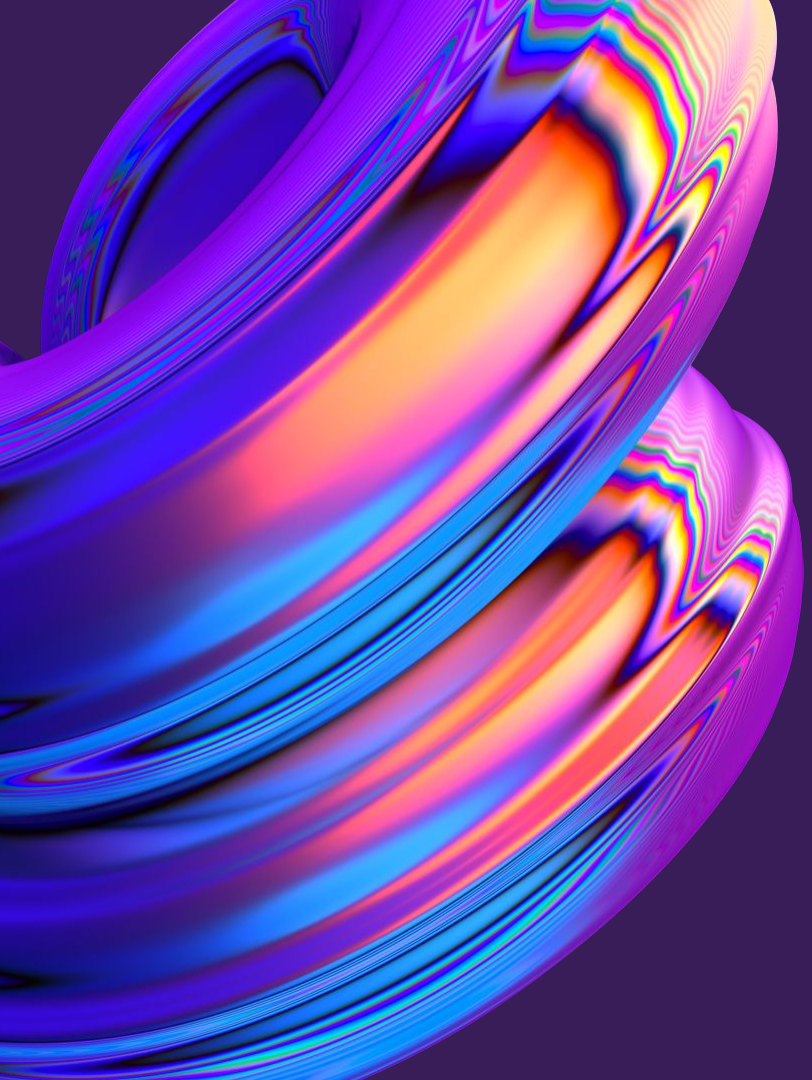
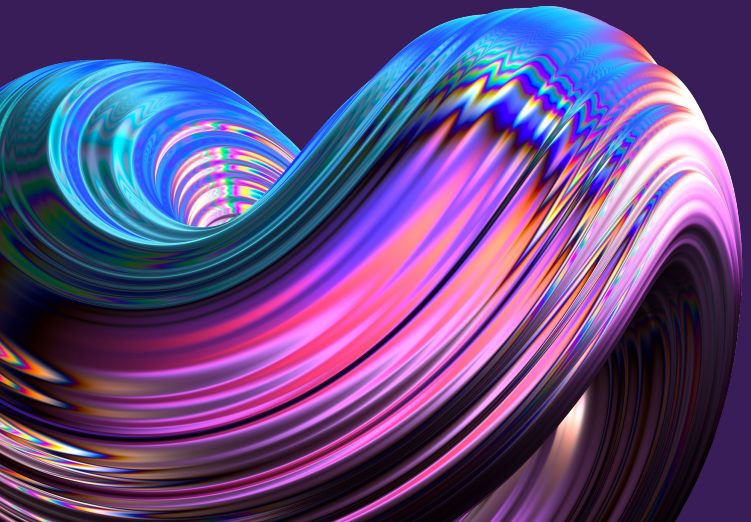


MERANTIX MOMENTUM

DB Systel MLOps Schulung

MERANTIX MOMENTUM



Session 1

Session 1 - *MLOps* und *CD4ML*

- Was ist MLOps und CD4ML?
- Maturitätsgrade in Machine Learning Systemen
- Vergleich ausgewählter Open Source (ML) Orchestration-Frameworks
- Hands-on Übung: *“Level 2 CICD Pipeline Automation”* mit Kubernetes, Argo und Flyte
    - Kurzeinführung Kubernetes
        - Was ist Kubernetes? Was sind Pods, Deployments und Services?
    - Einrichtung der Sandbox
    - Beispiel *“Level 1 Pipeline”*
    - Beispiel *“Level 2 Pipeline”* mit Continuous Delivery

Was ist MLOps und CD4ML?

Continuous Delivery for Machine Learning - Automating the end-to-end lifecycle of Machine Learning applications

Thoughtworks on martinFowler.com

MLOps: Continuous delivery and automation pipelines in machine learning

Google Cloud Architecture Center

Was ist MLOps und CD4ML?

These: Entwickelung, Deployment und die kontinuierliche Verbesserung ist bei traditioneller (nicht lernender) Software *viel* einfacher als bei Machine-Learning Systemen, bei denen es oft an Software-Engineering Best-Practices fehlt. *Klassische Software wird mit der Zeit besser - ML Systeme versagen katastrophal.*

Traditionelle Software

- Mögliche Änderungen
  - Code
- Software Release Workflow
  - Automatisierter Release-Prozess durch *Continuous Integration und Delivery* sichert Qualität, Disziplin und Reproduzierbarkeit

"*Continuous Delivery is the ability to get changes of all types — including new features, configuration changes, bug fixes, and experiments — into production, or into the hands of users, safely and quickly in a sustainable way". -- Jez Humble and Dave Farley*

Lernende Systeme

- Mögliche Änderungen
  - Model (Code und Hyperparameter)
  - (Glue) Code
  - Daten
- ML-Modell Release Workflow
  - Data engineers stellen Daten bereit
  - **Hand-Over**
  - Data Science team trainiert und evaluiert Modell in Jupyter-Notebook.
  - **Hand-Over**
  - ML-Engineering team dockerisiert Modell als Flask-App und operationalisiert es in der Cloud.

Was ist MLOps und CD4ML?

Organisations-Strukturen wie z.B. getrennte Verantwortlichkeiten und mehrere involvierte Teams sorgen oft für Reibung und langsame Release-Prozesse

Viele ML-Modelle

- “funktionieren nicht” mit Daten aus der echten Welt oder es gibt keine Visibilität bzgl. Modell-Performance im Betrieb
- verlassen nie die PoC-Phase.
- werden manuell released und können nur schwer geupdatet werden.
- sind teure “Science Projects” die keinen echten Business-Wert liefern

Ziel

“Data Scientist A entwickelt und evaluiert ein Modell mit Daten von Data Engineer B und übergibt das beste Modell an ML-Engineer C, der das Modell deployt. Wenn Data Scientist A im Urlaub ist, kann das Modell nicht geupdatet werden.”

“Ein Hyperparameter wird geändert und der komplette Prozess bis hin zum deployten Modell läuft vollautomatisch durch.”

Was ist MLOps und CD4ML?

“Continuous Delivery for Machine Learning (CD4ML) is a software engineering approach in which a cross-functional team produces machine learning applications based on code, data, and models in small and safe increments that can be reproduced and reliably released at any time, in short adaptation cycles.”

Thoughtworks

MLOps is [...] practice that aims at unifying ML system development (Dev) and ML system operation (Ops). Practicing MLOps means that you advocate for automation and monitoring at all steps of ML system construction, including integration, testing, releasing, deployment and infrastructure management.

“MLOps: Continuous delivery and automation pipelines in machine learning, Google Cloud Architecture Center”

Was ist MLOps und CD4ML?

“Machine Learning Operations (MLOps): Overview, Definition, and Architecture, Kreuzberger et al. 2022 definieren den Term “MLOps” als:

- “[…] a paradigm […], development culture, […] engineering practice”
- “[…] aimed at productionizing machine learning systems by bridging the gap between development (Dev) and operations (Ops)”
- “[…] leveraging […]:
  - CI/CD automation, workflow orchestration, reproducibility; versioning of data, model, and code; continuous ML training and evaluation; ML metadata tracking and logging; continuous monitoring; and feedback loops.”

Maturitätsgrade in Machine Learning Systemen

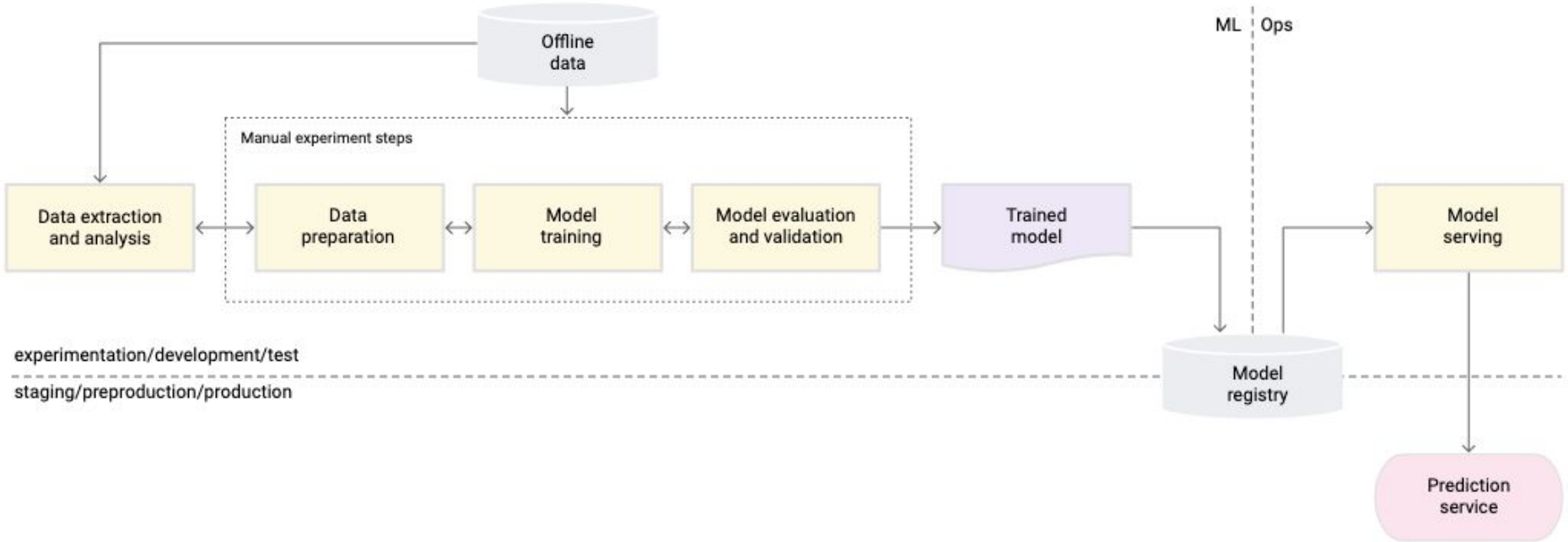
- Drei Maturitätsgrade definiert in *“MLOps: Continuous delivery and automation pipelines in machine learning, Google Cloud Architecture Center”*

- Maturität des ML-Prozesses durch Automatisierungsgrad definiert
  - Geschwindigkeit für neue Daten ein neues Modell zu trainieren und zu deployen
  - Geschwindigkeit mit einem neuen Ansatz/einer neue Modell-Implementierung ein neues Modell zu trainieren und zu deployen

- Ziel
  - Entwicklungszyklen verkürzen
  - “Time to deployed service” reduzieren
  - Manuelle Arbeit reduzieren

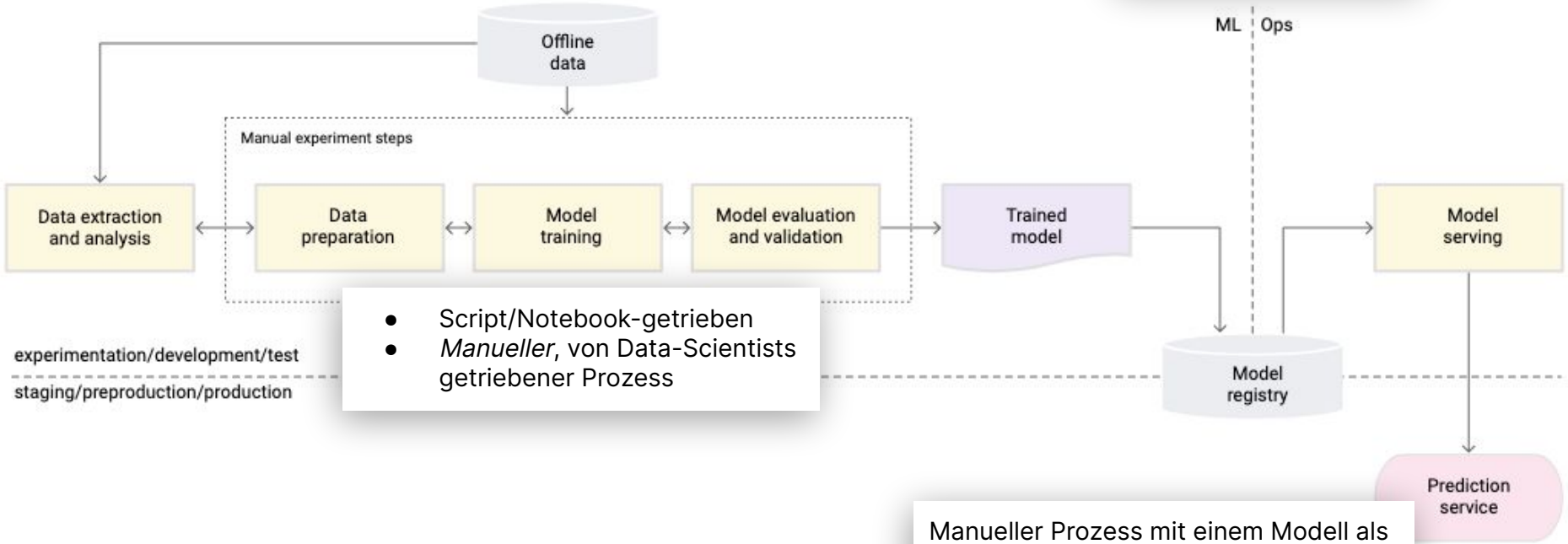
Die eigentliche Herausforderung für Organisationen ist nicht ein Modell zu trainieren, sondern ein integriertes ML-System kontinuierlich in Produktion zu operationalisieren und damit Geld zu verdienen.

