HAW HAMBURG

Tutorial:

Getting Started with the ICC

Florian Stäps & Ilona Blanck
ICC - Team

Inhaltsverzeichnis

1 Vorbedingungen (Installation auf dem lokalen Rechner)

- Ein 64-bit Betriebssystem
- Git installiert: Link
- Docker installiert
  - Linux
    * Debian
    * Fedora
    * Ubuntu
    * Andere
  - MacOS
  - Windows
- kubectl installiert: Link
- kubelogin installiert: Link
- Einmalig bei GitLab der HAW angemeldet: Link

1.1 Anmerkungen / Hinweise

Falls Sie Verbesserungsvorschläge oder Anmerkungen bezüglich dieses Tutorials haben, schicken Sie bitte eine Mail an: icc@informatik.haw-hamburg.de oder schreiben Sie uns über Slack im Channel icc an.

- **Nutzung im PC - Pool:** Die Benutzung von kubelogin, kubectl und Docker ist zurzeit unter Windows in den PC - Pools nicht möglich. Unter Linux kann kubelogin und kubectl genutzt werden, allerdings steht dort zurzeit kein Docker zur Verfügung.
- Falls Docker wieder beschlossen hat, vor die Downloads für MacOS und Windows einen Login zu setzen, sind im nachfolgenden jeweils ein Direktlink zu den Downloads. Die Links sind von der folgenden Seite: `https://github.com/docker/docker.github.io/issues/6910`, deshalb keine Garantie auf Gültigkeit der Links.
  - MacOS
  - Windows
- Die im folgenden Text gezeigten Beispieldateien, können alle aus dem haw-world Gitlab Projekt des AI Labors heruntergeladen werden. Sie ersparen sich dadurch Fehler beim Abtippen oder durch unsichtbare Zeichen beim Copy & Paste.
  Das Repository finden Sie hier: `https://git.haw-hamburg.de/icc/haw-world`
- **Nur Windows:** Verwenden Sie NICHT die mingw shell, die etwa bei einigen Git Installationen dabei ist. Hier werden keine interaktiven Sessions, etwa für Passworteingabe unterstützt. Nutzen Sie stattdessen das übliche CMD, die PowerShell oder Bash for Windows.
- **Nur macOS:** Je nach verwendeten macOS Version, ist es erforderlich die kubelogin Datei über Rechtsklick -> Öffnen zu starten, bevor diese über das Terminal verwendet werden kann.

- **Windows & macOS:** Im Nachfolgenden sind die Docker Befehle mit sudo angegeben, dies ist nur für die Linux Anwender relevant. Unter Windows und macOS ist sudo nicht nötig.
- **Nur Linux und macOS**: Benennen Sie die kubelogin Dateien, nachdem Sie diese runtergeladen haben, in kubelogin um.
- Für Docker muss Ihre CPU die vT Erweiterung unterstützen. Alle modernen CPUs haben diese, bei einigen Laptops lässt sich die Erweiterung im Bios aktivieren. Einige alte Geräte mit älteren Prozessoren haben leider keine vT Erweiterung. Hier müssen Sie auf die lokale Ausführung von Docker verzichten.
- Nutzen Sie bei der Namensgebung von Ressourcen innerhalb von Kubernetes z. B. Servicenamen oder Secrets, immer nur Zahlen, Kleinbuchstaben und Bindestriche. Ansonsten wird es zu Problemen führen.
- Wenn Sie mit dem Tutorial durch sind löschen Sie bitte das Deployment wieder mit:

  ```
  kubectl delete deployment haw-world -n <KubernetesProjectNamespace>
  ```

  oder

  ```
  kubectl delete -f deploy.yaml -n <KubernetesProjectNamespace>
  ```

2 Erstellung und Auslieferung einer Beispiel-Webapplikation

2.1 Anlegen des Projekts

1. Melden Sie sich bei Gitlab mit Hilfe des nachfoglenden Links an: Link
2. Klicken Sie im 'Projects'-Fenster auf den grünen *New Project - Button* und erstellen Sie ein neues, privates Projekt mit dem Namen „haw-world".

   ☞ **Hinweis 1**
   Nutzen Sie in Ihrem Gruppen- und Projektnamen nur kleine Buchstaben, Bindestriche und Zahlen, da es sonst später zu Problemen führt.

3. Öffnen Sie Ihr Projekt in GitLab, drücken Sie auf den Button *Clone* und kopieren Sie den Link bei SSH.
4. Öffnen Sie nun ein Terminal/Shell auf Ihrem Computer und navigieren Sie in ein Verzeichnis, in dem Sie Ihre Projekte verwalten.
5. Geben Sie in Ihrem Terminal den folgenden Befehl ein, um Ihr Git - Repository lokal zur Verfügung zu haben:

   ```
   # Die GitLab Repository URL ist ohne die <> Zeichen anzugeben
   git clone <Ihre GitLab Repository URL, aus dem vorherigen Schritt>
   ```

   ☞ **Hinweis 2**
   Sollten bei diesem Schritt Probleme auftreten, öffnen Sie Ihr neu erstelltes Repository in GitLab und führen die *Command line instructions* durch, die für Ihr Szenario notwendig sind. Sie finden diese weiter unten, auf der Details Seite Ihres Projekts.

