

SWD Übung

Fabian Weisser

Links:

GitRepo: <https://github.com/99818fhwn/SWDep2>

Docker-Image Registry-Link: <https://hub.docker.com/r/99818fhwn/nodeapp>

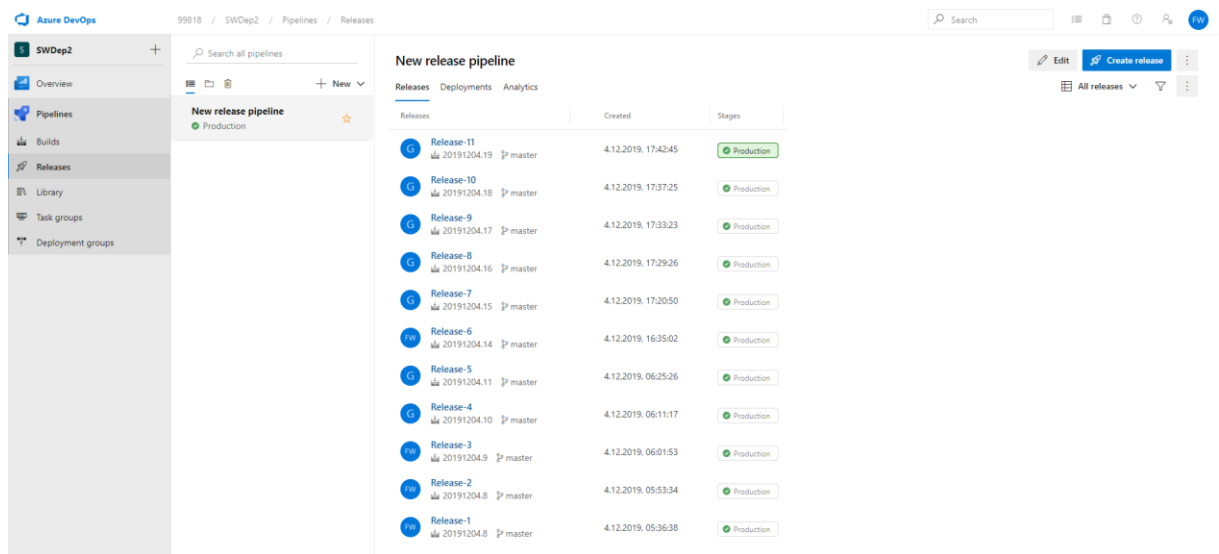
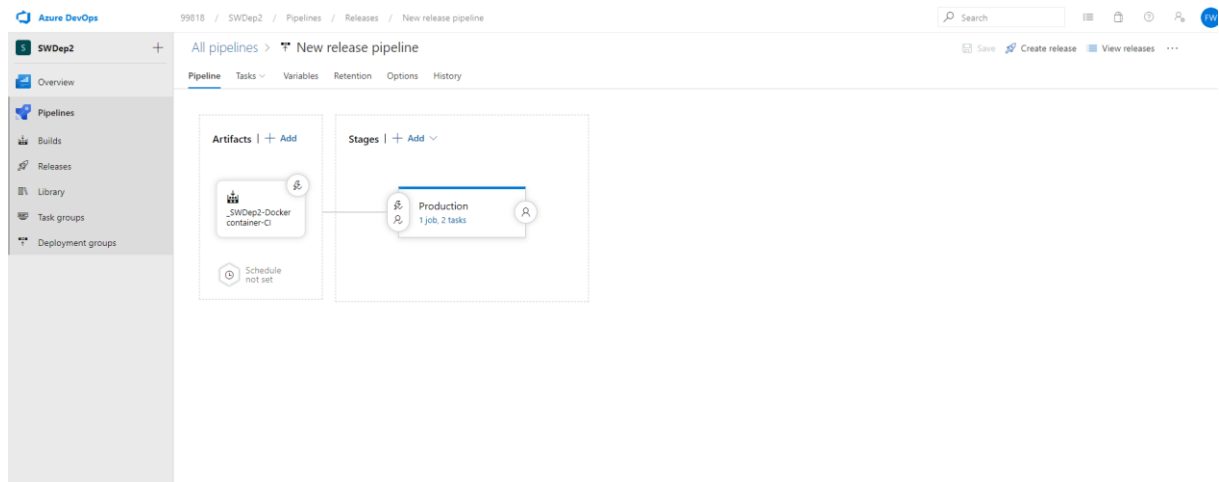
Website: <http://52.137.24.242:3000/>

Screenshots:

The top screenshot shows the Azure DevOps interface for the 'SWDep2-Docker container-CI' pipeline. The left sidebar contains navigation links: Overview, Pipelines, Builds, Releases, Library, Task groups, and Deployment groups. The main area displays the pipeline configuration for 'Agent job 1'. The configuration includes a 'Get sources' task, a 'Build an image' task, a 'Push an image' task, a 'Copy Files to: \$(Build.ArtifactStagingDirectory)' task, and a 'Publish Artifact: drop' task. The 'Agent job 1' configuration is shown on the right, including fields for 'Display name', 'Agent selection', 'Demands', 'Execution plan', 'Parallelism', 'Timeout', 'Job cancel timeout', and 'Dependencies'.

The bottom screenshot shows the 'Builds' page for the 'SWDep2-Docker container-CI' pipeline. The left sidebar is the same as the top screenshot. The main area displays a table of build history entries. The table has columns for 'Commit', 'Build #', 'Branch', 'Queued', and 'Duration'. The build history shows a series of builds, including 'Add heart', 'Change Port', 'Add new Port value', 'Add My phrase', 'Change the pod configuration', 'Removed the V2', and 'Changed something'. The builds are sorted by 'Queued' time, with the most recent build at the top.