Maturitätsgrade in Machine Learning Systemen
MLOps level 0: Manueller Prozess



Maturitätsgrade in Machine Learning Systemen
MLOps level 0: Manueller Prozess

Trennung zwischen “ML” und “Ops”



- Script/Notebook-getrieben
- *Manueller*, von Data-Scientists getriebener Prozess

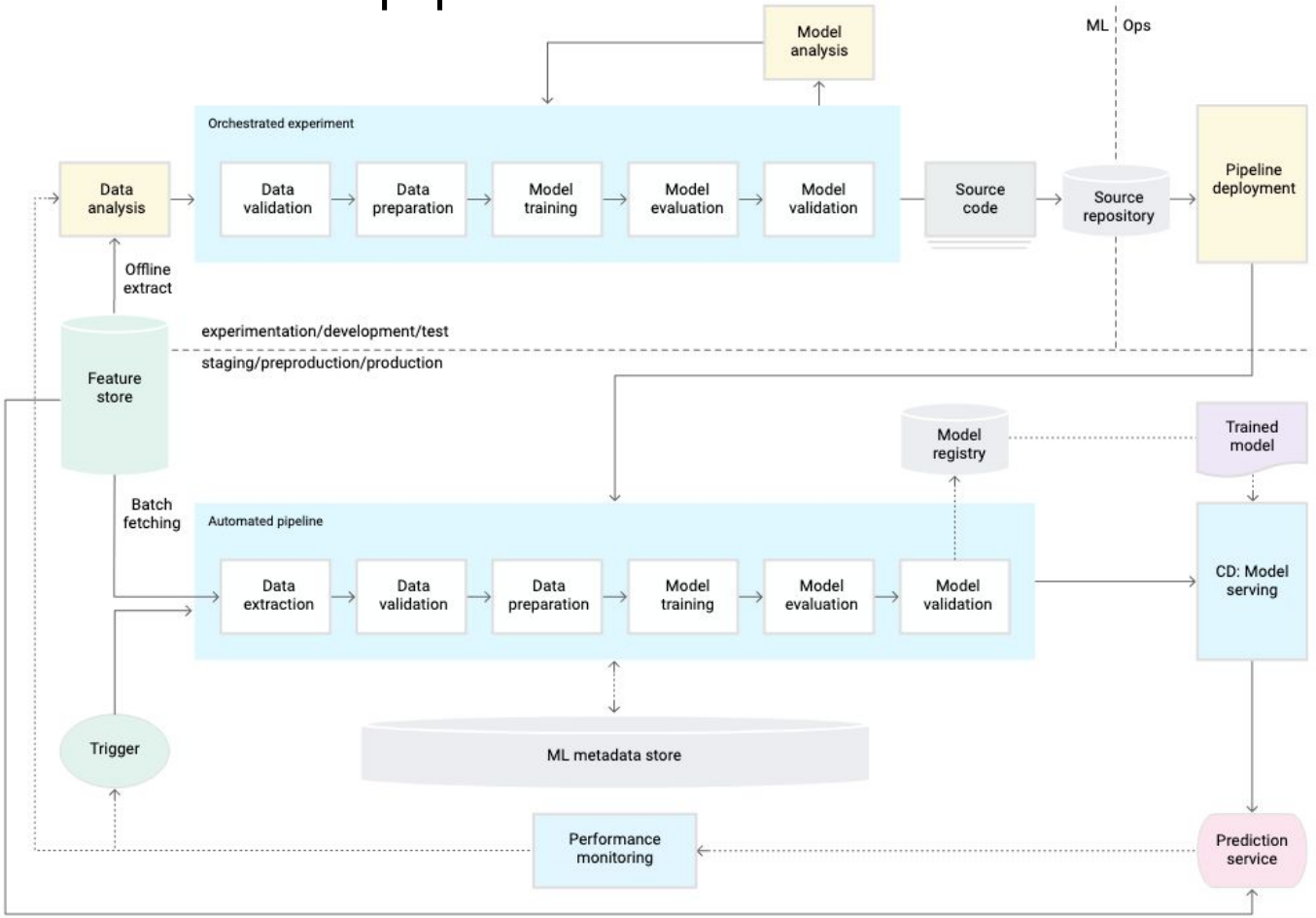
Manueller Prozess mit einem Modell als Prediction-Service als Ziel

Maturitätsgrade in Machine Learning Systemen
MLOps level 0: Manueller Prozess

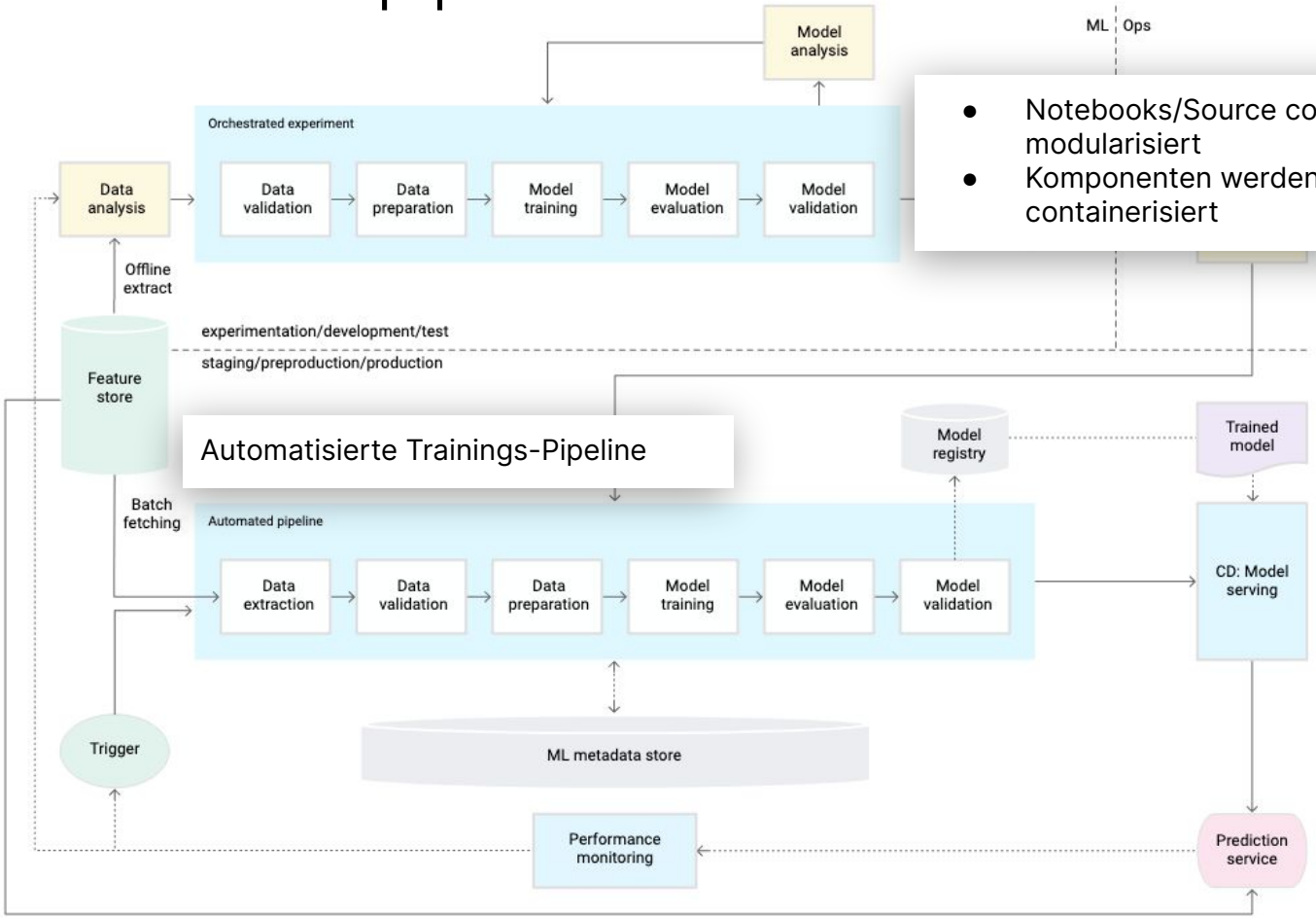
- **Eigenschaften**
  - Oft in Unternehmen oder Teams, die beginnen ML auf Ihre Use-Cases anzuwenden
  - Kein CI
    - Annahme: Implementierung wird selten geändert
    - Tests passieren idR innerhalb von Notebooks oder Scripten
  - Kein CD
    - Annahme: Das Modell wird nur selten geupdatet
  - Keine Modularisierung
  - Keine Automatisierung: Manuelle Ausführung der Schritte und manuelle Operationalisierung

- **Probleme**
  - Das Modell wird operationalisiert, nicht das ganze ML-System
  - Viele Modelle bleiben PoCs, da sie in der echten Welt nicht funktionieren und nicht schnell iteriert werden kann
  - Kein Logging und Performance-Monitoring
    - Keine Möglichkeit “Concept Drift” oder “Modell-Decay” zu detektieren
  - Teuer
    - nicht automatisiert
    - manchmal kein ROI
  - Abhängig von Personen (Urlaub, Krankheit, Kündigung)