2.2 Erstellen der Beispielapplikation und der Docker Images

1. Starten Sie Docker auf Ihrem Rechner.
2. Im Folgenden erstellen Sie die Webapplikation, die nach dem Start einen Text ausgibt. Als Arbeitsverzeichnis dient das Verzeichnis, aus dem vorherigen Schritt 2.1.
   Erstellen Sie eine Datei *index.php* mit folgendem Inhalt:

   ```php
   <?php
   echo "Hello HAW-World!";
   // Zugriffsdatum protokollieren
   file_put_contents("data/haw-world.log", strftime("%d.%m.%Y-%H:%M") . " ",
   FILE_APPEND);
   ?>
   ```

3. Damit Docker weiß, wie das Image aussehen soll, erstellen Sie in Ihrem Arbeitsverzeichnis eine Datei namens *Dockerfile* mit folgendem Inhalt:

   ```
   # inherit from the php:7.1-apache image from docker hub
   FROM php:7.1-apache
   # create data Directory and set permissions
   RUN mkdir -p /var/www/html/data
   # copies your php files
   COPY . /var/www/html/
   RUN chown -R www-data:www-data /var/www/html/*
   ```

4. Fügen Sie alle neue Dateien Ihrem Git Repository hinzu, commiten und pushen Sie diese:

   ```
   git add .
   git commit -m “Added basic files”
   git push -u origin
   ```

5. Loggen Sie sich bei der Informatik-Docker-Registry mit Ihrem HAW Account ein:

   ```
   sudo docker login git.haw-hamburg.de:5005
   ```

6. Erzeugen Sie nun das Image *haw-world*, durch Ausführen des nachfolgenden Befehls (Punkt am Ende für das aktuelle Verzeichnis nicht vergessen):

   ```
   sudo docker build -t haw-world .
   ```

7. Lassen Sie sich das erzeugte Image mit dem nachfolgenden Befehl anzeigen, die Ausgabe sollte ähnlich zu der in Abbildung 1 sein.

```
sudo docker images
```

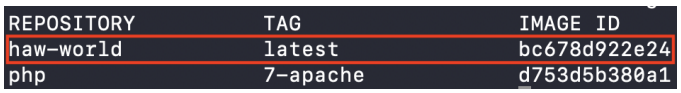

Abbildung 1: Beispielausgabe des Befehls

8. Damit das Docker-image in die Docker-Registry hochgeladen werden kann, muss ein spezieller Alias vergeben werden, dieser wird von GitLab vorgegeben.

   Sie finden diesen in dem Sie Ihr Projekt in GitLab öffnen und dort den Punkt *Packages -> Container Registry* aufrufen. In der Abbildung 2 sehen Sie einen Ausschnitt aus dem Menüpunkt. Die für Sie relevante Information wurde in der Abbildung mit einem roten Kasten markiert. Sie können die Information im Kasten durch einen Druck auf den danebenstehenden Button kopieren.

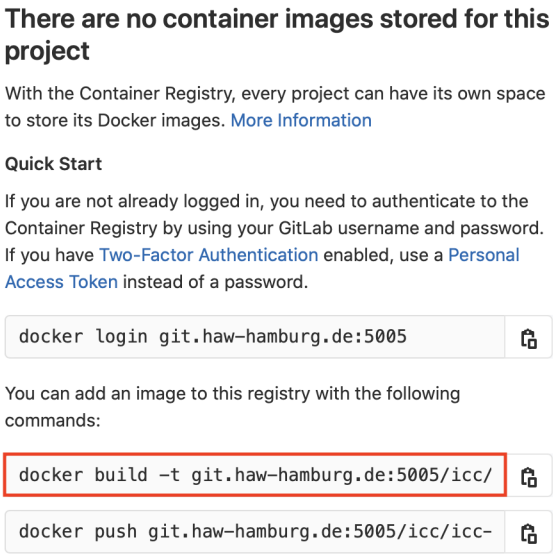

Abbildung 2: Ausschnitt aus dem GitLab Registry Menüpunkt

9. Um dem Image einen Alias zu geben, muss man es taggen.

```
sudo docker tag haw-world <Pfad aus Container Registry>
```

10. Lassen Sie sich erneut die lokal vorhandenen Images anzeigen, die Ausgabe des Befehls sollte ähnlich zu der in Abbildung 3 sein.

```
sudo docker images
```

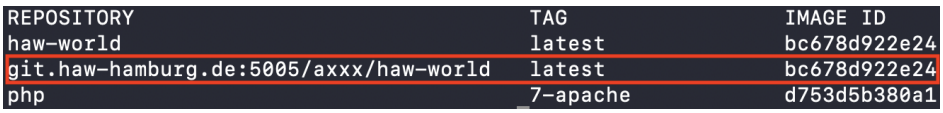

Abbildung 3: Beispielausgabe des Befehls

Kontrollieren Sie, dass die IMAGE ID bei beiden erzeugten Images identisch ist, da es sich bei dem neu benannten Image, lediglich um einen Alias handelt.

