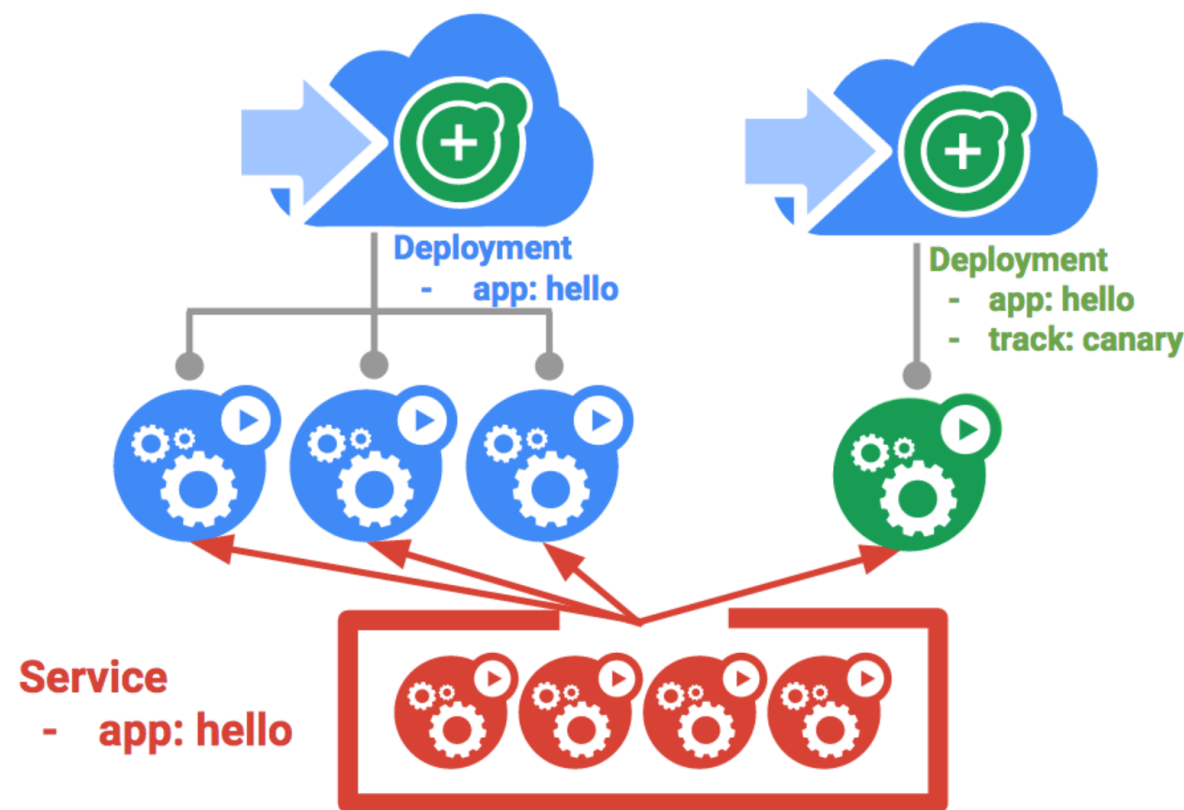
# Rolling Update



- ★ **Deployments** unterstützen das **Aktualisieren** von **Image** auf eine neue Version mithilfe eines fortlaufenden Aktualisierungsmechanismus.
- ★ Wenn ein Deployment mit einer neuen Version aktualisiert wird, erstellt es ein neues ReplicaSet und erhöht kontinuierlich die Anzahl der Replikat im neuen ReplicaSet, da die Replikat im alten ReplicaSet verringert werden.