MLOps level 1: ML pipeline automation



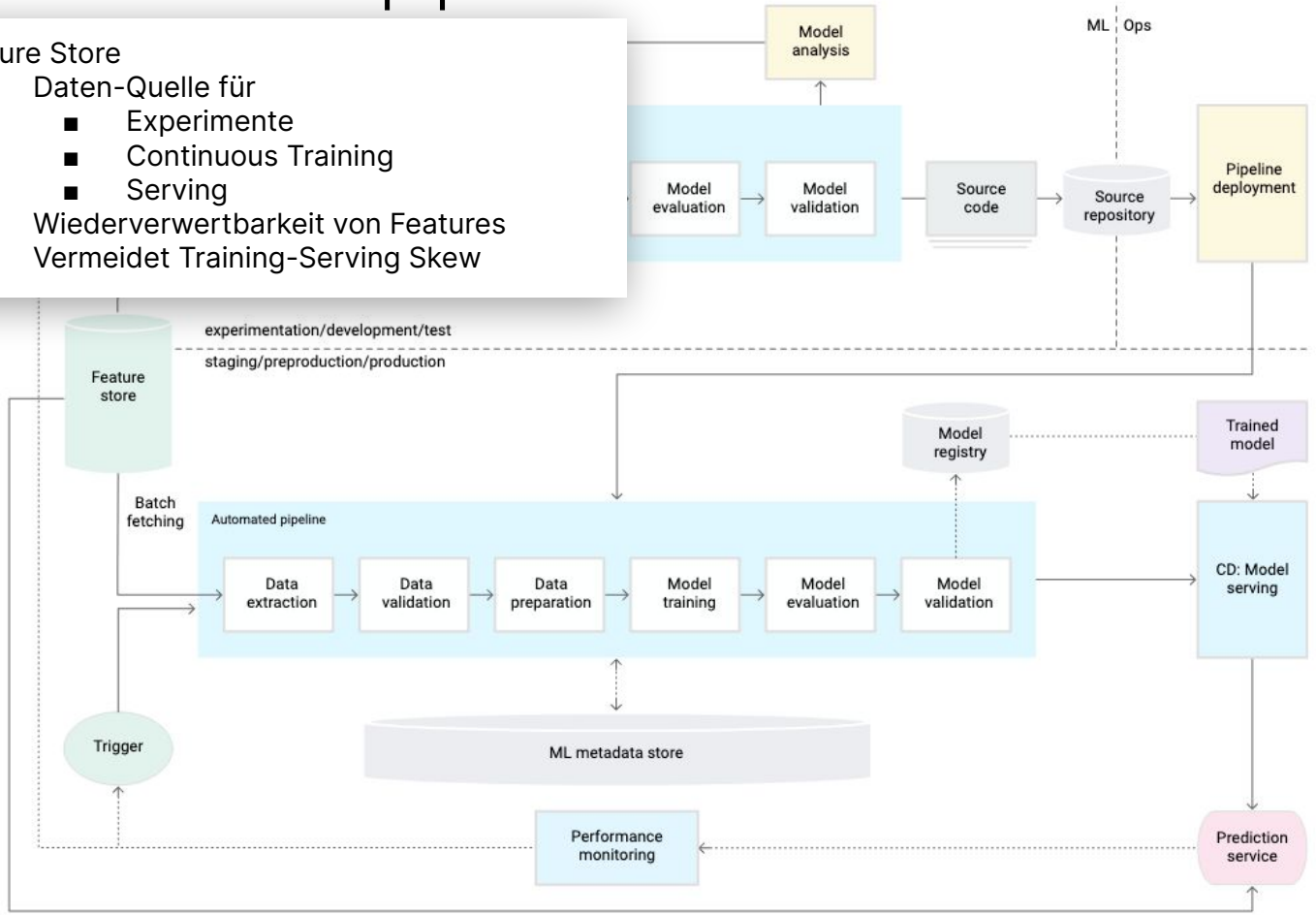
MLOps level 1: ML pipeline automation



- Notebooks/Source code wird modularisiert
- Komponenten werden containerisiert

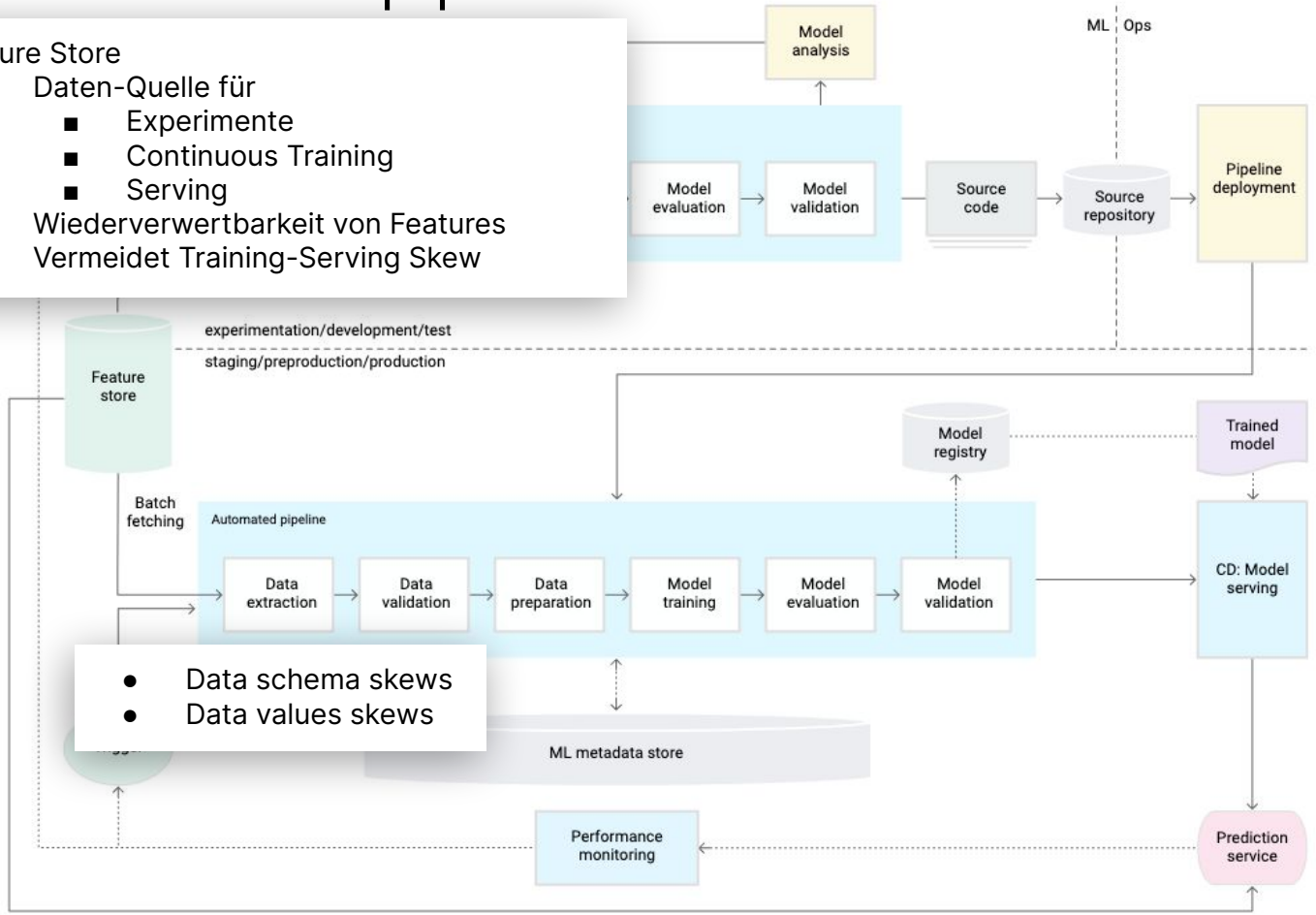
MLOps level 1: ML pipeline automation

- Feature Store
  - Daten-Quelle für
    - Experimente
    - Continuous Training
    - Serving
  - Wiederverwertbarkeit von Features
  - Vermeidet Training-Serving Skew



MLOps level 1: ML pipeline automation

- Feature Store
  - Daten-Quelle für
    - Experimente
    - Continuous Training
    - Serving
  - Wiederverwertbarkeit von Features
  - Vermeidet Training-Serving Skew

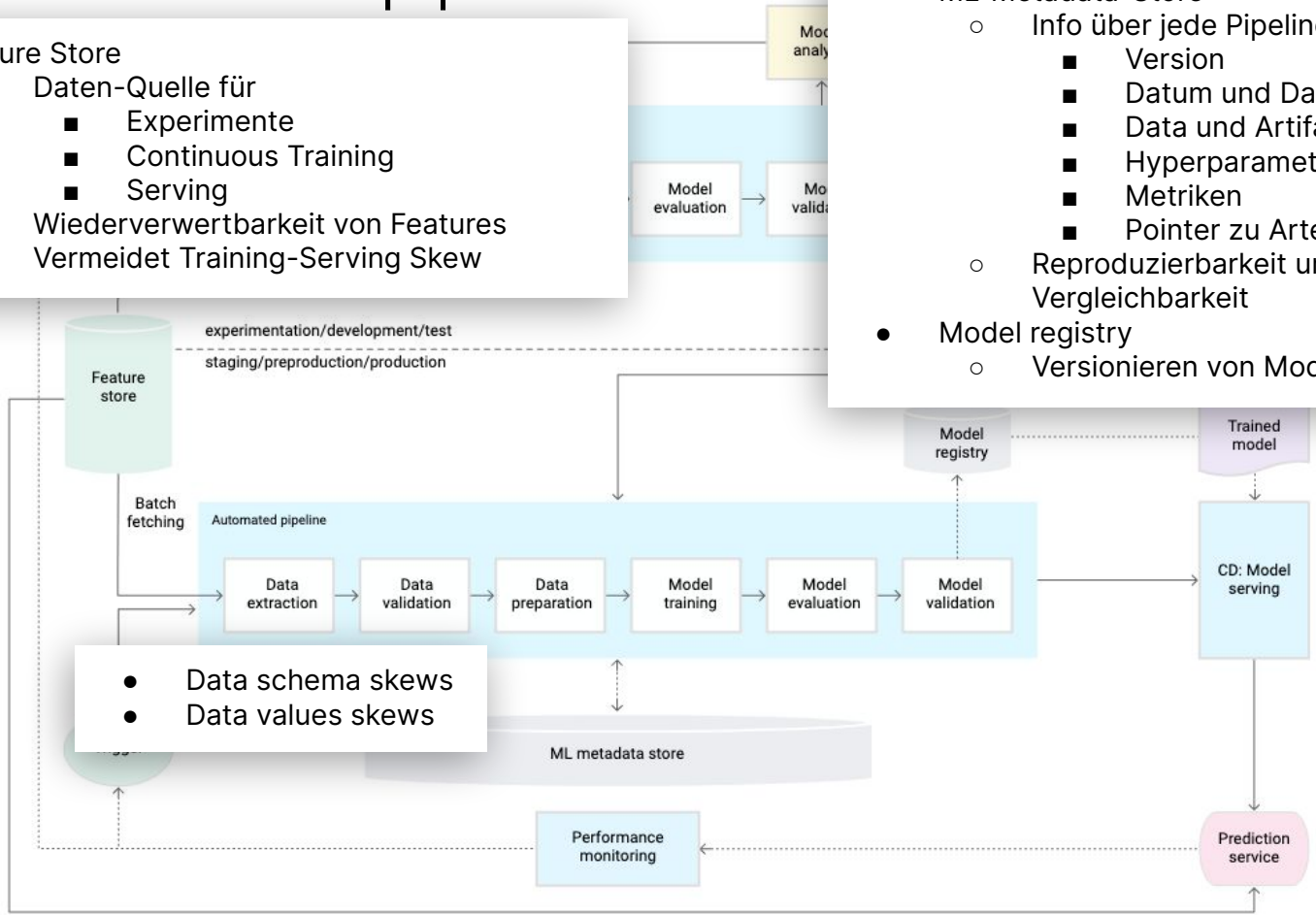


- Data schema skews
- Data values skews

MLOps level 1: ML pipeline autom

- Feature Store
  - Daten-Quelle für
    - Experimente
    - Continuous Training
    - Serving
  - Wiederverwertbarkeit von Features
  - Vermeidet Training-Serving Skew

- ML-Metadata-Store
  - Info über jede Pipeline-Exekution
    - Version
    - Datum und Dauer
    - Data und Artifact Lineage
    - Hyperparameter
    - Metriken
    - Pointer zu Artefakten
  - Reproduzierbarkeit und Vergleichbarkeit
- Model registry
  - Versionieren von Modellen

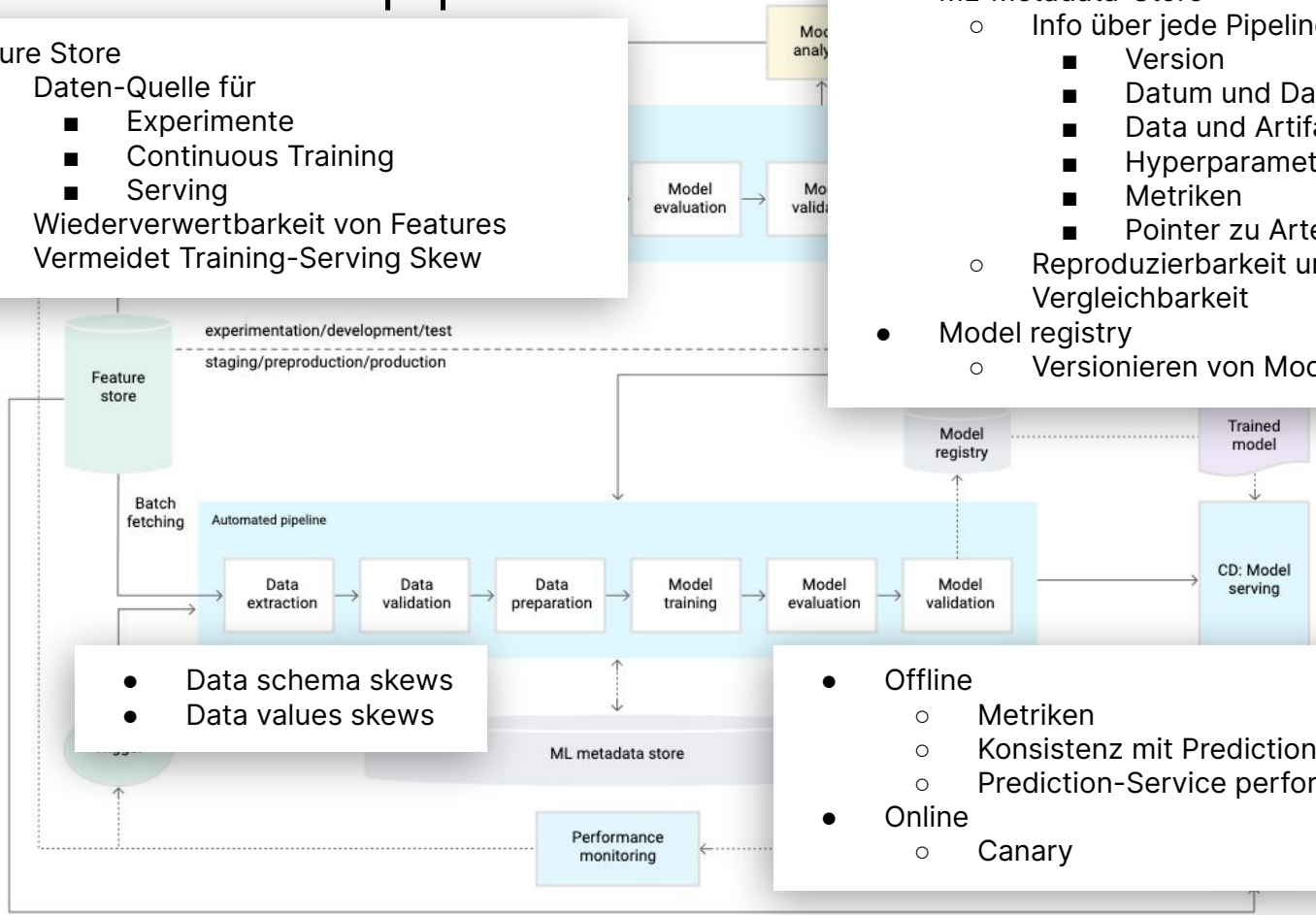


- Data schema skews
- Data values skews

MLOps level 1: ML pipeline autom

- Feature Store
  - Daten-Quelle für
    - Experimente
    - Continuous Training
    - Serving
  - Wiederverwertbarkeit von Features
  - Vermeidet Training-Serving Skew