11. Laden Sie Ihr Image nun mit dem push Befehl in die Informatik-Docker-Registry:

```
sudo docker push <Pfad aus Container Registry>
```

2.3 Deployment (Auslieferung) der Applikation in die ICC

1. Erstellen Sie in GitLab ein Deploy - Token, dieses benötigt Kubernetes, damit auf private Repositories zugegriffen werden kann.
   (a) Öffnen Sie Ihr Projekt in GitLab
   (b) Öffnen Sie über die Navigationsleiste den Punkt Settings -> Repository und öffnen über den Button *Expand* den Punkt *Deploy Token*. Sie sollten anschließend, wie in Abbildung 4 zusehen ist, den folgenden Menüpunkt vor sich haben.

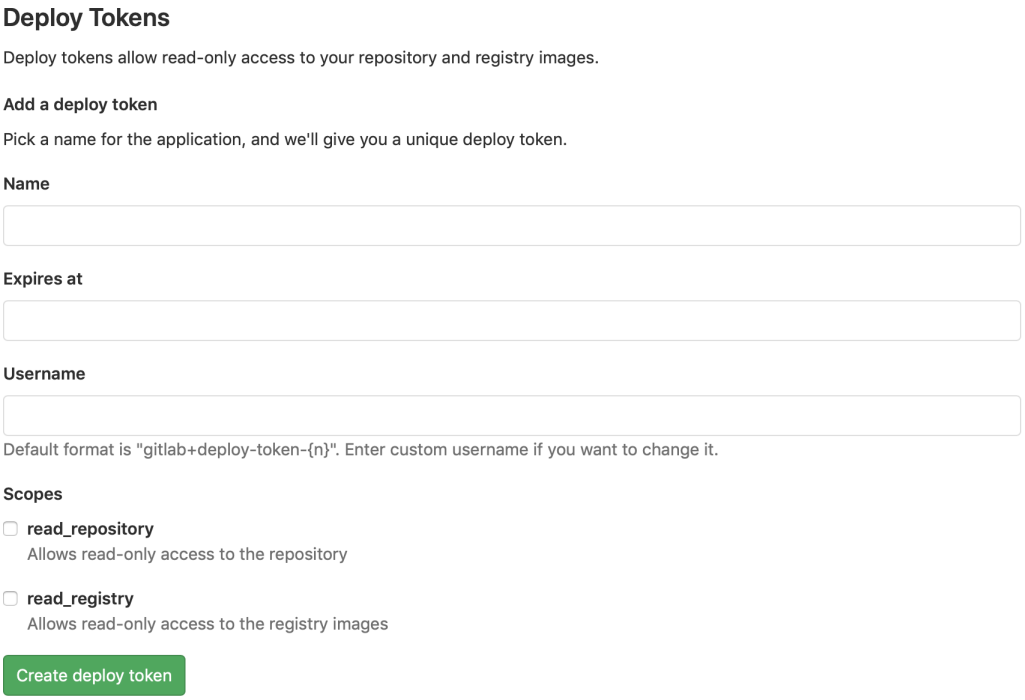

Abbildung 4: Ausschnitt des Menüpunkts Deploy Token

(c) Geben Sie im Feld *Name* einen Namen für das Token ein z.B. Haw World Tutorial

(d) Sie können auch ein Verfalldatum angeben, in dem Sie in das Feld *Expires at* klicken und dann anschließend über den Kalender ein Datum auswählen.

(e) Es ist auch möglich, einen eigenen Benutzernamen anzugeben, wenn Sie mit dem von GitLab generierten unzufrieden sind. Ansonsten lassen Sie das Feld leer.

(f) Setzten Sie den Hacken beim Feld *read_registry*. Danach sollte dies ungefähr so bei Ihnen aussehen, wie in Abbildung 5 zu sehen ist.

(g) Klicken Sie anschließend auf den Button *Create deploy token*.

(h) GitLab zeigt Ihnen anschließend ein neues Fenster an, indem Sie einen generierten Benutzernamen und ein generiertes Passwort sehen.

(i) Kopieren Sie sich die beiden Informationen, da GitLab Ihnen das Passwort nicht noch einmal anzeigen wird und Sie die beiden Informationen im nächsten Schritt benötigen.