Commit	Build #	Branch	Queued	Duration
Add heart CI build for 99818fhwn	20191204.19	master	2019-12-04 - 17:40	1:28.289
Change Port CI build for 99818fhwn	20191204.18	master	2019-12-04 - 17:35	1:36.021
Change Port CI build for 99818fhwn	20191204.17	master	2019-12-04 - 17:31	1:22.927
Change Port CI build for 99818fhwn	20191204.16	master	2019-12-04 - 17:27	1:24.921
Add new Port value CI build for 99818fhwn	20191204.15	master	2019-12-04 - 17:18	1:48.576
Add My phrase Manual build for Fabian Weisser	20191204.14	master	2019-12-04 - 16:33	1:22.999
Add My phrase CI build for 99818fhwn	20191204.13	master	2019-12-04 - 16:25	1:36.960
Change the pod configuration Manual build for Fabian Weisser	20191204.12	master	2019-12-04 - 15:56	1:22.849
Change the pod configuration CI build for 99818fhwn	20191204.11	master	2019-12-04 - 06:23	1:21.878
Removed the V2 CI build for 99818fhwn	20191204.10	master	2019-12-04 - 06:09	1:20.598
Changed something Manual build for Fabian Weisser	20191204.9	master	2019-12-04 - 06:00	1:22.626



Ablauf:

1. Anlegen eines Github Repository
2. Klonen der erstellten Repo -> in den Ordner wechseln
3. Erstellen der Node-Demo-App mittels:

```
npm install express-generator -g
```

4. Installieren der Node-modules: „npm install -g“
5. Hinzufügen der .gitignore Datei und „/node-modules“ eintragen

6. Erstellen des „Dockerfile“ und Inhalt anpassen: (Erstellt ein Nodelmage und übernimmt die Implementation im GitRepo / der Ordner Umgebung):

```
FROM node:latest

# Create project directory (workdir)
RUN mkdir /app
WORKDIR /app

# Add package.json to WORKDIR and install dependencies
COPY package.json .
RUN npm install

# Add the remaining source code files to WORKDIR
COPY . .

# Start the application
CMD ["npm", "start"]
```

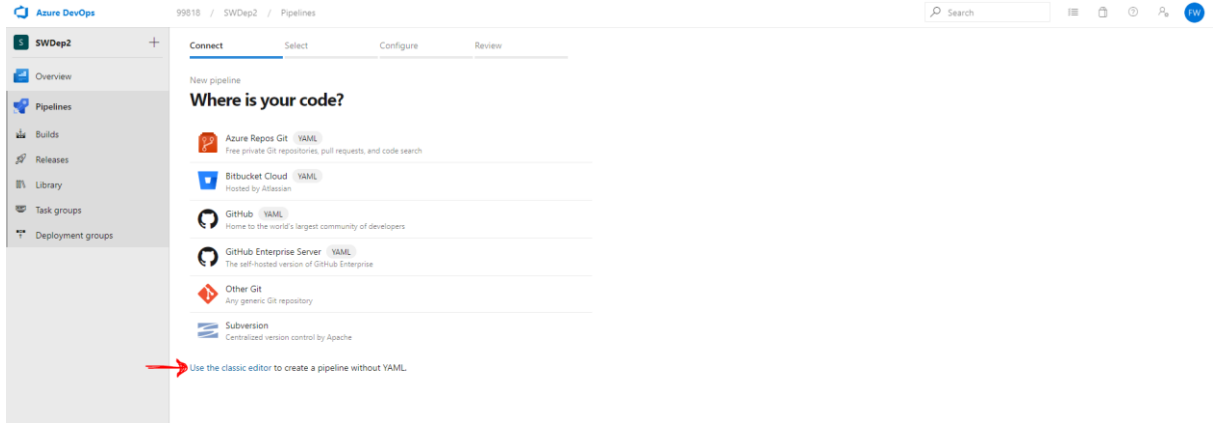
7. „deployment.azure.yaml“ Datei erstellen und Einträge anpassen