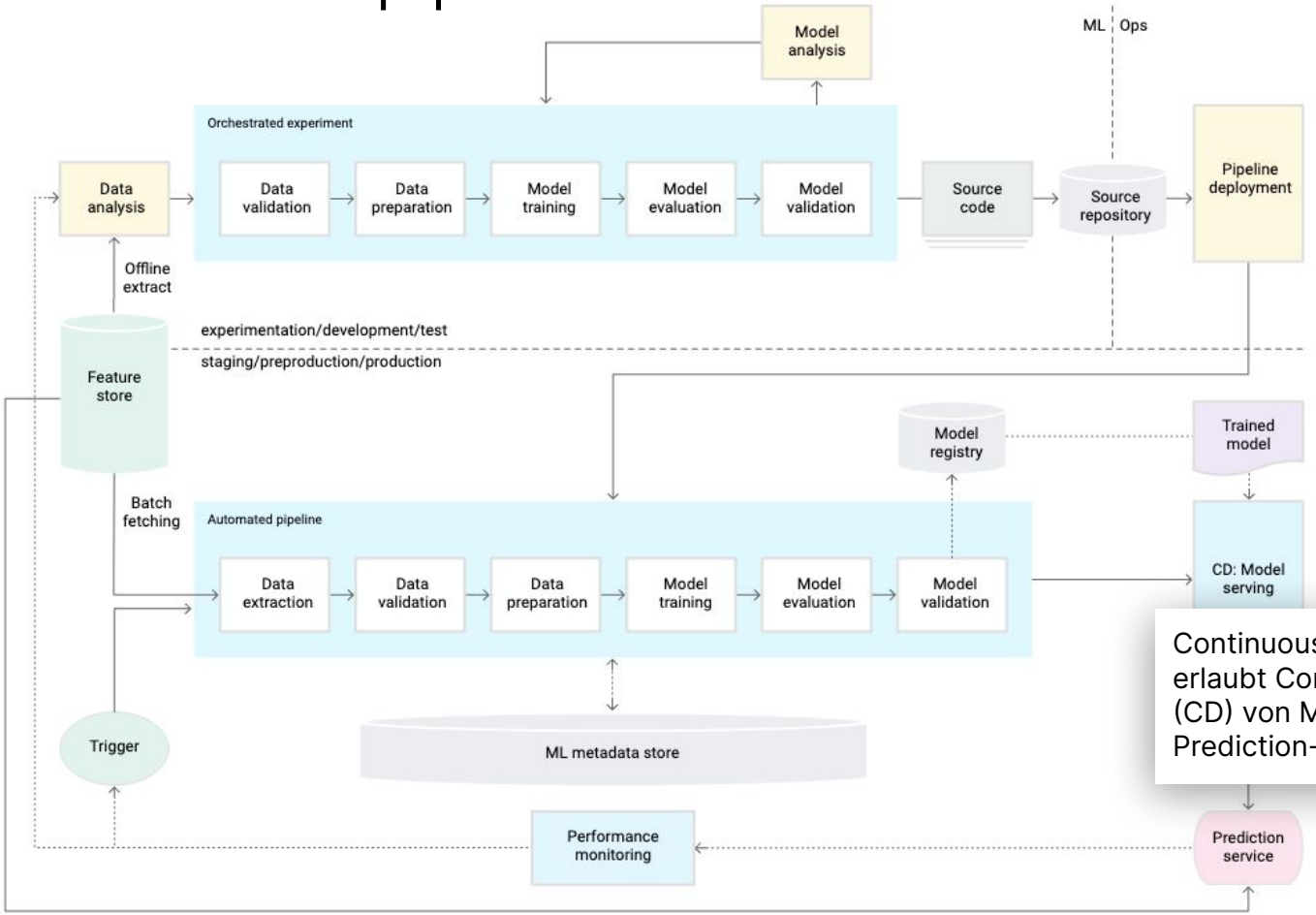
- ML-Metadata-Store
  - Info über jede Pipeline-Exekution
    - Version
    - Datum und Dauer
    - Data und Artifact Lineage
    - Hyperparameter
    - Metriken
    - Pointer zu Artefakten
  - Reproduzierbarkeit und Vergleichbarkeit
- Model registry
  - Versionieren von Modellen



- Data schema skews
- Data values skews

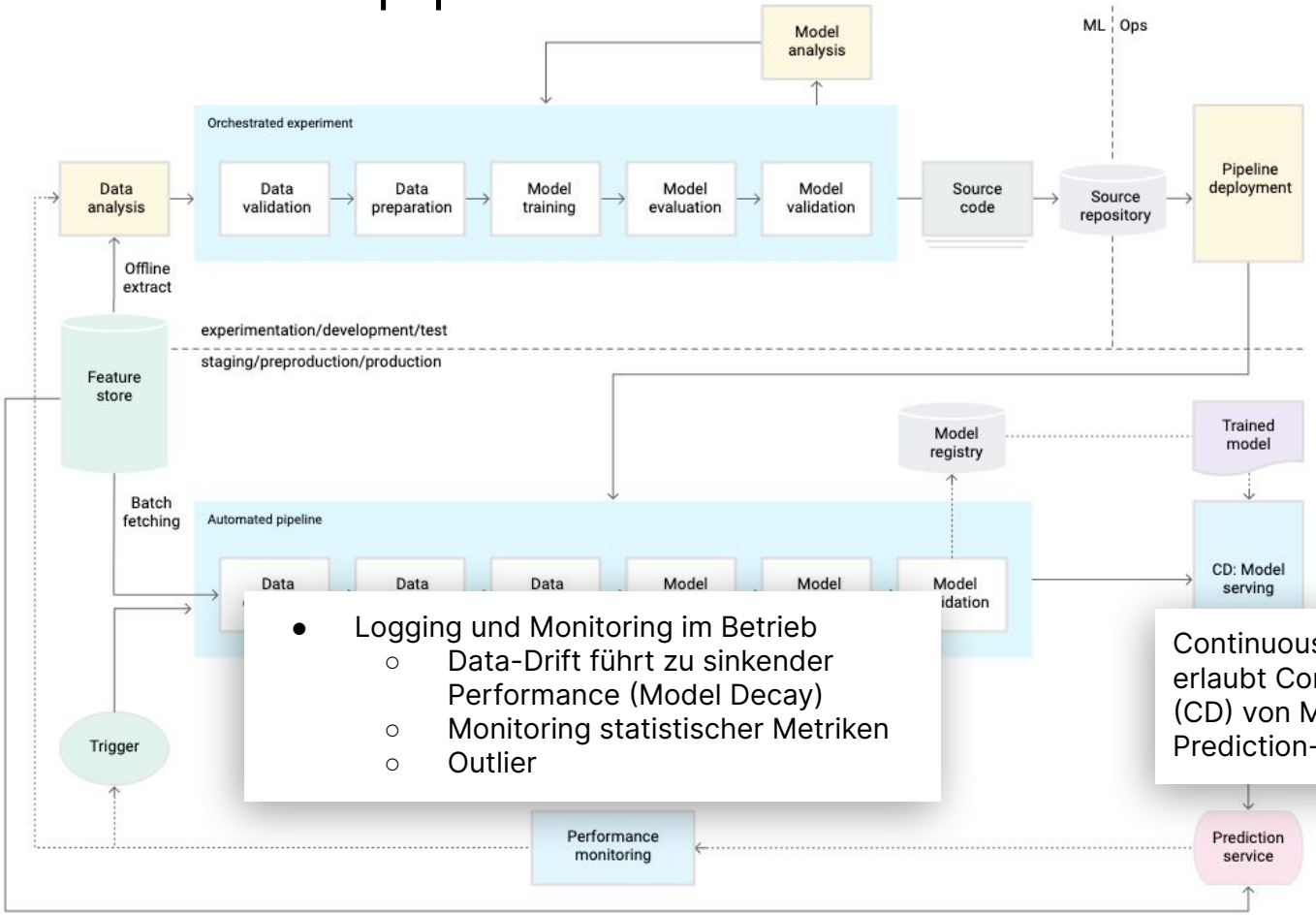
- Offline
  - Metriken
  - Konsistenz mit Prediction API
  - Prediction-Service performance/latency
- Online
  - Canary

MLOps level 1: ML pipeline automation



Continuous Training (CT) erlaubt Continuous Delivery (CD) von Model Prediction-Services.

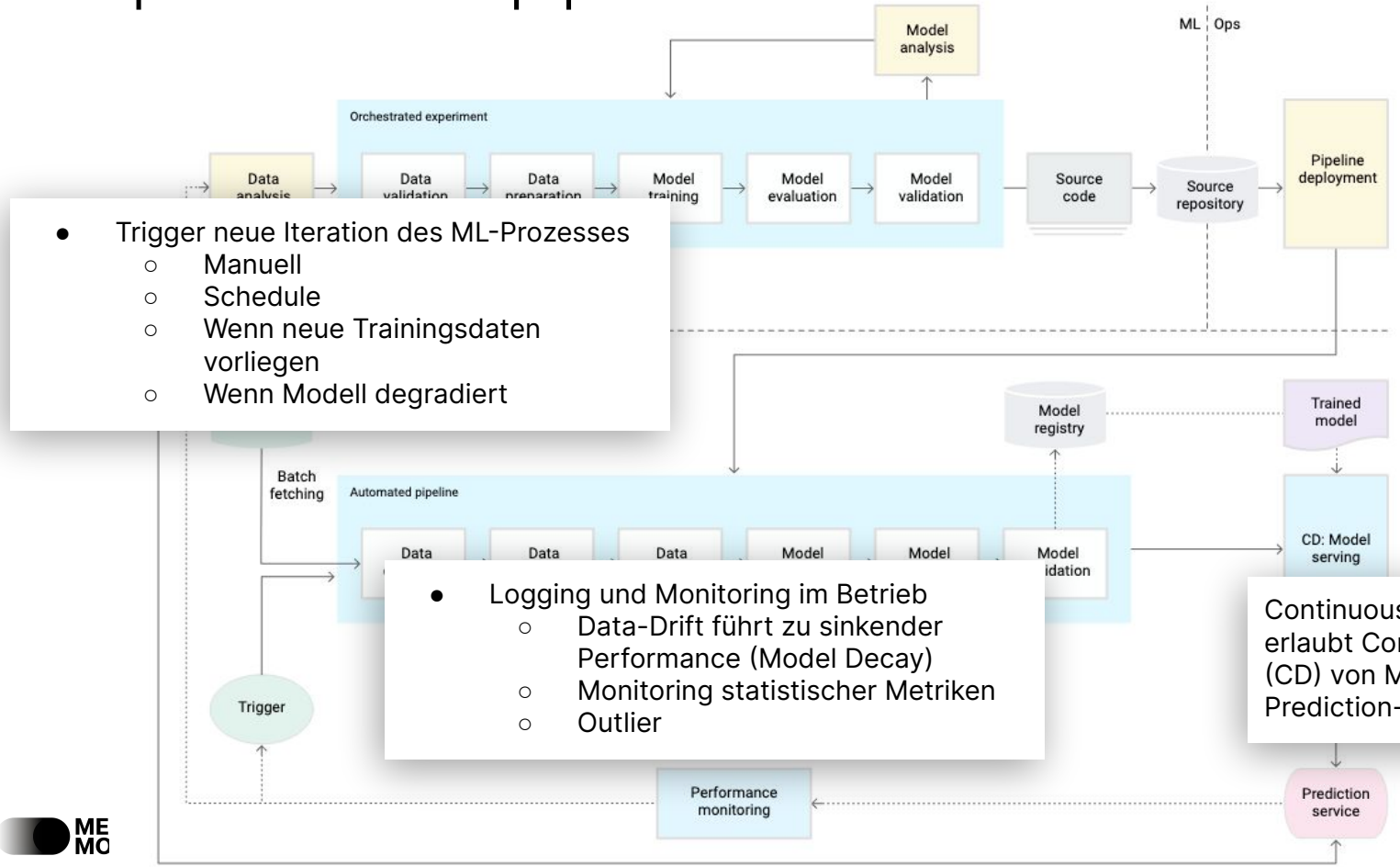
MLOps level 1: ML pipeline automation



- Logging und Monitoring im Betrieb
  - Data-Drift führt zu sinkender Performance (Model Decay)
  - Monitoring statistischer Metriken
  - Outlier

Continuous Training (CT) erlaubt Continuous Delivery (CD) von Model Prediction-Services.

MLOps level 1: ML pipeline automation

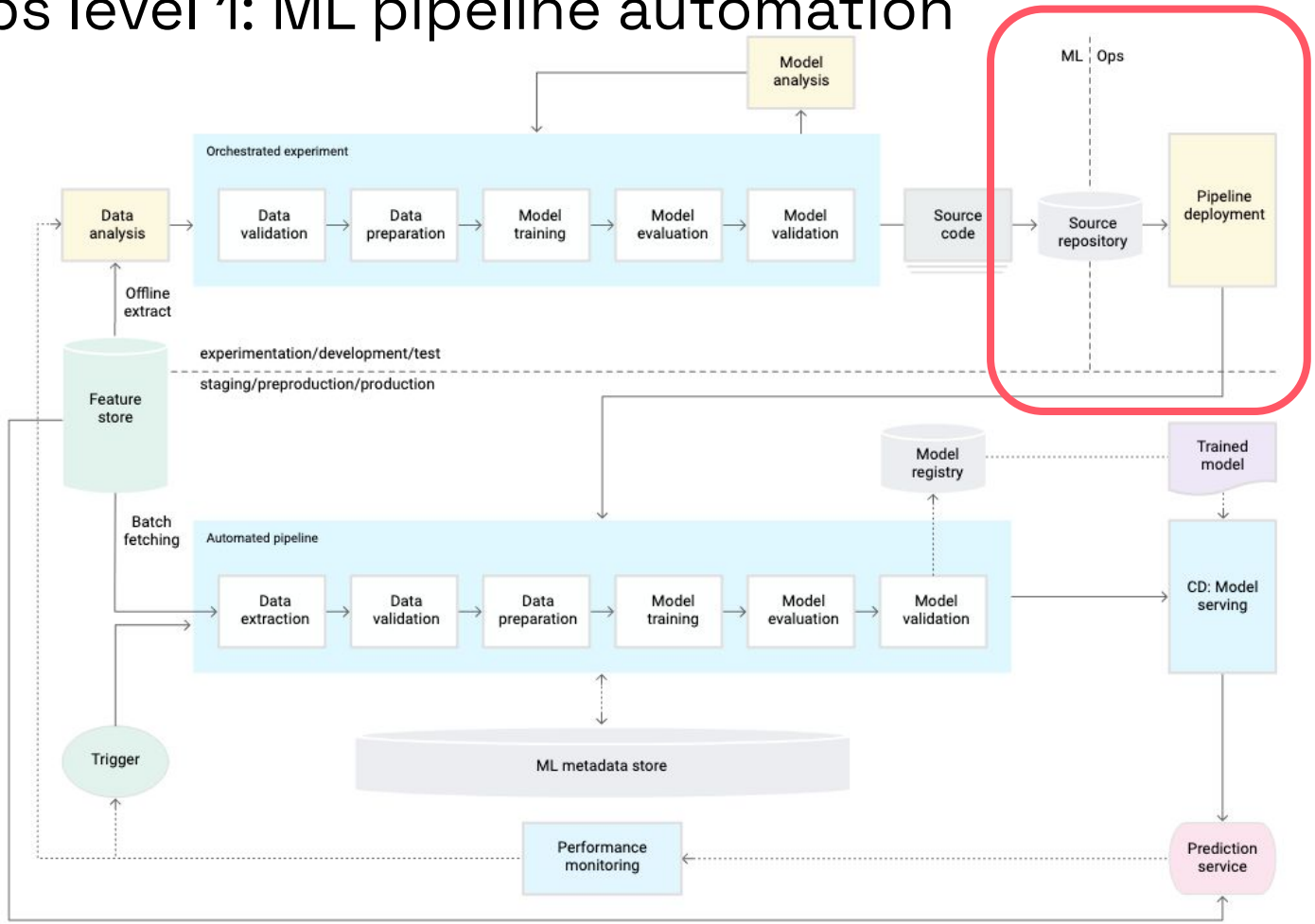


- Trigger neue Iteration des ML-Prozesses
  - Manuell
  - Schedule
  - Wenn neue Trainingsdaten vorliegen
  - Wenn Modell degradiert

- Logging und Monitoring im Betrieb
  - Data-Drift führt zu sinkender Performance (Model Decay)
  - Monitoring statistischer Metriken
  - Outlier

Continuous Training (CT) erlaubt Continuous Delivery (CD) von Model Prediction-Services.