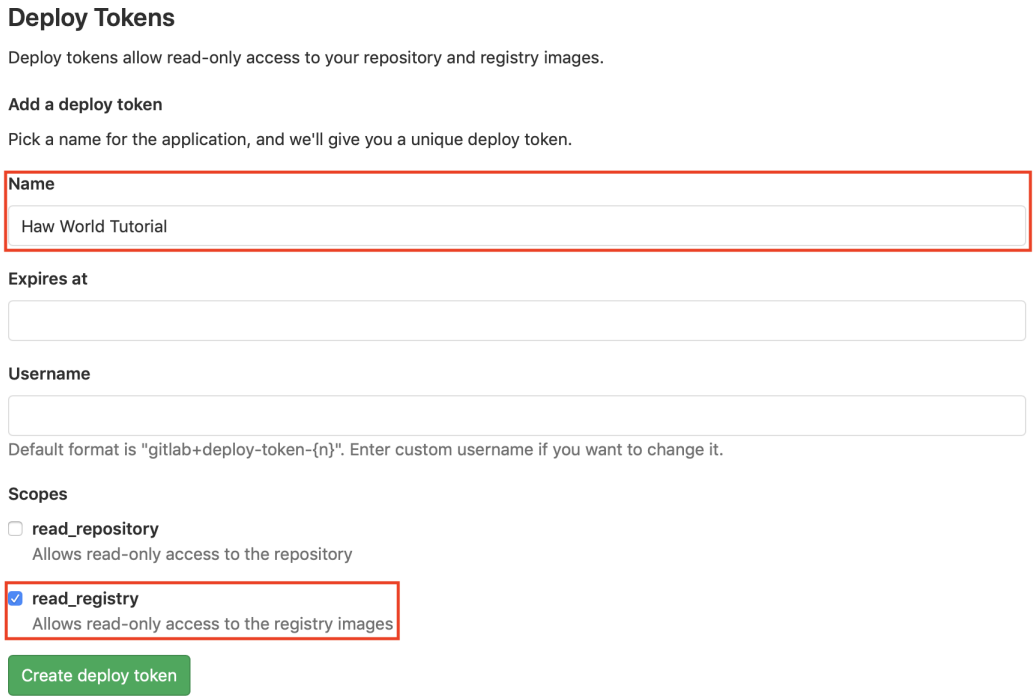

Abbildung 5: Ausschnitt des Menüpunkts Deploy Token ausgefüllt

2. Führen Sie mithilfe des nachfolgenden Befehls kubelogin aus, um sich gegen das ICC-Cluster zu authentifizieren. Dadurch erhalten Sie die Zugriff Konfiguration auf Ihrem Computer, die kubectl benötigt.

   ```
   ./kubelogin
   ```

   Sie werden nach ihrem Benutzername und Passwort gefragt, hier nutzen Sie bitte Ihre a - Kennung.

3. Damit Kubernetes auf Ihr Repository zugreifen kann, müssen die Tokens aus dem vorherigen Schritt als Secret angelegt werden. Führen Sie dazu den folgenden Befehl aus und ergänzen Sie die fehlenden Informationen.

   (Die \ Zeichen am Ende der Befehle sind nur notwendig, wenn Sie den Befehl aus der PDF - Datei kopieren. Damit erkennt das Terminal den Befehl als einen über mehrere Zeilen an.)

   ```
   kubectl create secret docker-registry <SecretName> -n <ProjectNamespace> \
   --docker-server=git.haw-hamburg.de:5005 \
   --docker-username=<token-benutzername> \
   --docker-password=<token-passwort> \
   --docker-email=<your-email>
   ```

   - Bei <*SecretName*> tragen Sie einen Namen für das Secret ein, dieses wird später noch benötigt. Nutzen Sie hier nur Zahlen, Kleinbuchstaben und Bindestriche.
   - Bei <*ProjectNamespace*> tragen Sie den Kubernetes Namespace ein (Mehr dazu siehe Hinweis 3 / Abbildung 6)

- Bei *docker-username* geben Sie den Usernamen Token aus Schritt 1 an.
- Bei *dockerpassword* geben Sie das Password Token aus Schritt 1 an.
- Bei *docker-email* tragen Sie Ihre HAW - Mail Adresse ein

☞ **Hinweis 3**
Den Kubernetes Namespace finden Sie in GitLab. Öffnen Sie dazu Ihr Projekt und gehen Sie dort auf *Operations* -> *Kubernetes* und wählen anschließend den Punkt *haw-icc* aus. Dort scrollen Sie dann runter, bis Sie den Punkt *Kubernetes cluster details* finden und klicken dort auf den Expand Button. Dies sollte wie in Abbildung 6 aussehen.