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodeapp-deployment
spec:
  selector:
    matchLabels:
      app: nodeapp
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nodeapp
    spec:
      containers:
        - name: nodeapp
          image: 99818fhwn/nodeapp:${Build.BuildId} #imagename in dockerhub
          ports:
            - containerPort: 3000 # port defined by the node-express app
---
# https://kubernetes.io/docs/concepts/services-networking/service/#defining-a-service
kind: Service
apiVersion: v1
metadata:
  name: nodeapp-service
spec:
  selector:
    app: nodeapp
  ports:
    - protocol: TCP
      port: 3000 # port defined by the node-express app
      targetPort: 3000 # port defined by the node-express app
  type: LoadBalancer
```

8. Commit und Push der Änderungen auf das Repo

9. Auf <https://dev.azure.com/> einloggen und neues Projekt erstellen mit dem GitRepo

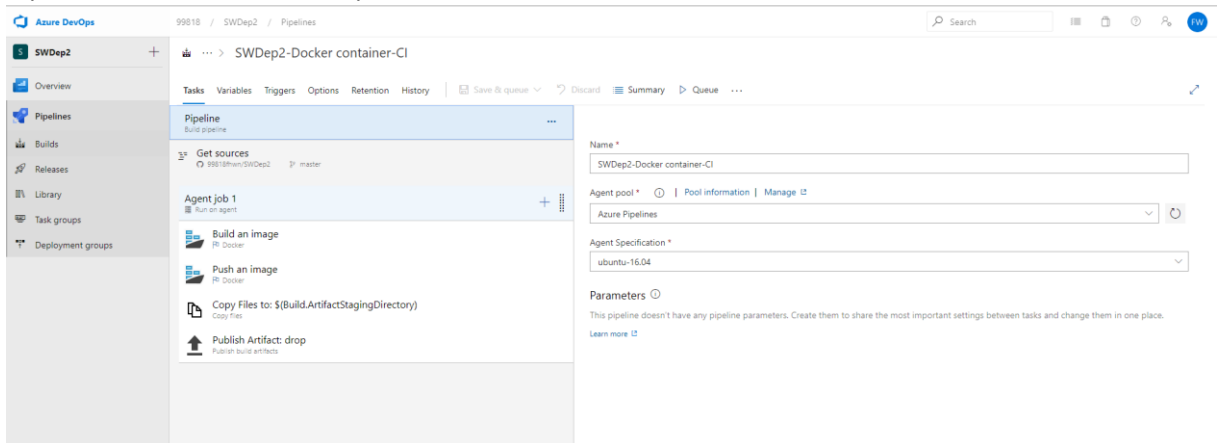
10. Hinzufügen einer BuildPipeline -> New Pipeline -> Use the classic editor



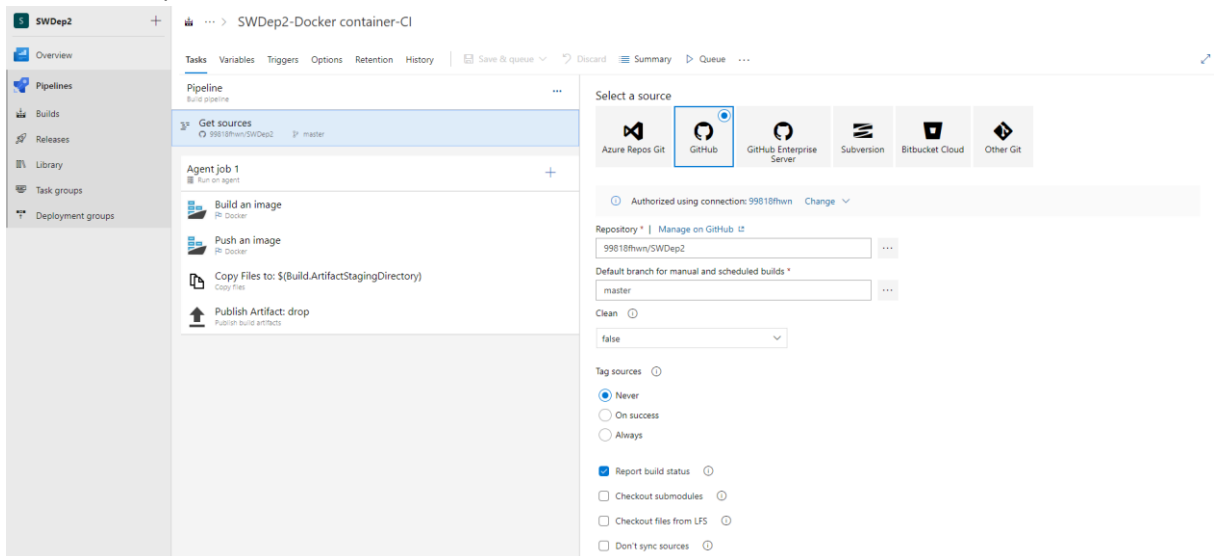
11. GitHub auswählen

12. Nach „Docker Container“ suchen und anwenden

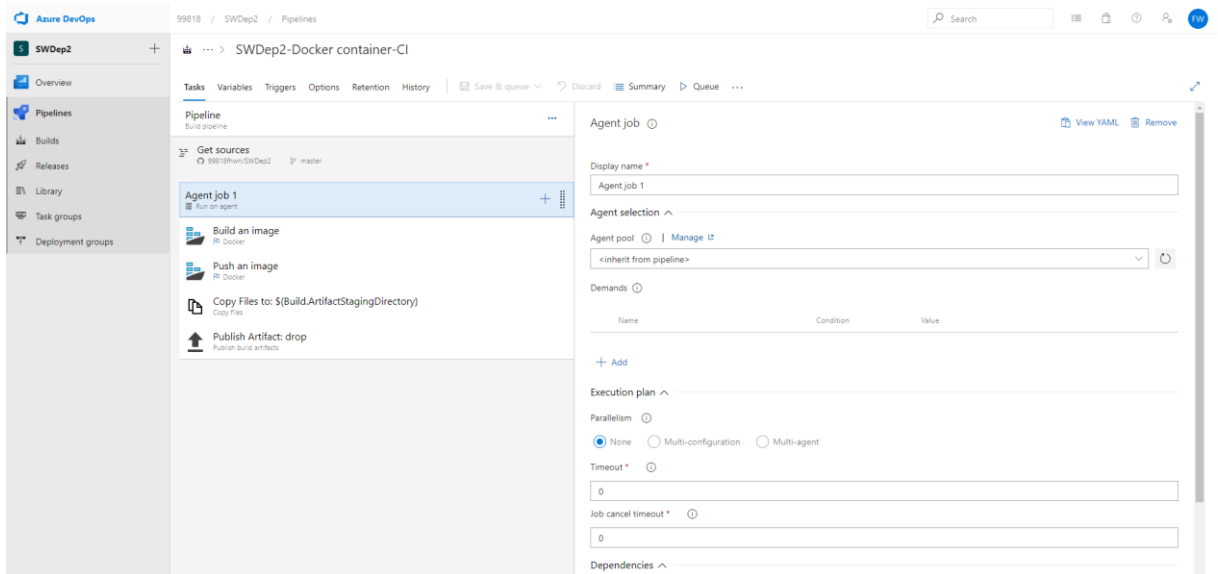
13. Pipeline benennen + Azure Pipelines + ubuntu-16.04



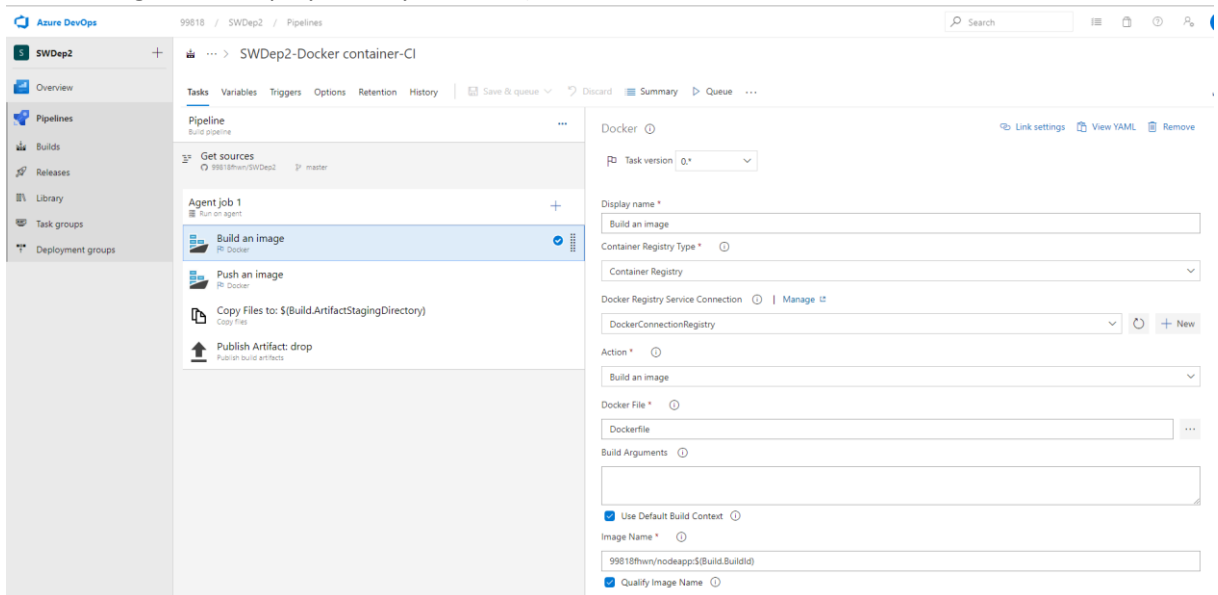
14. Sources überprüfen



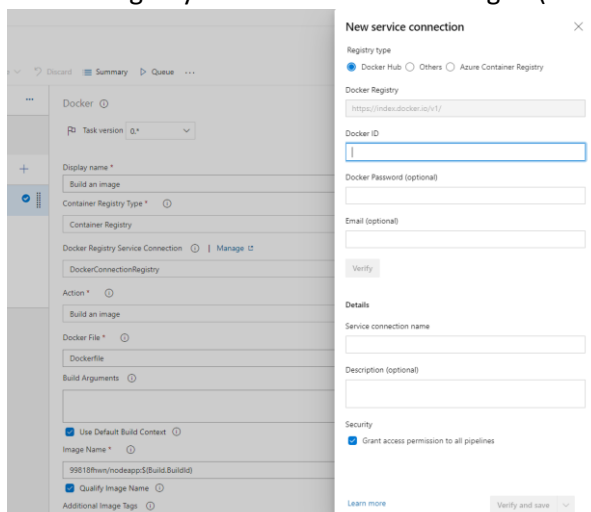
15. Agent Job überprüfen