MLOps level 1: ML pipeline automation

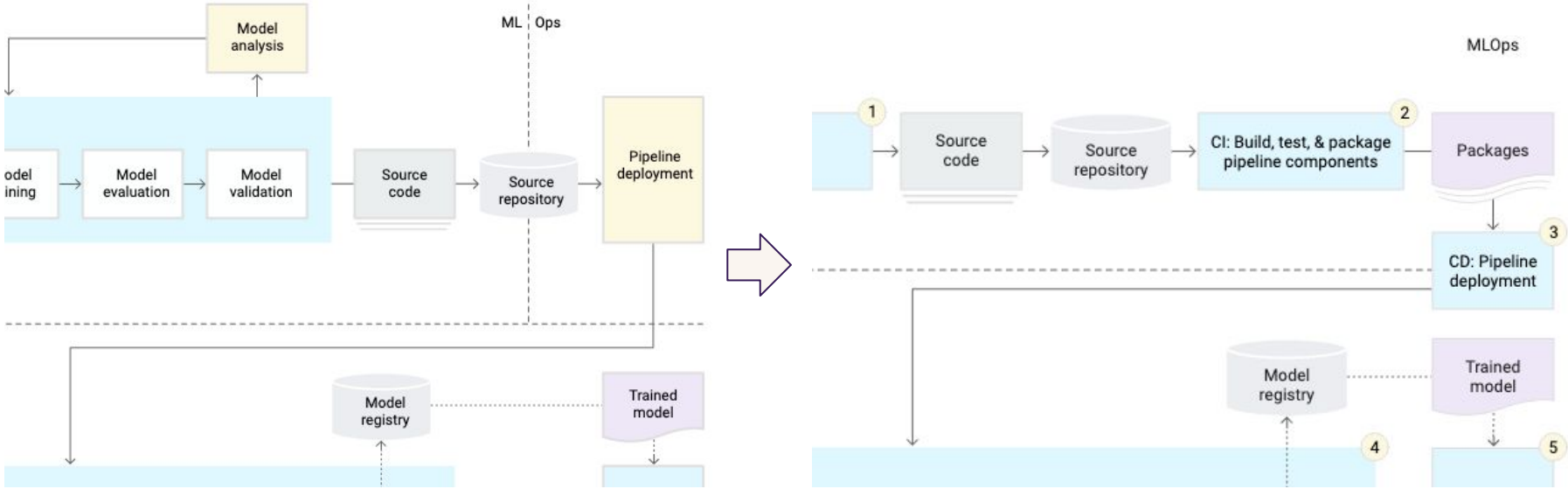


MLOps level 1: ML pipeline automation

- **Eigenschaften**
  - Pipeline deployment:
    - Level 0: Manuelles Deployment eines Modells/Prediction Services
    - Level 1: Manuelles Deployment einer Trainings-Pipeline, die automatisch Modelle bereitstellt
  - Selbe Pipeline während Experimenten und in Produktion
    - Vereinheitlicht Development und Operations
  - Schnelleres Iterieren und Experimentieren

- **Probleme**
  - Manuelles Testen und Deployen einer neuen Pipeline-Version und ihrer Komponenten
    - Annahme: nicht zu oft und nicht zu viele Pipelines *da manueller Schritt*
    - Geeignet wenn neue Modelle auf neuen Daten basieren und nicht neuen ML-Ansätzen

MLOps level 2: CI/CD pipeline automation



Level 1

- Komponenten und Pipelines werden manuell getestet und deployt
- u.U. von verschiedenen Teams

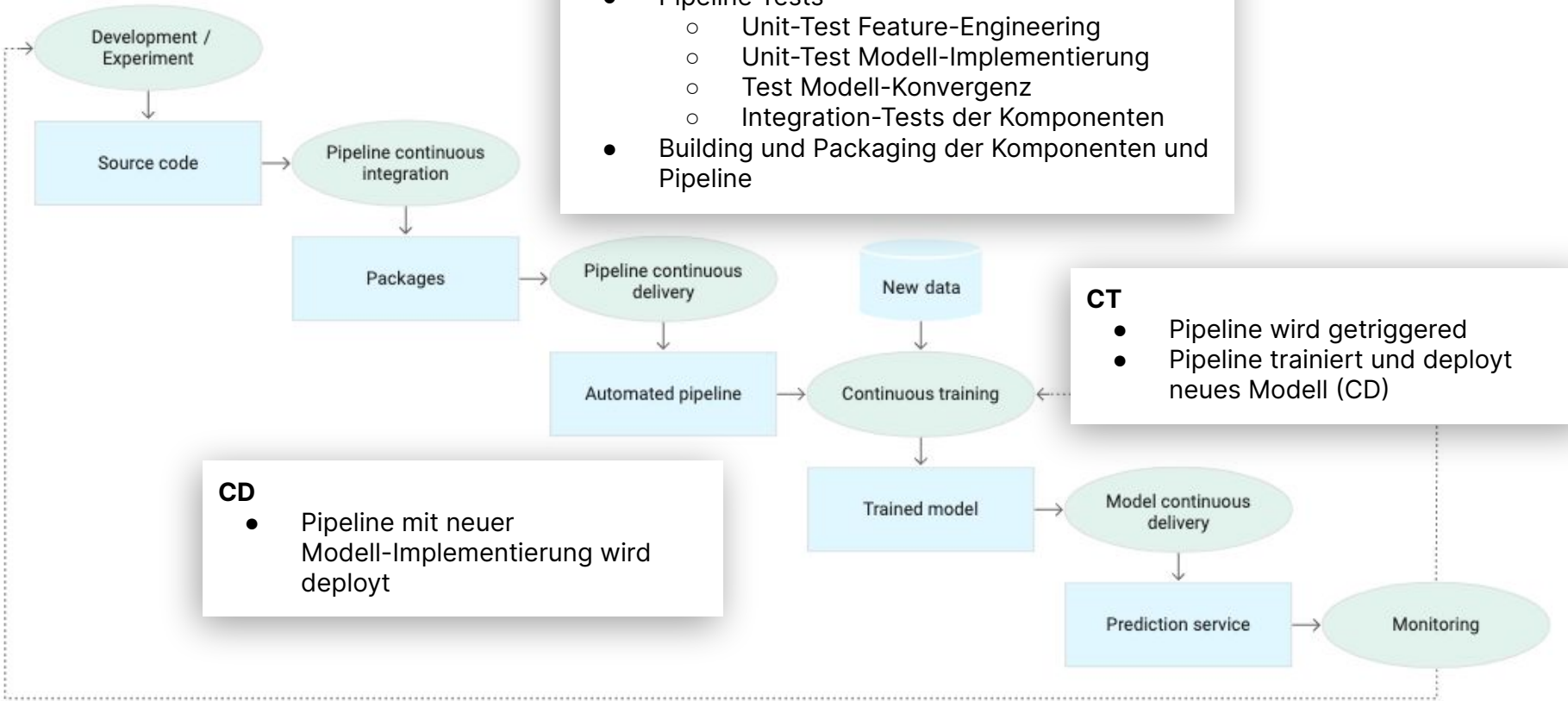
Level 2

- Komponenten und Pipelines werden automatisch in CI getestet und in CD deployt

MLOps level 2: CI/CD pipeline automation

CI

- Pipeline Tests
  - Unit-Test Feature-Engineering
  - Unit-Test Modell-Implementierung
  - Test Modell-Konvergenz
  - Integration-Tests der Komponenten
- Building und Packaging der Komponenten und Pipeline



CT

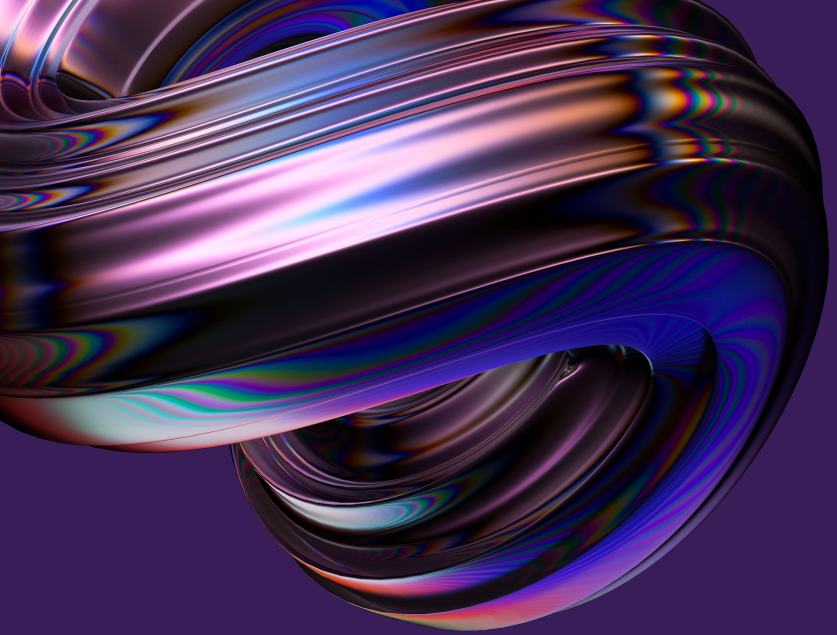
- Pipeline wird getriggered
- Pipeline trainiert und deployt neues Modell (CD)

CD

- Pipeline mit neuer Modell-Implementierung wird deployt

MLOps level 2: CI/CD pipeline automation

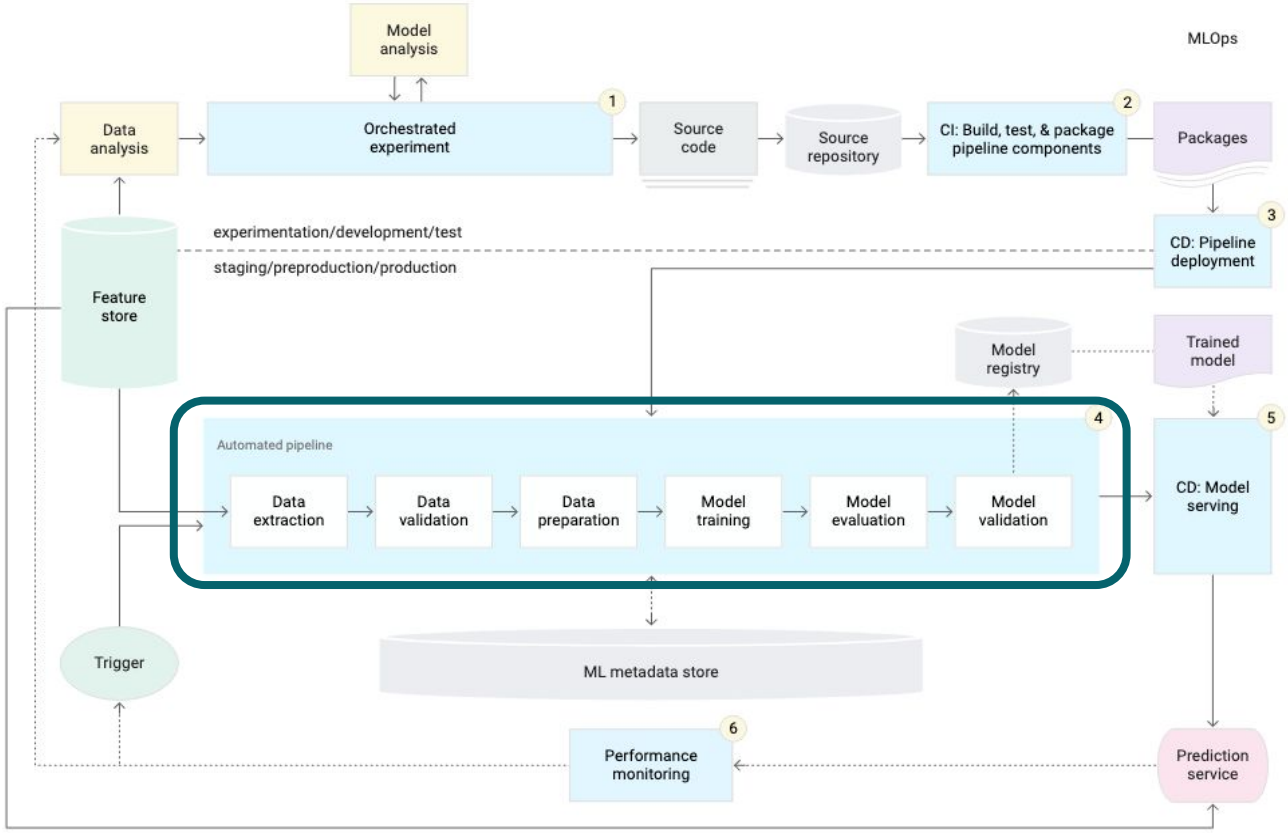
- **Eigenschaften**
  - Pipeline deployment:
    - Level 1: Manuelles Deployment einer Trainings-Pipeline, die automatisch Modelle deployt
    - Level 2: CI/CD System, das automatisiertes Testen und Deployen neuer Pipeline-Implementierungen erlaubt
  - System, das manuelle Arbeit und “Time to deployed service” minimiert und Reproduzierbarkeit und Automatisierung maximiert
  - Erlaubt schnelles Testen neuer Ideen, dadurch iteratives Verbessern und gleichzeitig rapides und verlässliches Updaten von ML-Systemen in Produktion
- **Probleme**
  - *“Klingt in der Theorie gut aber auch sehr kompliziert. Wie setzt man das um?”*



MERANTIX MOMENTUM

Vergleich OSS (ML) Orchestration-Frameworks

Vergleich ausgewählter Open Source (ML) Orchestration-Frameworks



Kubeflow

Flyte

PREFECT

argo TEKTON

Apache Airflow

METAFLOW

...