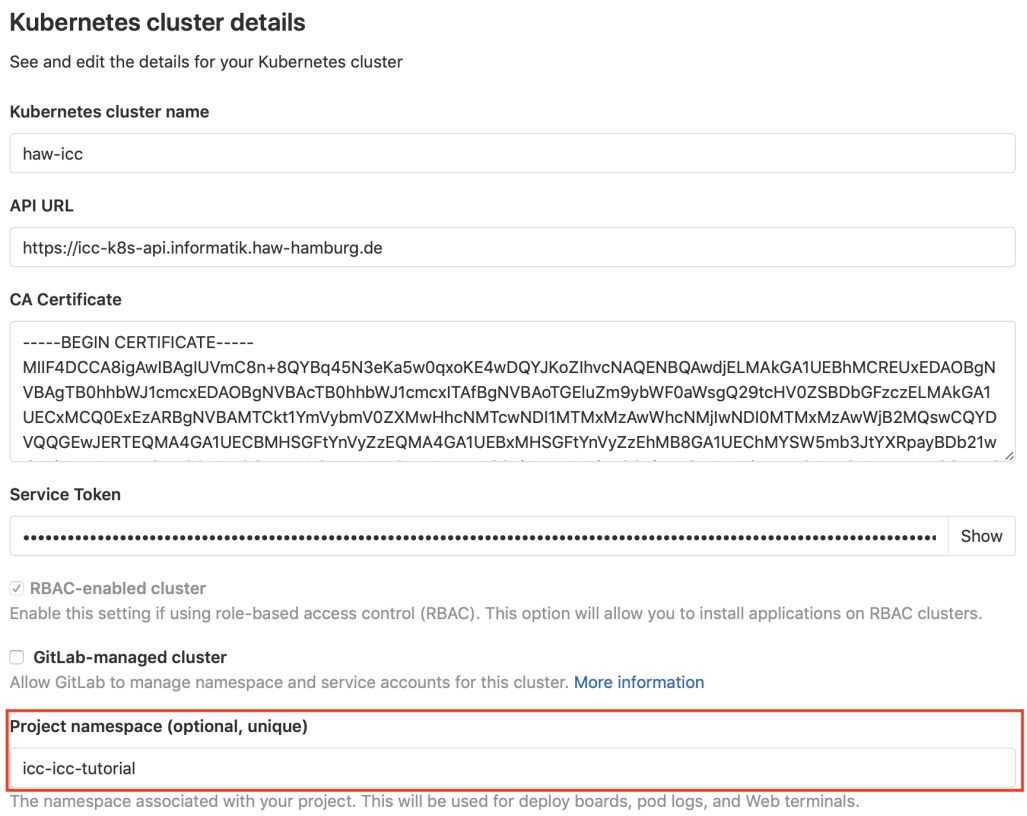

Abbildung 6: Ausschnitt des Kubernetes Integrations Punkt

4. Erstellen Sie nun eine Beschreibungsdatei mit dem Namen *deploy.yaml* für das Deployment des Containers in der ICC mit folgendem Inhalt (Achtung: Einrückungen haben semantische Bedeutung und dürfen nicht verändert werden. Es wird daher empfohlen, die Beispieldatei aus dem Beispielprojekt zu verwenden und anzupassen, da es zu Problemen führt, wenn die Zeilen aus der pdf kopiert werden. Die URL zum Projekt finden Sie hier 1.1).

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    service: haw-world
  name: haw-world
spec:
  selector:
    matchLabels:
      service: haw-world
  template:
    metadata:
      labels:
        service: haw-world
    spec:
      containers:
      - image: git.haw-hamburg.de:5005/<GitLab-Gruppe(=a-Kennung)>/haw-world
        name: haw-world
        ports:
        - containerPort: 80
          protocol: TCP
      imagePullSecrets:
      - name: <SecretName>

---

apiVersion: v1
kind: Service
metadata:
  labels:
    service: haw-world
  name: haw-world
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    service: haw-world
  type: ClusterIP
```

   - Ersetzen Sie <*GitLab-Gruppe(=a-Kennung)*> durch Ihre a - Kennung
   - Ersetzen Sie <*SecretName*> durch den Namen, den Sie in Schritt 3 festgelegt haben

5. Fügen Sie die YAML-Datei Ihrem Git-Repository hinzu

```
git add deploy.yaml
git commit -n -m "Added YAML - Deploy file"
git push -u origin
```

6. Schreiben Sie nun diese Deploymentbeschreibung via kubectl in das Cluster. Führen Sie dazu folgenden Befehl aus:

```
kubectl apply -f deploy.yaml -n <KubernetesProjectNamespace>
```

   Das Cluster startet nun Ihre Container in den Pods und erzeugt dazu Services, so dass diese Container zugänglich sind.

7. Sehen Sie sich das Ergebnis mit folgendem Befehl an

```
kubectl get deploy,pod,svc -n <KubernetesProjectNamespace>
```

   Die Ausgabe des Befehls sollte ähnlich wie in Abbildung 7 aussehen.

```
NAME                              DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/haw-world   1         2         1            1           23d

NAME                              READY   STATUS              RESTARTS   AGE
pod/haw-world-7b8bd84bf4-95xq8    0/1     ContainerCreating   0          46m
pod/haw-world-7cb68b9c58-qz6zd    1/1     Running             0          21d

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/haw-world    ClusterIP   10.32.34.216   <none>        80/TCP    28d
```

Abbildung 7: Ergebnis des Befehls

   Die Ausgabe zeigt Ihnen ihren pod Namen an, alles ab dem Punkt *pod/*, ob der Pod Ready ist, den Status, hier sollte Running stehen, die Anzahl der Neustarts (Restart.), hier sollte am Anfang 0 stehen und das Alter (Age).

8. Führen Sie nun den kubectl proxy aus, damit Sie auf Ihre Webseite von Ihrem Rechner aus zugreifen können:

```
kubectl proxy
```

   Sie können nun Ihren Service über die folgende URL aufrufen:
   `localhost:8001/api/v1/namespaces/<ProjektNamespace>/services/<ServiceName>:http/proxy/`

   Ergänzen Sie die Felder <Projekt Namespace> und <ServiceName> um Ihre entsprechenden Daten.

> ☞ **Hinweis 4**
> Den ServiceName finden Sie in Ihrer *deploy.yaml* - Datei, im oberen Teil unter dem Punkt *service*.

9. Um eine interaktive Shell in Ihrem Container zu erhalten, geben Sie bitte ein:

```
kubectl exec -it <Name des Pods> -n <KubernetesProjectNamespace> --bash
```

> ☞ **Hinweis 5**
> Beachten Sie, dass alle Änderungen beim Stoppen des Containers verloren gehen, falls Sie keinen persistenten Speicher eingerichtet haben. Mehr zur Einrichtung eines peristenten Sepichers finden Sie im Abschnitt 4.

10. Löschen Sie nun Ihr Deployment und damit den haw-world Pod, in dem Sie folgenden Befehl eingeben:

```
kubectl delete deployment haw-world -n <KubernetesProjectNamespace>
```

3 Automatisierung des Image Builds als Pipeline

3.1 Anmerkungen

- Sie finden die Beispiel YAML - Datei unter folgendem Link: Link zur YAML Datei

  Diese können Sie für Ihre Anwendung bereits so nutzen. Achten Sie aber darauf, dass in dieser Datei davon ausgegangen wird, das Ihre Deploy - YAML Datei, *deploy.yaml* heißt.

  Falls Sie Ihre Datei nicht umbenennen wollen, müssen Sie in der Beispieldatei die vorletzte Zeile anpassen.

Das von Ihnen erstellte Image beinhaltet eine konkrete Version Ihres Programmcodes. Wenn Sie Ihren Code ändern, müssen Sie also auch ein neues Image erzeugen. Da darauf keiner Lust hat, automatisieren wir in den folgenden Schritten diesen Prozess.

1. Erstellen Sie eine Datei mit Namen *.gitlab-ci.yml* (beachten Sie den Punkt am Anfang des Namens, die Datei ist versteckt und wird standardmäßig vom Betriebssystem nicht angezeigt.) im Wurzelverzeichnis Ihrer Anwendung (also auf einer Ebene mit index.php und den anderen YAML Dateien). Gitlab liest diese Datei automatisch aus und erstellt daraus eine Buildpipeline.

2. Die *.gitlab-ci.yml* wird für unseren Anwendungsfall einen Job in einer Stage definieren. Dort bauen wir das Docker Image, taggen und pushen es schließlich in die Nexus Container Registry der HAW Hamburg.

   Kopieren Sie dazu den folgenden Inhalt in die zuvor erstelle Datei und achten Sie darauf, dass die Formatierung eingehalten wird. Es wird empfohlen die Beispieldatei aus den Anmerkungen zu verwenden siehe 3.1.

```
# .gitlab-ci.yaml
stages:
    - dockerize
    - deploy

variables:
 DOCKER_REGISTRY: "git.haw-hamburg.de:5005"
 SERVICE_NAME: "haw-world"
 DOCKER_TLS_CERTDIR: "/certs"

create_image:
 stage: dockerize
 image: docker:19.03.3
 services:
   - docker:19.03.3-dind
 tags:
   - dind
   - docker
 script:
   - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $DOCKER_REGISTRY
   - docker build -t $SERVICE_NAME:latest .
   - docker tag $SERVICE_NAME:latest
   $DOCKER_REGISTRY/$CI_PROJECT_NAMESPACE/$SERVICE_NAME:latest
   - docker push $DOCKER_REGISTRY/$CI_PROJECT_NAMESPACE/$SERVICE_NAME:latest

deploy image:
  environment:
    name: ICC-K8s
  stage: deploy
  image: $DOCKER_REGISTRY/icc/kubectl:v1.14.0
  script:
  - kubectl apply -f deploy.yaml
```

   In der Jobdefinition (alles ab *createImage*) wird ebenfalls ein Image definiert. Dies liegt daran, dass auch der eigentliche Build Vorgang in einem Container in der ICC ausgeführt wird.

   Als Basis Image verwenden wir hier das docker Image, da wir in unserem Build Container Docker Befehle ausführen können wollen.

   Zusätzlich spezifizieren wir einen Service (*services*). Dadurch wird ein zusätzlicher Container gestartet, der unserem Build Container via localhost zur Verfügung steht und eine Docker Engine anbietet.

   Die tags dienen dafür, dass der priviligierte Runner für diesen Schritt genutzt wird.

   Im „script“ Teil schließlich führen wir die bekannten Docker Befehle aus.

3. Als nächstes aktivieren Sie den Shared Runner für Ihr Projekt. Öffnen Sie dafür Ihr GitLab Projekt und gehen dort auf Settings -> CI/CD -> Runner Settings und aktivieren Sie den Shared Runner. (Enable Shared Runner)

4. Commiten und pushen Sie die .gitlab-ci.yml nun via Git in Ihr Gitlab Repository.