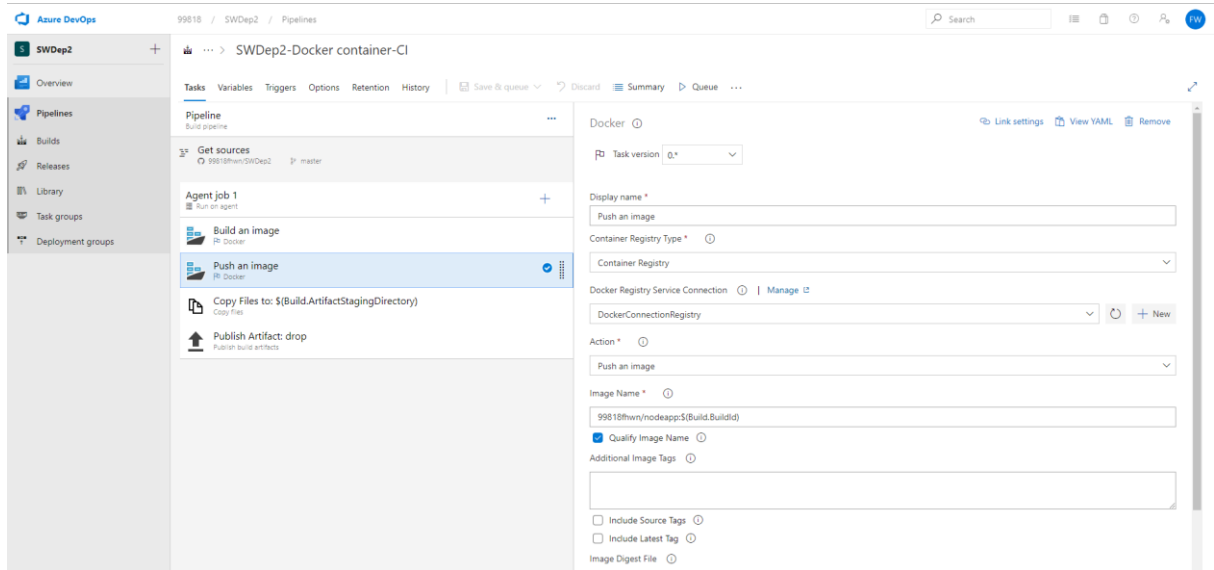
16. „Build an Image“ anpassen -> Conainer Registry Type auf Container Registry setzten -> Dockerfile aus Repo hinterlegen -> Image Name auf die Dockerhub ID anpassen (muss ident mit der Angabe in „deploy.azure.yaml“ sein)



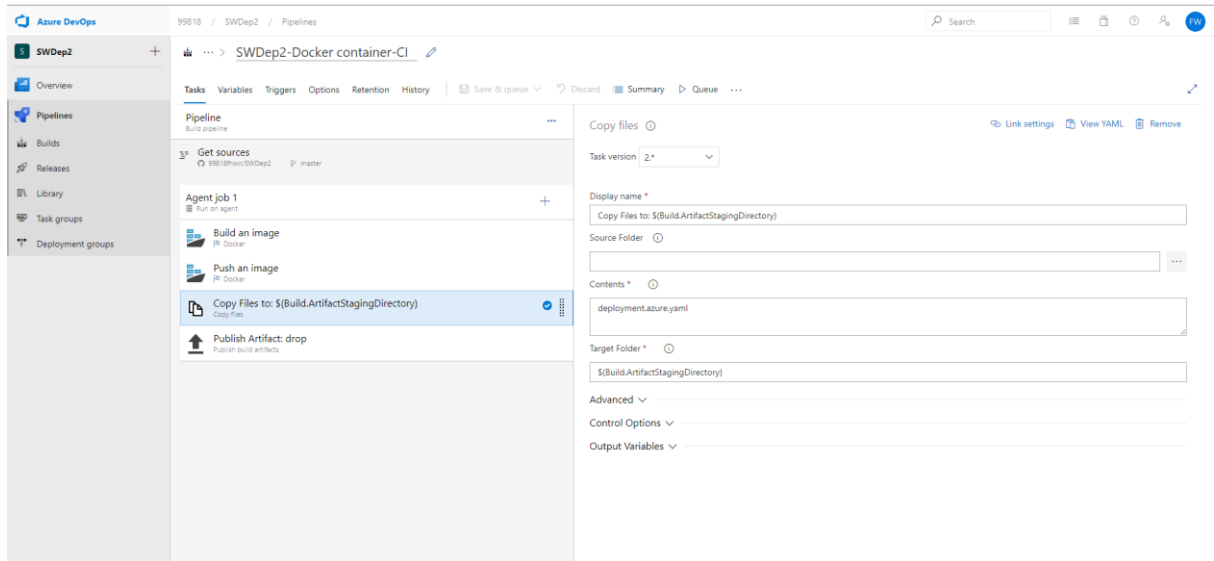
➔ Docker Registry Service Connection anlegen (mit DockerHub + Docker-Id + Passwort)



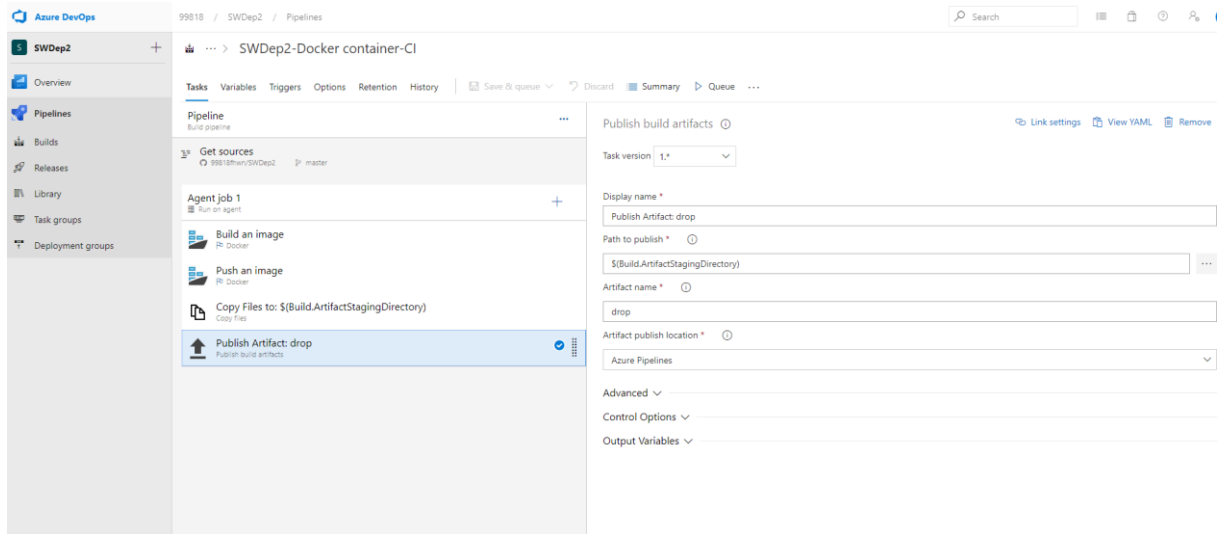
17. „Push an Image“ Anpassen -> Container Registry Type auf Container Registry setzen -> Zuvor verwendete Service Connection wählen -> Image Name auf die Dockerhub ID anpassen (muss ident mit der Angabe in „deployment.azure.yaml“ sein)