Vergleich ausgewählter Open Source Orchestration-Frameworks

- **Airflow**
  - Pro:
    - Sehr mature (Top-Level Apache Software Foundation project seit Januar 2019)
    - Sehr mächtig mit vielen Funktionen
    - Funktioniert mit Kubernetes (Helm chart)
    - Python
  - Contra:
    - Es gibt Tools speziell für Machine Learning, die komfortabler und einfacher zu benutzen sind

Vergleich ausgewählter Open Source Orchestration-Frameworks

- **Kubeflow pipelines**
  - Pro:
    - Kubeflow hat viel Funktionalität für Machine Learning
      - Notebook Server
      - KFServing
      - Hyperparameter Tuning
      - Training Operators für Distributed Training
    - Basiert auf Kubernetes
    - Python (wird in YAML konvertiert)
  - Contra:
    - Installation und Verwaltung hat lange Zeit sehr viel Kubernetes Wissen vorausgesetzt
    - Unübersichtlicher, wenig dokumentierter Code, wenig und veraltete Dokumentation, unübersichtliche API (persönliche Meinung)
    - Schlechte Integration mit Kubeflow Training Operators (z.B. nur mit TfJob)
    - Zitat eines unserer Partner: *“Kubeflow ist eine lose Sammlung von mehr oder weniger guten Komponenten, die nicht miteinander funktionieren.”*

Vergleich ausgewählter Open Source Orchestration-Frameworks

- **Metaflow**
  - Von Netflix entwickelt, seit 2019 Open Source
  - Pro:
    - Sehr gute Integration mit AWS
      - Amazon S3
      - AWS Batch
      - AWS Fargate + Amazon RDS (Metadata Store)
      - Amazon Sagemaker Notebooks
      - AWS Step Functions + Amazon EventBridge (Scheduling)
      - Sagemaker Models
  - Contra:
    - Bis vor kurzem nur in Verbindung mit AWS. Jetzt auch auf Kubernetes aber noch nicht mature

Vergleich ausgewählter Open Source Orchestration-Frameworks

- **Argo/Tekton**
  - Pro:
    - Kubernetes
    - Haben alle für die Task-Orchestrierung benötigten Funktionen
      - Workflow CRD
      - Event Manager
    - Declarative Continuous Delivery nach dem Gitops Prinzip
    - Dashboard
    - Funktionieren gut
    - Fully Open Source
    - Große Community
  - Contra:
    - Workflows werden in YAML und nicht in Python konfiguriert

Vergleich ausgewählter Open Source Orchestration-Frameworks

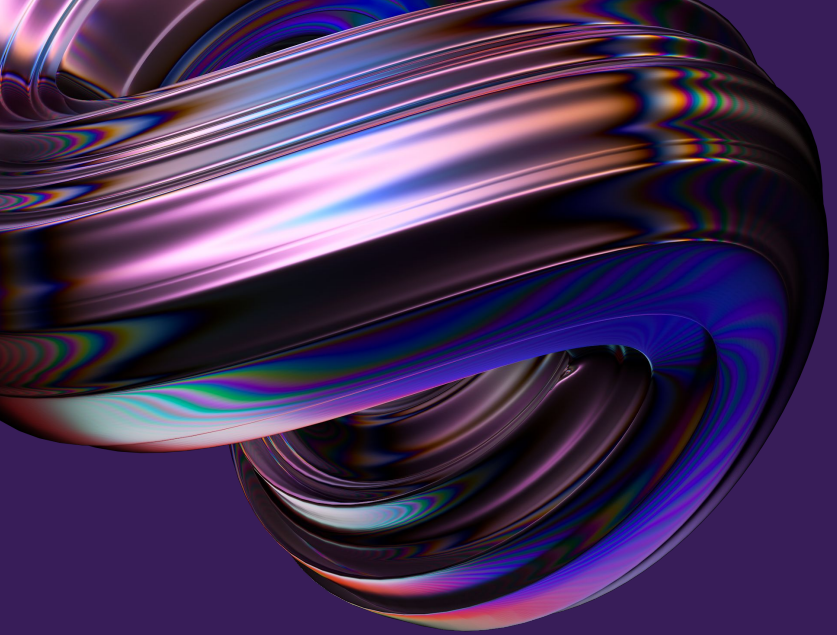
- **Prefect**
  - Pro:
    - Kubernetes
    - Einfach zu installieren und benutzen
    - Workflows werden in Python konfiguriert (Beispiel)
    - Saubere und gut dokumentierte API
    - Integriert mit einer Vielzahl an Services
      - AWS Tasks, Databricks Tasks, Dropbox Tasks, Github Tasks, KafkaTasks, Twitter Tasks, ...
  - Contra:
    - Open-core mit kostenpflichtigen Funktionen wie Multi-Tenancy
    - Bei einem `KubernetesRun` konfiguriert man die Ressourcen für den ganzen Run, nicht einzelne Tasks (Beispiel)

Vergleich ausgewählter Open Source Orchestration-Frameworks

- **Flyte**
  - Pro:
    - Fully Open Source
    - Kubernetes
    - Wird z.B. bei Lyft und Spotify at scale verwendet
    - Workflows werden in Python konfiguriert
    - Sauberer, dokumentierter Code
    - Saubere und (halbwegs) gut dokumentierte API
    - Integriert besser mit Kubeflow Operators als Kubeflow Pipelines
    - Integriert mit AWS Sagemaker
  - Contra:
    - Etwas komplizierter als Prefect

Flyte

```
@task(
    task_config=Spark(
        spark_conf={"spark.driver.memory": "1000M", ...}
)
def preprocess(partitions: int) -> str:
    # Do preprocessing on ephemeral spark cluster in k8s
    return dataset_uri

@task(
    task_config=PyTorch(
        num_workers=20,
        per_replica_requests=Resources(gpu="1")
    )
)
def training(uri: str) -> str:
    # Do distributed training on 20 nodes
    Return model_uri


@workflow
def train_pipeline() -> str:
    data_uri = preprocess()
    model_uri = training(uri=data_uri)
    return model_uri
```



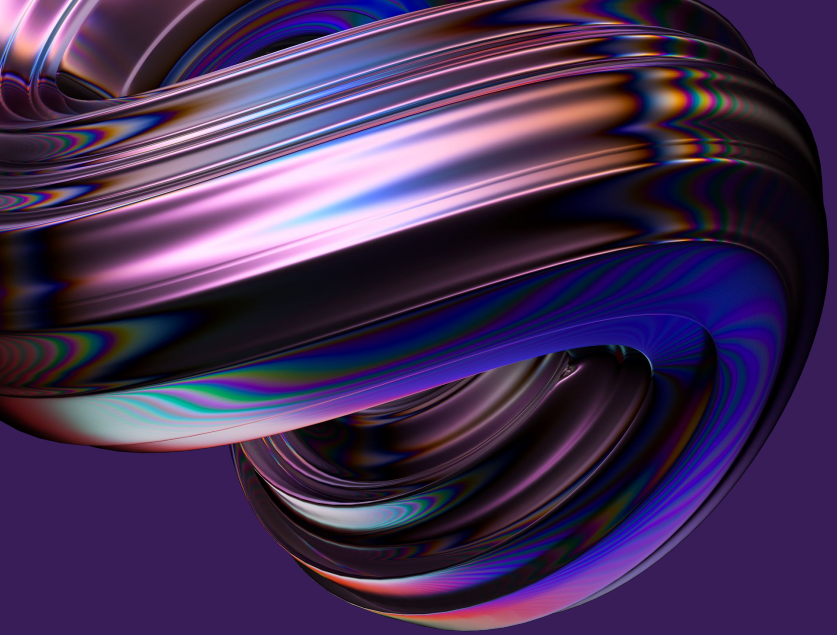
MERANTIX MOMENTUM

Hands-on Übung

Level 2 CICD Pipeline Automation mit Kubernetes, Argo und Flyte

```python
@task
def prepare_data() -> str:

@task
def train_model(data_uri: str) -> Tuple[str, str]:

@task
def deploy_model(run_id: str, model_uri: str) -> str:

@workflow
def pipeline() -> str:
    data_uri = prepare_data()
    run_id, artifact_uri = train_model(data_uri=data_uri)
    endpoint_uri = deploy_model(run_id=run_id, model_uri=artifact_uri)

    return endpoint_uri
```

- Beispiel ML-Pipeline
  - `flytesnacks/workflows/workflow_part1.py`
- Ausführen
  - `pip install flytekit`
  - `python flytesnacks/workflows/workflow_part1.py`
- Aufgaben
  - In der “Cloud” deployen und continously delivern
  - Mit Leben füllen



MERANTIX MOMENTUM

Kurzeinführung Kubernetes

Was ist Kubernetes?

- Aus der Dokumentation:
  - *“[...] extensible, open-source platform for managing containerized workloads and services [...]”*
  - *“If an application can run in a container, it should run great on Kubernetes.”*
  - *“[...] eliminates the need for orchestration.”; “[...] declarative configuration [...]”*
  - *“[...] provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application [...]”*
  - *“Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check [...]”*
  - *“You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.”*
  - *“Service discovery and load balancing”*

Kurzeinführung Kubernetes

- Ziel: Wie deployt man ein Docker-Image in Kubernetes?
- Beispiel
  - `docker run -p 8080:80 nginx:latest`
  - http://localhost:8080/
- Jetzt in Kubernetes …
  - `k3d cluster create tmp-cluster`

Was ist ein *Pod* in Kubernetes?

- Aus der Dokumentation:
  - *“Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.”*
  - *“A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.”*
  - *“In terms of Docker concepts, a Pod is similar to a group of Docker containers with shared namespaces and shared filesystem volumes.”*

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
      - containerPort: 80
```

```
docker run -p 8080:80 nginx:latest
```

Was ist ein *Pod* in Kubernetes?

- Manifest als pod.yaml speichern
- Pod erstellen
  - `kubectl apply -f pod.yaml`
- Beobachten, wie der Pod startet
  - `kubectl get pods`

- Logs anschauen
  - `kubectl logs nginx`

```
Every 2.0s: kubectl get pods

NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          24s
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
      - containerPort: 80
```

Was ist ein *Service* in Kubernetes?

- Aus der Dokumentation:
  - *"[...] if some set of Pods (call them "backends") provides functionality to other Pods (call them "frontends") inside your cluster, how do the frontends find out and keep track of which IP address to connect to, so that the frontend can use the backend part of the workload?*

    Enter Services."
  - *"In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service). The set of Pods targeted by a Service is usually determined by a selector."*
  - *"Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods, and can load-balance across them."*

```yaml
apiVersion: v1
kind: Service
metadata:
  namespace: default
  name: nginx-service
  labels:
    app: nginx
spec:
  selector:
    app: nginx

  ports:
  - port: 80
    targetPort: 80
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest

    ports:
      - containerPort: 80
```

Was ist ein *Service* in Kubernetes?

- Manifest als service.yaml speichern
- Service erstellen
  - `kubectl apply -f service.yaml`
- Wo sieht man den service?
  - `kubectl get services`

```
Every 2.0s: kubectl get services

NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes      ClusterIP   10.43.0.1      <none>        443/TCP   28m
nginx-service   ClusterIP   10.43.46.102   <none>        80/TCP    40s
```

- Wie kann man den service verwenden, um auf den micro-service zuzugreifen?
  - `kubectl port-forward service/nginx-service 8081:80`
  - http://localhost:8081/
  - Innerhalb des Clusters (im gleichen namespace) ist der Pod unter http://nginx-service:80 erreichbar

```yaml
apiVersion: v1
kind: Service
metadata:
  namespace: default
  name: nginx-service
  labels:
    app: nginx
spec:
  selector:
    app: nginx

  ports:
  - port: 80
    targetPort: 80
```

Was ist ein *Deployment* in Kubernetes?

- Aus der Dokumentation:
  - *"Create a Deployment to rollout a ReplicaSet. The ReplicaSet creates Pods in the background."*
  - *"A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods."*
  - *"Scale up the Deployment to facilitate more load."*

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: default
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
      - containerPort: 80
```

Was ist ein *Deployment* in Kubernetes?

- Manifest als deployment.yaml speichern
- Deployment erstellen
  - `kubectl apply -f deployment.yaml`

```
Every 2.0s: kubectl get deployments

NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   5/5     5            5           10m
```

```
Every 2.0s: kubectl get pods

NAME                     READY   STATUS    RESTARTS   AGE
nginx-585449566-lzbj7    1/1     Running   0          10m
nginx-585449566-vbpq8    1/1     Running   0          10m
nginx-585449566-fp8qs    1/1     Running   0          76s
nginx-585449566-w6cxg    1/1     Running   0          76s
nginx-585449566-8vvfq    1/1     Running   0          76s
```

- Aufräumen
  - `k3d cluster delete tmp-cluster`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: default
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

Welche Objekte gibt es noch in Kubernetes?