```
git add .gitlab-ci.yml
git commit -m "Added GitLab CI YAML - File"
git push
```

5. Navigieren Sie im Browser in Ihr Gitlab Projekt und dann auf CI / CD -> Pipelines.

   Sie sollten hier nun eine Darstellung Ihrer Build Pipeline sehen. Mit einem Klick auf das Symbol in der Spalte Stages gelangen Sie in die Details Ihres Build Jobs.

   Etwaige Fehler in der Konfiguration werden Sie hier finden. Läuft der Build wie geplant durch, wird die Pipeline grün und Sie können Ihr Image mit Docker unter Angabe der Pipeline ID als Tag abrufen. Geben Sie dazu den folgenden Befehl in ein Terminal ein:

```
sudo docker pull <Pfad aus Container Registry>:latest
```

4 Einrichtung eines persistenten Speichers

Das AI-Labor bietet begrenzten persistenten Speicher, als Block Storage über einen CEPH Cluster an. Dieser Speicherplatz kann in der ICC in Form von Volumes mit festgelegter Größe genutzt werden. Um ein Persistent Volume anzufordern gehen Sie wie folgt vor:

1. Erzeugen Sie in Ihrem Projekt eine Datei *my-pvc.yaml* für einen PersistentVolumeClaim mit folgendem Inhalt.

   Es wird empfohlen, die Beispieldatei aus dem Beispielprojekt zu verwenden und anzupassen, da es zu Problemen führt, wenn die Zeilen aus der pdf kopiert werden. Die URL zum Projekt finden Sie hier 1.1

   ```
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: my-storage-claim
     labels:
       service: haw-world
   spec:
     accessModes:
     - ReadWriteOnce
     resources:
       requests:
         storage: 10Mi
   ```

   ☞ **Hinweis 6**
   Die Storage Größe können Sie variieren, in dem Sie den Wert in der letzten Zeile verändern. Bedenken Sie jedoch, dass Speicherplatz nur begrenzt zur Verfügung steht und Volumes nicht vergrößert- oder verkleinerbar sind. Der Zugriffsmodus steht auf 'ReadWriteOnce', was bedeutet, dass Lesend wie Schreibend nur von einem Pod auf dieses Volume zugreifbar ist. Alternativen finden Sie in der Kubernetes Dokumentation.

2. Legen Sie den Claim im Cluster an. Geben Sie dazu den folgenden Befehl in einem Terminal an. (Sollten Sie wegen des Kubernetes Projekt Namespace nicht sicher sein, finden Sie unter dem Punkt Hinweis 3 eine Anleitung dazu.)

   ```
   kubectl apply -f my-pvc.yaml -n <KubernetesProjectNamespace>
   ```

3. Erweitern Sie das Deployment in Ihrer *deploy.yaml* um die Referenzierung des VolumeClaims und geben Sie den Mountpfad an. Es wird empfohlen, die Beispieldatei (deploy_pvc.yaml) aus dem Beispielprojekt zu verwenden und anzupassen, da es zu Problemen führt, wenn die Zeilen aus der pdf kopiert werden. Die URL zum Projekt finden Sie hier 1.1

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    service: haw-world
  name: haw-world
spec:
  selector:
    matchLabels:
      service: haw-world
  template:
    metadata:
      labels:
        service: haw-world
    spec:
      containers:
      - image: <Pfad aus Container Registry>
        name: haw-world
      imagePullSecrets:
      - name: <SecretName>
        volumeMounts:
        - mountPath: "/var/www/html/data"
          name: my-volume # Hier können Sie einen beliebigen Namen wählen
        ports:
        - containerPort: 80
          protocol: TCP
      volumes:
      - name: my-volume
        persistentVolumeClaim:
         claimName: my-storage-claim # Dies ist der Name aus der YAML - Datei
---
apiVersion: v1
kind: Service
metadata:
  labels:
    service: haw-world
  name: haw-world
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
```

```
    service: haw-world
  type: ClusterIP
```

Ergänzen Sie den Punkt *<Pfad aus Container Registry>* um Ihren Projektpfad. Sollten Sie hier unsicher sein, finden Sie unter dem Punkt 8 eine Anleitung, wie Sie diesen herausfinden.

Beachten Sie, dass der *MountPath* auf das *var/www/html/data* Verzeichnis Ihres Containers gesetzt wird. Damit sind alle Daten innerhalb dieses Verzeichnisses auf dem Volume gespeichert. Ihre Anwendung legt die Zugriffslogs entsprechend in diesem Verzeichnis ab.