18. Hinzufügen der „Copy Files“ Aktion (+ neben Agent job -> suchen) -> Contents die „deployment.azure.yaml“ Datei angeben -> Anpassen Target Folder auf: „\$(Build.ArtifactStagingDirectory)“



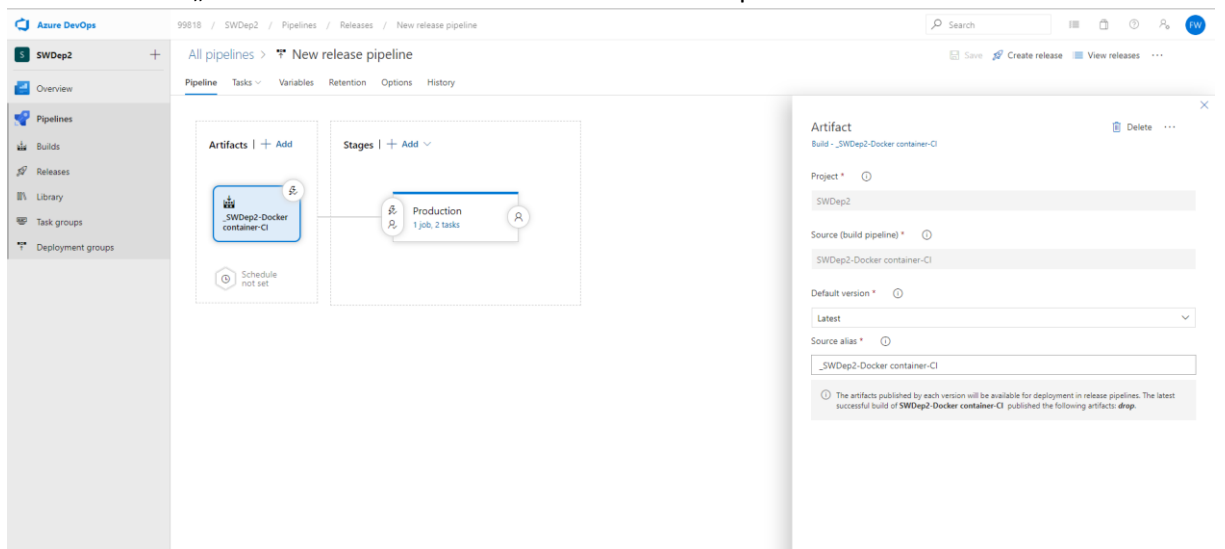
19. Hinzufügen der „Publish Artifact“ Aktion (,+’ neben Agent job -> suchen) -> Artifact „drop“ benennen -> „Path to publish“ auf: „\$(Build.ArtifactStagingDirectory)“ setzen -> Artifact Name „drop“ bezeichnen -> „Publish location“ auf „Azure Pipelines“ stellen



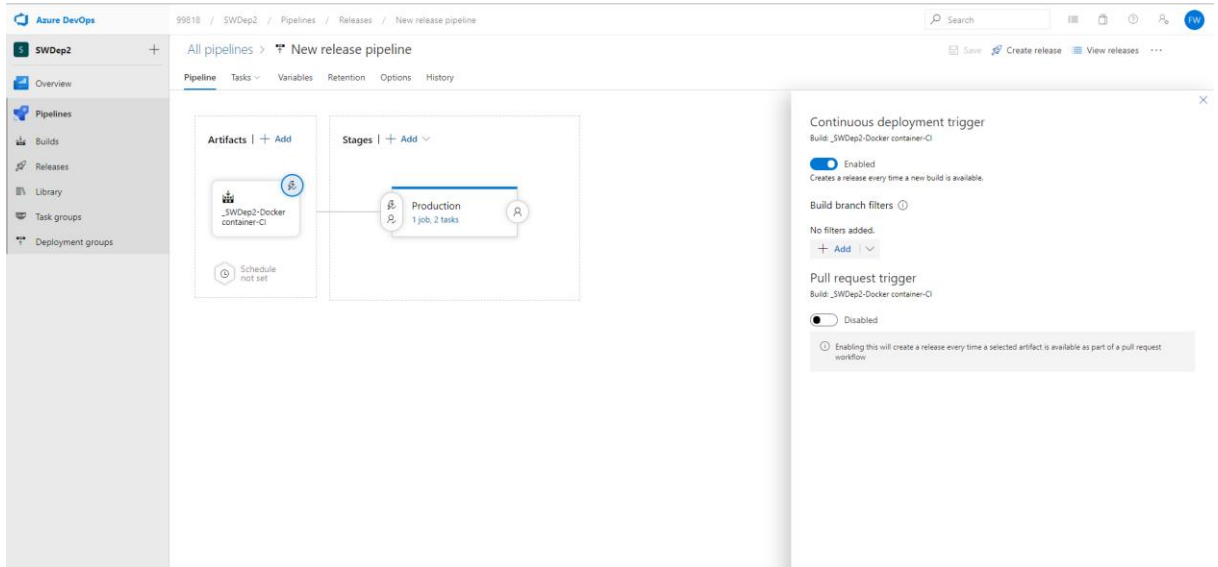
20. Speichern der Pipeline (Mitte-Oben) und Testen -> sollte auf Dockerhub eine Registry erstellen unter dem Account, mit der BuildNummer (kann getestet werden, mittels Erstellen eines