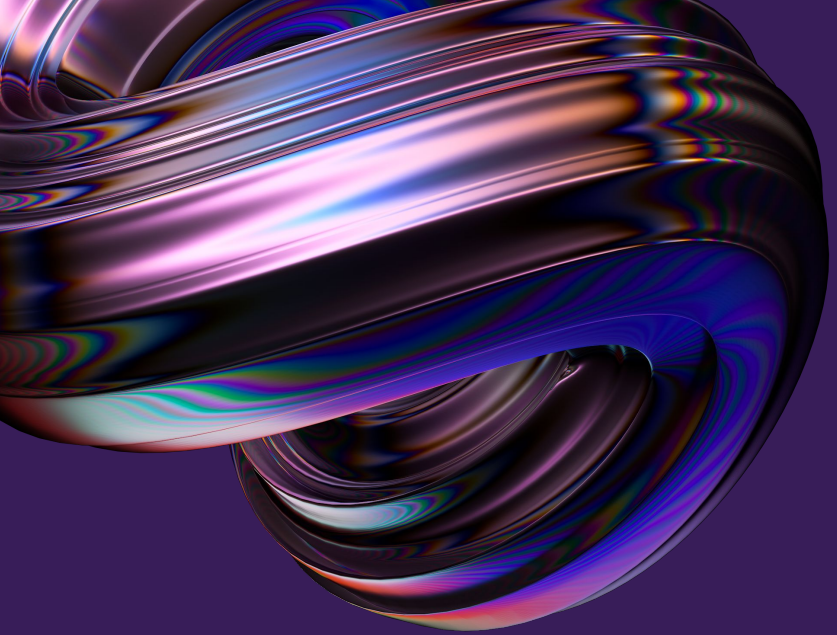
- Jobs
- StatefulSets
- Secrets
- ServiceAccounts
- Roles
- Rolebindings
- Namespaces
- ConfigMaps
- Volumes, PersistentVolumes, PersistentVolumeClaims
- ...
- **CustomResourceDefinition**
  - *“A custom resource is an extension of the Kubernetes API that is not necessarily available in a default Kubernetes installation. It represents a customization of a particular Kubernetes installation. However, many core Kubernetes functions are now built using custom resources, making Kubernetes more modular.”* (Quelle)

Kurzeinführung Kubernetes - Zusammenfassung

- Um ein Docker Image in einem Kubernetes Cluster als Container starten zu lassen verwendet man z.B. einen *Pod* oder ein *Deployment*
- Um mit anderen Micro-Services zu kommunizieren (DNS discovery) verwendet man *Services*

Warum Machine Learning auf Kubernetes?

- Rapide wachsendes Ökosystem an Open Source (ML-) Tools da …
  - … Kubernetes in vielen Organisationen benutzt wird, um tausende Microservices zu deployen. ML-Modelle, die von diesen Microservices benutzt werden, müssen idealerweise ebenfalls in Kubernetes laufen.
  - … K8s das Training, Operationalisieren und Monitoring von tausenden ML-Modellen erlaubt, ohne, dass sich Data-Scientists um Infrastruktur kümmern müssen.
- Warum ist Kubernetes auch für kleinere Teams/Organisationen interessant?
  - Ökosystem an Open Source Tools für die Kubernetes als *“Verbindungsmechanismus”* dient.
  - Deep-Learning Workflows brauchen viele Ressourcen. K8s kann diese temporär bereitstellen, die Anzahl der Nodes dynamisch hochskalieren und die Workflows orchestrieren
    - Spark, Dask und Distributed Training (MPI, Gloo, NCCL, …) innerhalb von Kubernetes



MERANTIX MOMENTUM

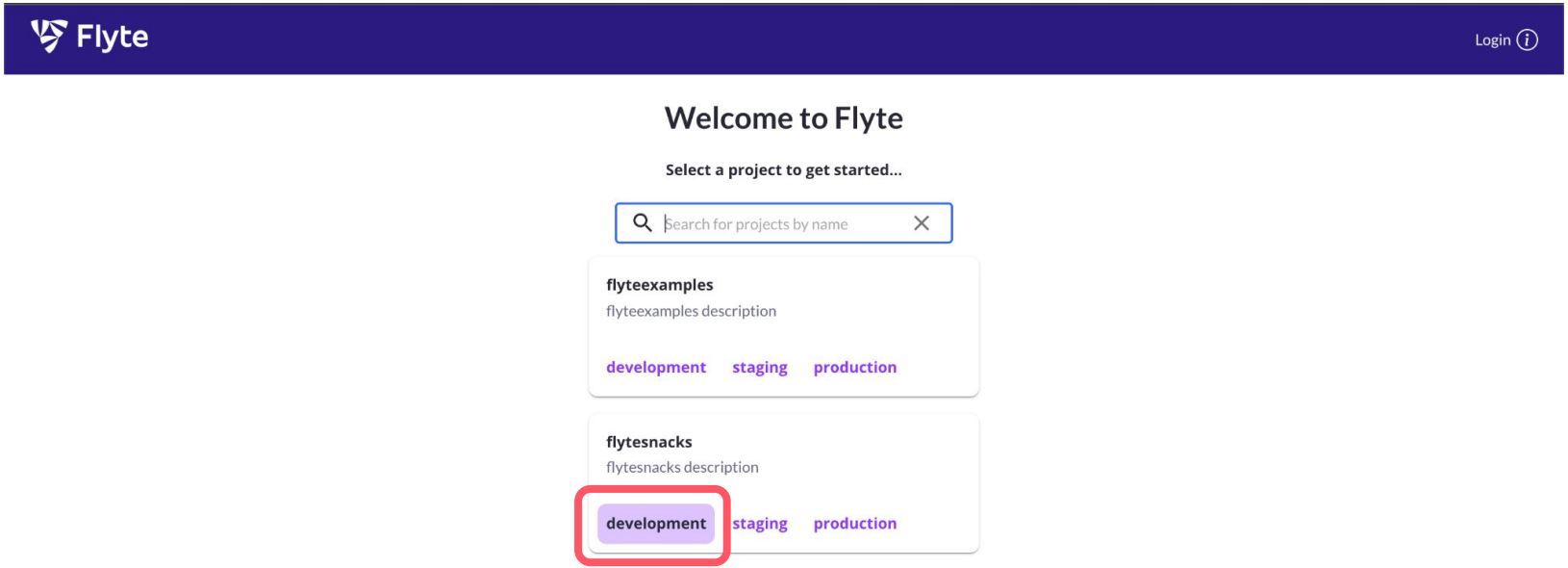
Sandbox

Konfiguration unserer MLOps Sandbox

- Ladet den Sandbox-Code herunter
- Erstellt eine lokale container registry
  - `k3d registry create registry.localhost --port 5000`
- Und ein Volume
  - `mkdir -p /tmp/k3dvol`
- Erstellt einen lokalen kubernetes cluster mit Hilfe von k3d
  - `k3d cluster create --volume /tmp/k3dvol:/tmp/k3dvol -p "30081:30081@server:0:direct" -p "30084:30084@server:0:direct" -p "6000:6000@server:0:direct" -p "2746:2746@server:0:direct" --no-lb --k3s-arg '--no-deploy=traefik' --k3s-arg '--no-deploy=servicelb' --registry-use k3d-registry.localhost:5000 sandbox`
- Testet, dass der lokale sandbox cluster funktioniert
  - `kubectl get namespace`

- `jupyter-notebook Workshop_Befehle_Zusammenfassung.ipynb`

Konfiguration unserer MLOps Sandbox



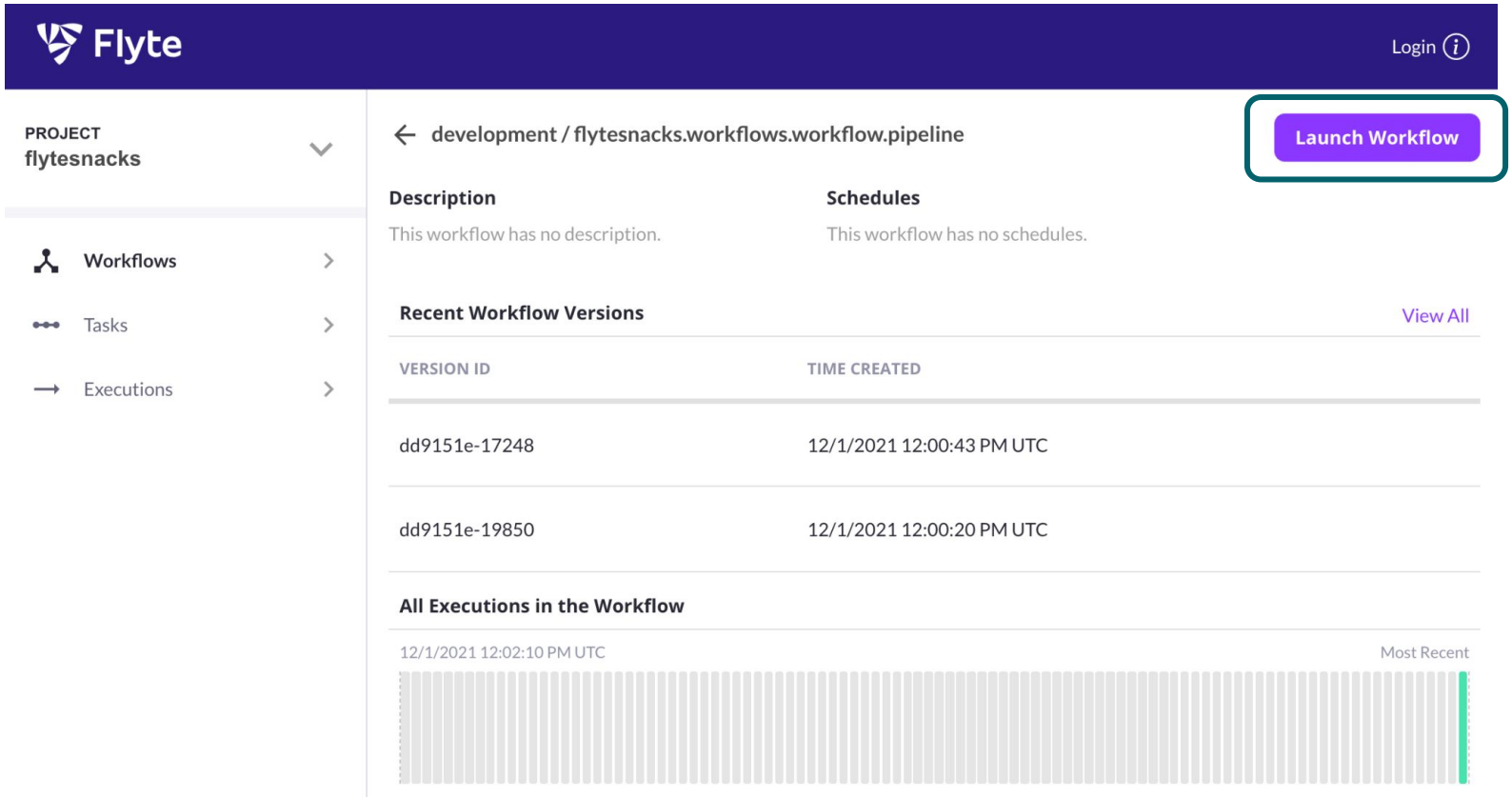
Pipeline packen, registrieren und ausführen

- Docker Image für den Fylte Workflow bauen und pushen

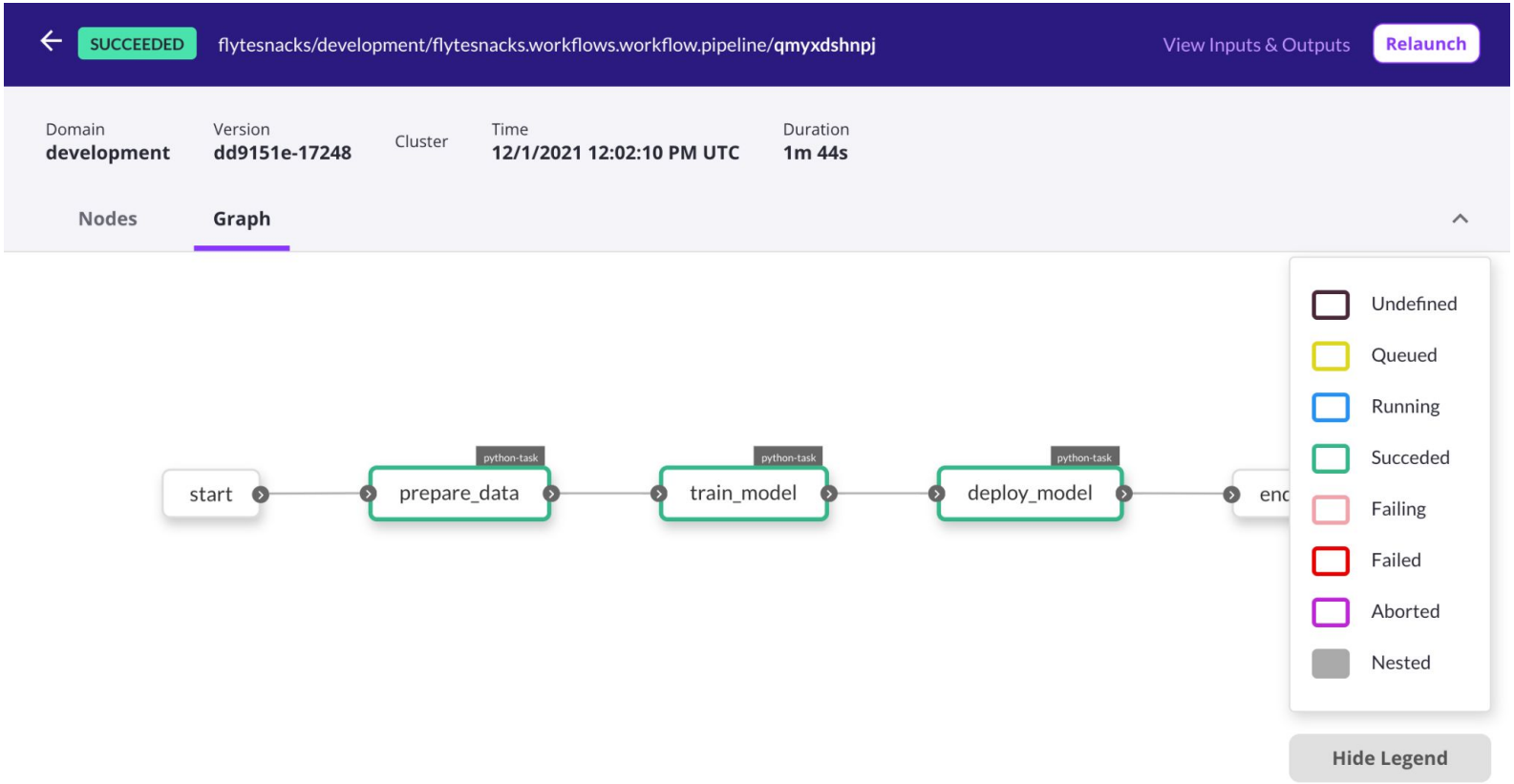
```
docker build -t localhost:5000/workflow:latest .
docker push localhost:5000/workflow:latest
```