4. Schreiben Sie nun diese Deploymentbeschreibung in den Cluster, führen Sie dazu den folgenden Befehl aus.

   ```
   kubectl apply -f deploy.yaml -n <KubernetesProjectNamespace>
   ```

   Sobald der Pod zum Deployment erzeugt wird, wird der PersistentVolumeClaim durch Kubernetes an ein PersistentVolume gebunden, welches gegen den CEPH Cluster provisioniert wird. Je nach angeforderter Größe kann die Provisionierung zwischen wenigen Sekunden (< 1GB) bis hin zu mehreren Minuten in Anspruch nehmen.

   Sollten Fehler auftreten, können Sie die Logs mit Hilfe der nachfolgenden Befehle einsehen.

   Auslesen des Namens vom PVC:

   ```
   kubectl get pvc -n <KubernetesProjectNamespace>
   ```

   Einsehen der Logs:

   ```
   kubectl describe pvc <PVCName> # Name aus dem vorherigen Schritt
   ```

   ☞ **Hinweis 7**
   Das erzeugte PersistentVolume bleibt solange erhalten, wie der PersistentVolumeClaim vorhanden ist. Sie können also Ihre Deployments, Pods und Services löschen, ohne Ihre Daten zu verlieren. Löschen Sie aber den PersistentVolumeClaim, so sind auch die Daten weg.

5. Wenn Sie Ihr PVC nicht mehr benötigen, löschen Sie dieses wieder mit dem folgenden Befehl, damit die Ressourcen wieder freigegeben werden.

   ```
   kubectl delete pvc -f my-pvc.yaml -n <KubernetesProjectNamespace>
   ```

5 Kubernetes Jobs

Kubernetes bietet die Möglichkeit Pods mittels Jobs zu verwalten. Dadurch werden die Ressourcen der ICC nur solange reserviert wie auch die dazugehörigen Pods aktiv laufen. Diese Methode wird daher empfohlen zu nutzen, wenn lediglich bestimmte Aufgaben erfüllt werden sollen und keine permanenten Dienste notwendig sind.

Im folgenden ist ein Beispiel aufgeführt, welches einen Job definiert, der π bis zur 2000 Nachkommastelle berechnet.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

- Bei name können Sie einen Namen frei für den Job vergeben
- Unter dem Punkt spec -> Containers muss das Docker Image angegeben werden, welches ausgeführt werden soll.
- Bei backoffLimit wird festgelegt wie häufig ein Job neugestartet werden soll, falls dieser fehlschlägt

> ☞ **Hinweis 8**
> Aus der Sicht von Kubernetes wird ein Jobs als erfolgreich/successful anerkannt wenn die Anwendung innerhalb des Images erfolgreich terminiert (exit code 0). Wird von der Anwendung etwas anderes zurückgegeben startet Kubernetes den Pod bis zum *backoffLimit* spezifizierten Limit neu.

Die erzeugten Jobs können sich mithilfe des folgenden Befehls angezeigt werden lassen:

```
kubectl get jobs -n <Kubernetes Namespace>
```

Werden weitere Informationen über den Job benötigt ist es möglich mittels des folgenden Befehls weitere Details anzeigen zulassen:

```
kubectl describe <Jobname> -n <Kubernetes Namespace>
```

Das Ergbnis des Job kann sich, wenn das Ergebnis von der Anwendung über das Terminal ausgegeben wird, über den folgenden Befehl angezeigt werden lassen. Dies ist solange möglich wie auch der Job bzw. die dazugehörigen Pods existieren.

```
kubectl get logs <Podname> -n <Kubernetes Namespace>
```

Wird der Job nicht mehr benötigt, kann dieser mittels des folgenden Befehls gelöscht werden:

```
kubectl delete job <Job Name> -n <Kubernetes Namespace>

# Alternativ
# Ist der der Namespace in der yaml Datei enthalten, kann das -n Argument weggelassen
werden
kubectl delete -f <yaml Datei> -n <Kubernetes Namespace>
```

6 Verwendung von GPUs

In der ICC stehen eine limitierte Anzahl an GPUs (Tesla V100) zur Verfügung, deshalb sind Interessenten gebeten sich eigenständig an das ICC Team mittles Slack / Mattermost oder per Mail zu wenden.

Generelle Voraussetzung ist, dass das Container-Image GPU-Unterstützung hat, dazu könnte es zum Beispiel auf den NVIDIA Cuda Images basieren.

Damit ein Container auf eine GPU zugreifen kann, muss zuerst eine toleration gesetzt sein, damit der Container die Erlaubnis hat auf einen der GPU-Knoten zu laufen:

```yaml
tolerations:
- key: "gpu"
  operator: "Equal"
  value: "true"
```

Nun muss nur noch ein resource-limit gesetzt werden, damit dem Container eine GPU zugeordnet wird (wichtig hierbei ist, dass man keine fraktale einer GPU anfragen kann und request == limit):

```yaml
resources:
  limits:
    nvidia.com/gpu: 1
```

> ☞ **Hinweis 9**
> Da die GPUs nur in begrenzter Anzahl zur Verfügung stehen empfehlen wir Kubernetes Jobs zu verwenden, näheres dazu siehe Abschnitt 5.