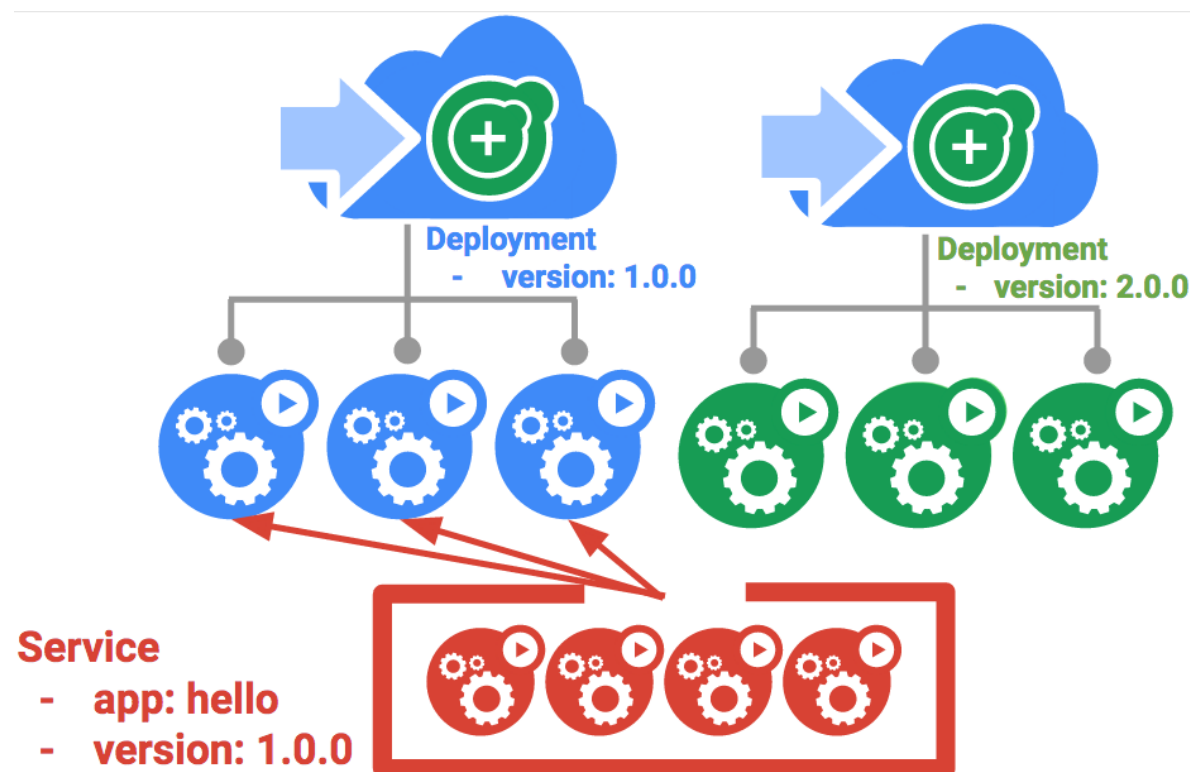


# Canary Deployments



- ★ Wenn Sie eine neue SW Version in der Produktion mit einer **kleinen Gruppe** Ihrer Benutzer testen möchten, können Sie ein «Canary» Deployment durchführen.
- ★ Bei «**Canary**» Deployment können Sie eine Änderung nur einer **kleinen Gruppe** Ihrer Benutzer **freigeben**, um das Risiko von neuen Versionen zu verringern.
- ★ **Lösung:** zweites Deployment, welches nur die Anzahl Canary Pods beinhaltet. Der Service steuert auf beide Deployments zu.

# Blue/Green Deployments



- ★ Rolling-Updates sind ideal, da sie Ihnen ermöglichen, eine Anwendung langsam mit minimalem Overhead, minimalen Auswirkungen auf die Leistung und minimalen Ausfallzeiten bereitzustellen.
- ★ Es gibt jedoch Fälle, in denen es sinnvoll ist, die Load Balancer so zu ändern, dass sie **erst** nach der **vollständigen Verteilung** auf die **neue Version** verweisen.
- ★ In diesem Fall sind so genannte Blue-Green-Bereitstellungen der richtige Weg.
- ★ **Lösung:** zweites Deployment und ein Service welcher erst nach dem vollständigen Erstellen der Pods auf Version 2.0.0 umgestellt wird.

# Übungen: Rolling Update, Blue/Green, Canary

## ★ Rolling Update

- [https://gitlab.com/ch-tbz-hf/Stud/cnt/-/blob/main/2\\_Unterrichtsressourcen/K/jupyter/09-4-Deployment.ipynb](https://gitlab.com/ch-tbz-hf/Stud/cnt/-/blob/main/2_Unterrichtsressourcen/K/jupyter/09-4-Deployment.ipynb)

## ★ Blue/Green Deployment

- [https://gitlab.com/ch-tbz-hf/Stud/cnt/-/blob/main/2\\_Unterrichtsressourcen/K/jupyter/09-4-Deployment-BlueGreen.ipynb](https://gitlab.com/ch-tbz-hf/Stud/cnt/-/blob/main/2_Unterrichtsressourcen/K/jupyter/09-4-Deployment-BlueGreen.ipynb)

## ★ Canary Deployment

- [https://gitlab.com/ch-tbz-hf/Stud/cnt/-/blob/main/2\\_Unterrichtsressourcen/K/jupyter/09-4-Deployment-Canary.ipynb](https://gitlab.com/ch-tbz-hf/Stud/cnt/-/blob/main/2_Unterrichtsressourcen/K/jupyter/09-4-Deployment-Canary.ipynb)

# Auftrag für Praktische Arbeit



- ★ Problemstellung
  - Die Entwickler erwarten von der IT den gleichen Komfort wie in der Cloud.
  - Die Entwickler haben Ihre Applikationen als Container Images verpackt und erwarten, dass das Operating diese in einem Kubernetes Cluster betreibt. Auch sollen Updates (CD) möglich sein.
  - Zusätzlich hätten Sie gerne eine Umgebung, wo Sie die Container Images bauen (CI) können.
- ★ Aufgabe
  - Setzt einen Kubernetes Cluster mit einer Applikation, bestehend aus mehreren Microservices, auf.
  - Zusätzlich: erstellt für die Entwickler eine Build Umgebung, wo Sie die Container Images bauen können.
  - Präsentiert, am Schluss, die Lösung (20') und die erworbenen Kompetenzen.
- ★ Zeit
  - 9 Lektionen

# Präsentation der Lösungen



- ★ Team Vorstellung
- ★ Eure Lösung
  - Idee
  - Umsetzung
  - Probleme
- ★ Erworbene Kompetenzen.
- ★ Fazit