Docker Containers basierend auf dem Image (pull command auf DockerHub nachsehen ! „:BuildNummer“ nicht vergessen!) -> docker run -d -p 80:3000 99818fhwn/nodeapp:31 -> IP checken und im Browser öffnen)

21. Erstellen einer Release Pipeline -> „Deploy to a Kubernetes cluster“ auswählen -> Stage benennen

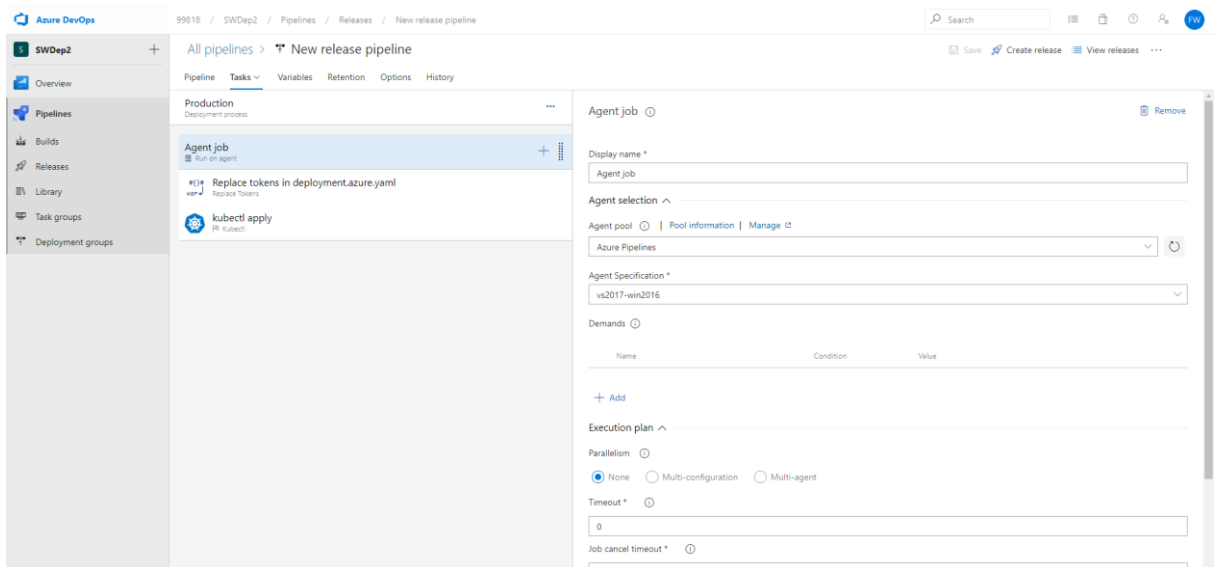
22. Auswählen des „Artifacts“ -> das verwendete aus der Build-Pipeline



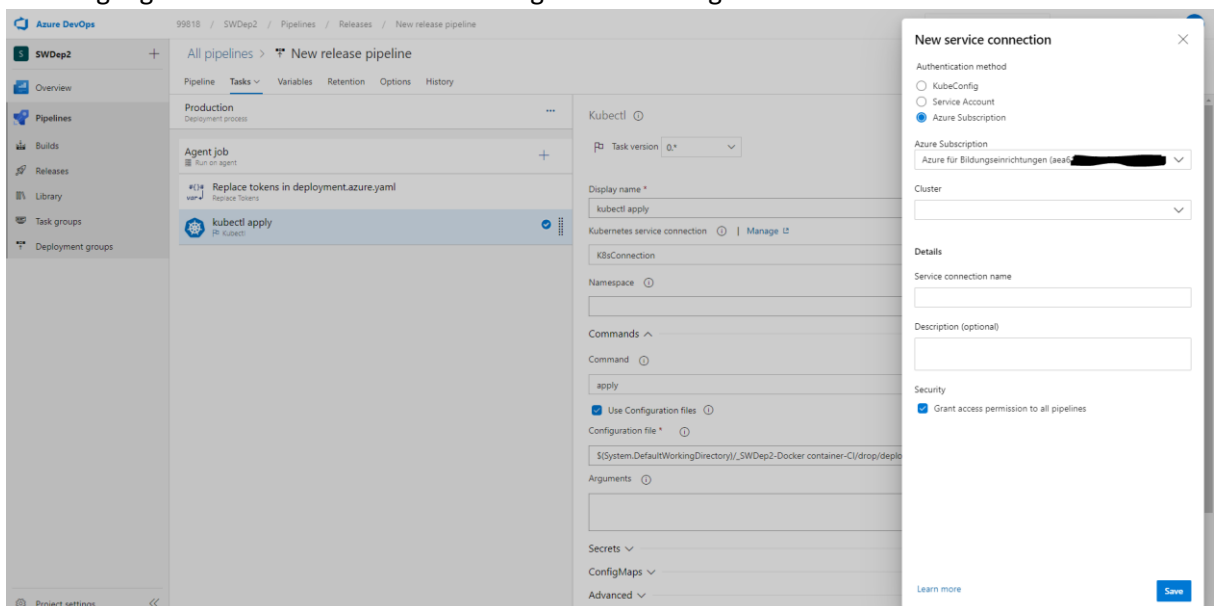
23. Trigger Setzen



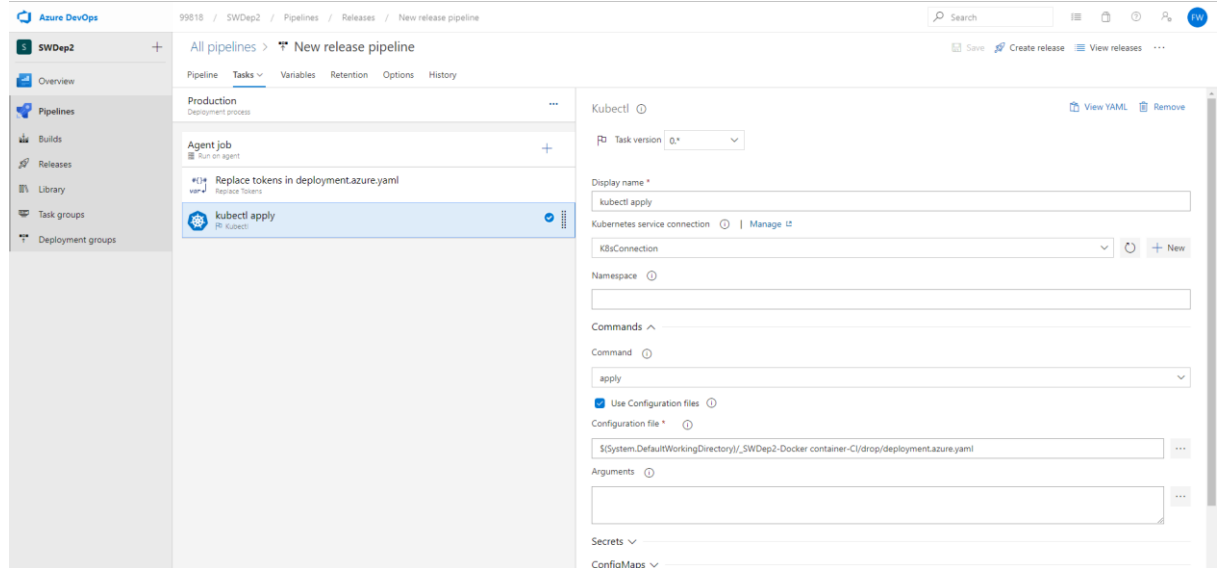
24. Stages bearbeiten (auf „1 job ...“ klicken)-> Agent Job überprüfen



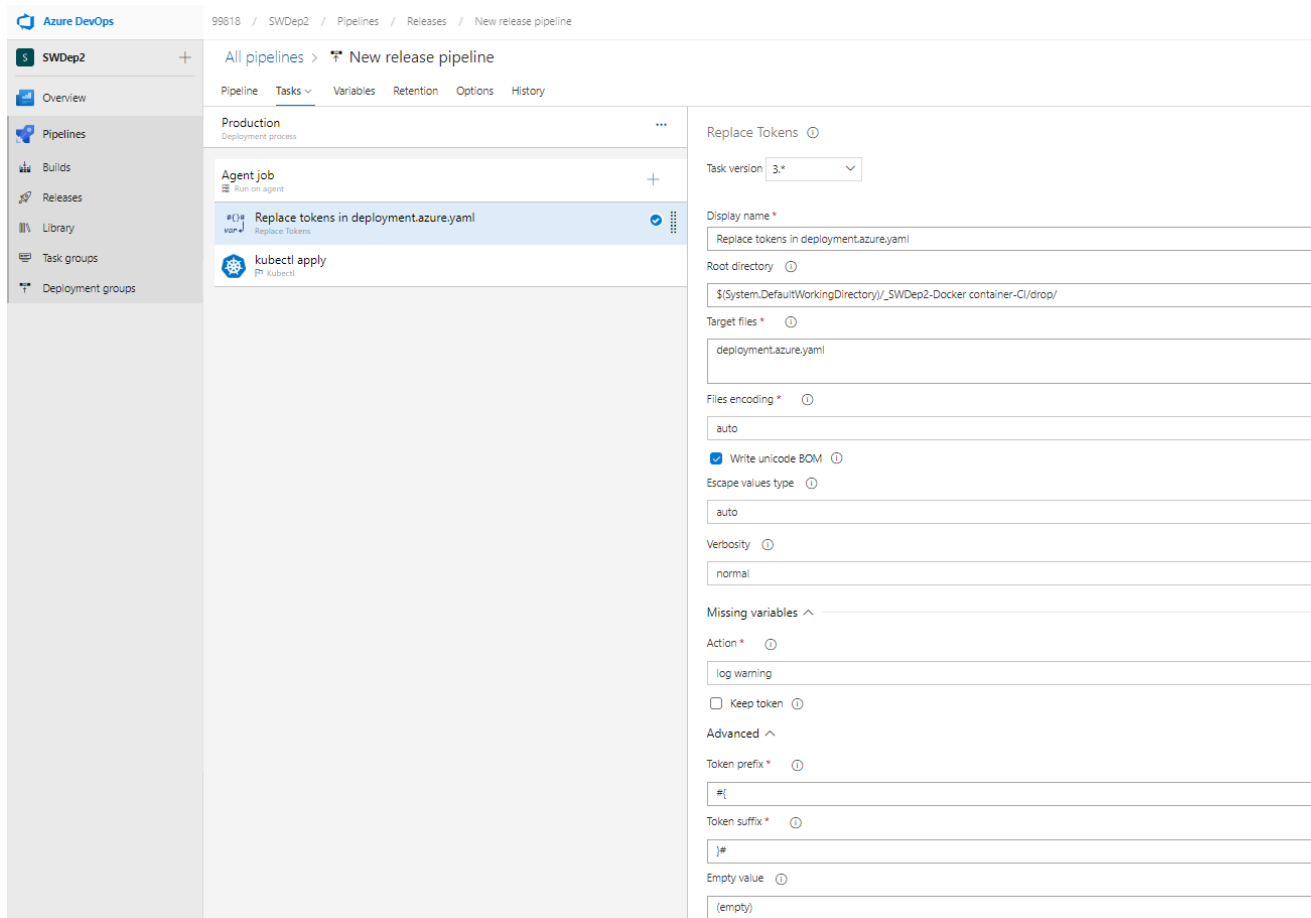
25. Anpassen des „kubectl apply“ -> „Kubernetes Service Connection“ erstellen basierend auf dem Zugang -> Cluster wählen -> Verbindungs-Name festlegen



- ➔ Unter „Command“ : „apply“ auswählen -> Aktivieren von „Use Configuration file“ -> Configuration File auswählen (falls die Hierarchie zu „deployment.azure.yaml“ nicht angezeigt wird wurde das Artefakt nicht gesetzt)



26. „Replace tokens“ dem Agent hinzufügen (,+‘ neben Agent job -> suche -> installieren) -> „Root directory“ anpassen -> „Target file“ auf „deployment.azure.yaml“ einstellen -> Unter „Advanced“ Token-Präfix/Suffix prüfen (dies erstellt aus dem Token die eigentliche Build-Tag-Nummer um den docker pull zu erzeugen)



27. Speichern der Release-Pipeline (Oben-Rechts)