- Workflow manuell registrieren
  - ```
    pyflyte --config flyte_config register --project flytesnacks\
            --image k3d-registry.localhost:5000/workflow:latest\
            --version 1 flytesnacks/workflows/workflow_part1.py
    ```
- Workflow auf http://localhost:30081/console/projects/flytesnacks/workflows starten
    ```

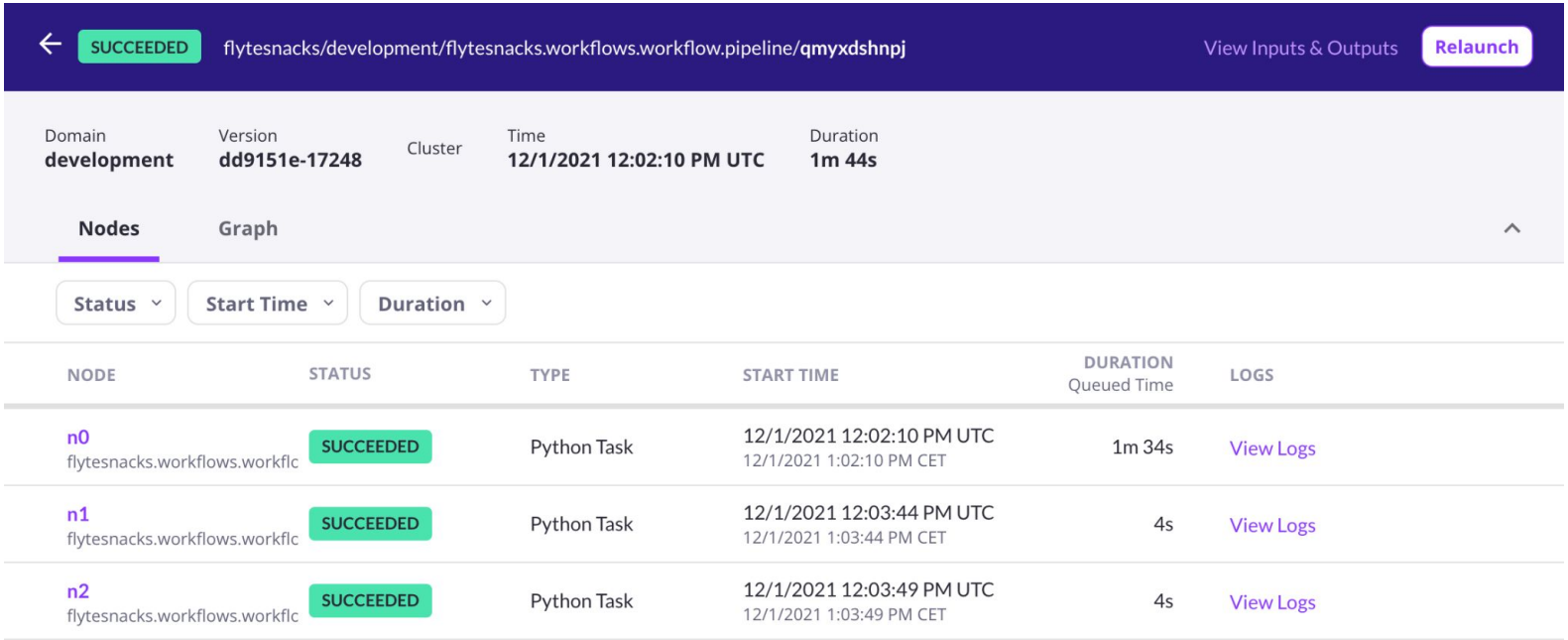
Pipeline packen, registrieren und ausführen



Pipeline packen, registrieren und ausführen



Pipeline packen, registrieren und ausführen



Pipeline packen, registrieren und ausführen

- Wo werden die Tasks ausgeführt?
  - In Pods im Namespace flytesnacks-development (project-domain)

```
kubectl get pods -n flytesnacks-development
NAME                READY   STATUS      RESTARTS   AGE
qmyxdshnpj-n0-0     0/1     Completed   0          11m
qmyxdshnpj-n1-0     0/1     Completed   0          9m32s
qmyxdshnpj-n2-0     0/1     Completed   0          9m27s
```

- Wo sind die Logs der Tasks?

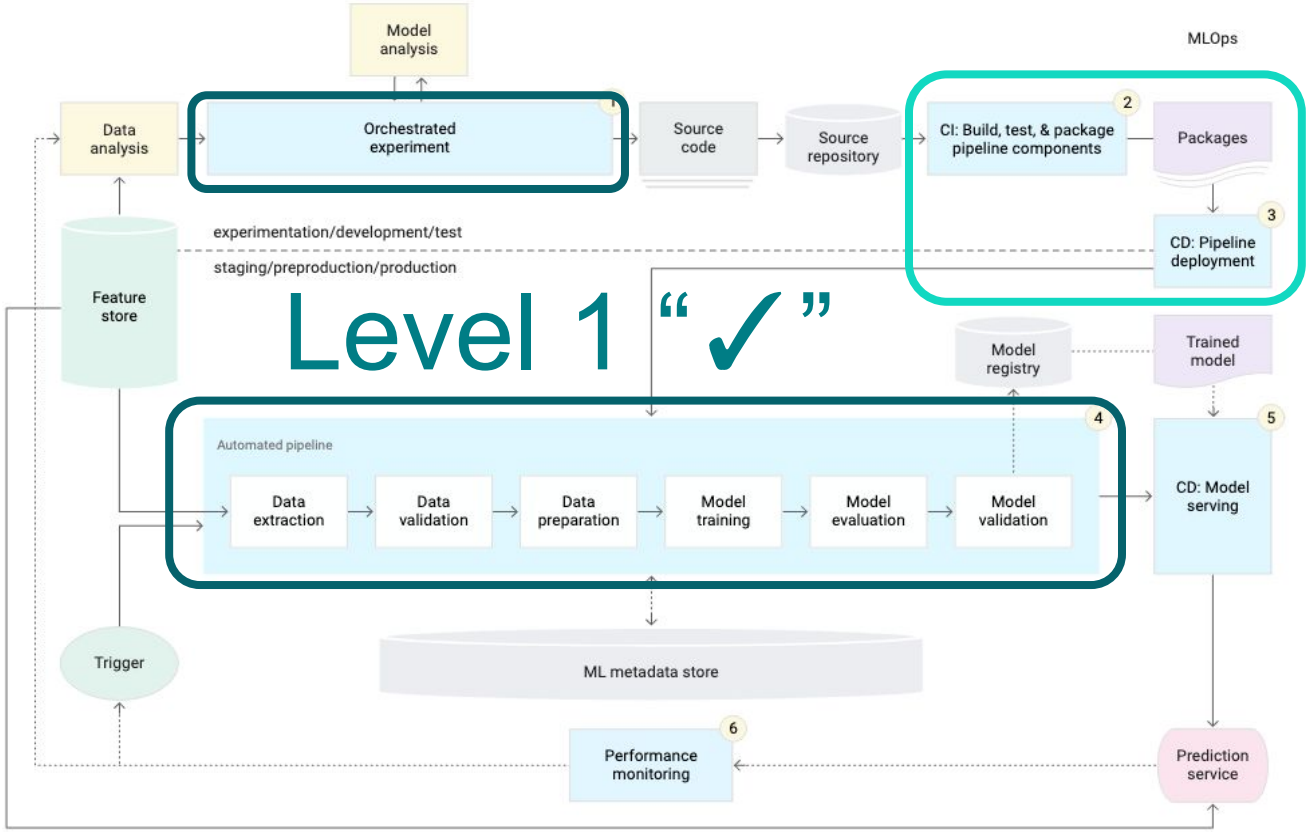
```
kubectl -n flytesnacks-development logs qmyxdshnpj-n2-0
```

- Wie werden Task-Ressourcen konfiguriert?

```
@task(requests=Resources(gpu="1"))
def my_deep_learning_task():

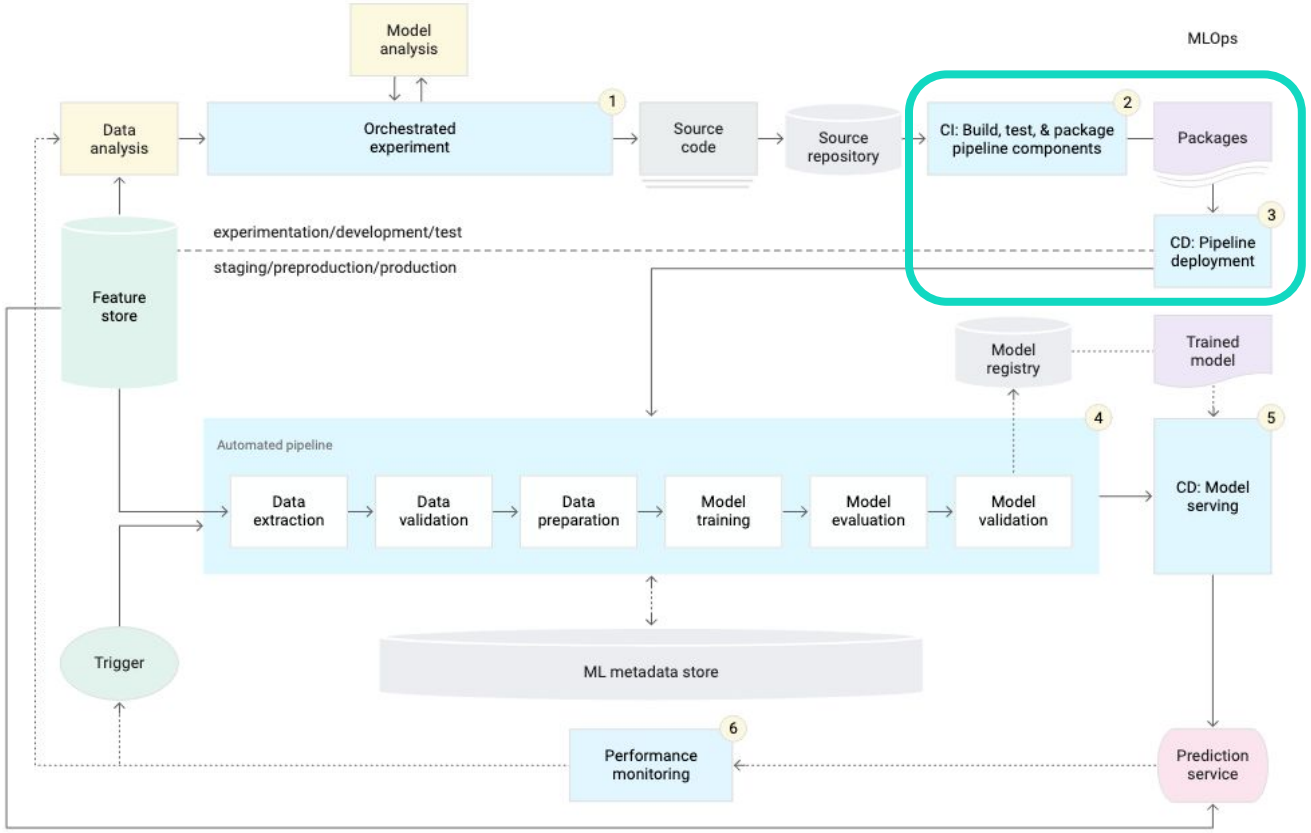
@task(
    task_config=PyTorch(
        num_workers=20, per_replica_requests=Resources(gpu="1"), ...
```

Und jetzt?



Level 2?

CICD Tools



Jenkins

argo

RUNNER

circleci

Travis CI

Google Cloud Build

...

MLOps level 2: CI/CD pipeline automation

- **CI [skip]**
  - Unit Tests Feature-Engineering, ML-Modell
  - Unit Test, dass Tasks die erwarteten Artefakte produzieren
  - Integrationstest Tasks und Workflow

- **CD**
  - Docker Image bauen und Workflow packen und registrieren
    - Vorher
      - `docker build …, docker push …`
      - `pyflyte register …`
      - `Manuelles starten des Workflow …`
    - Im Argo Workflow
      - `Lokal*`
        - `docker build …, docker push …`
      - `Im Cluster`
        - `Github Repository klonen`
        - `Das Workflow File packen (pyflyte package)`
        - `Das Workflow file in Flyte registrieren (flytectl register files)`
        - `erstellen des Launch Plans (flytectl get launchplan)`
        - `ausführen des Launch Plans (flytectl create execution)`

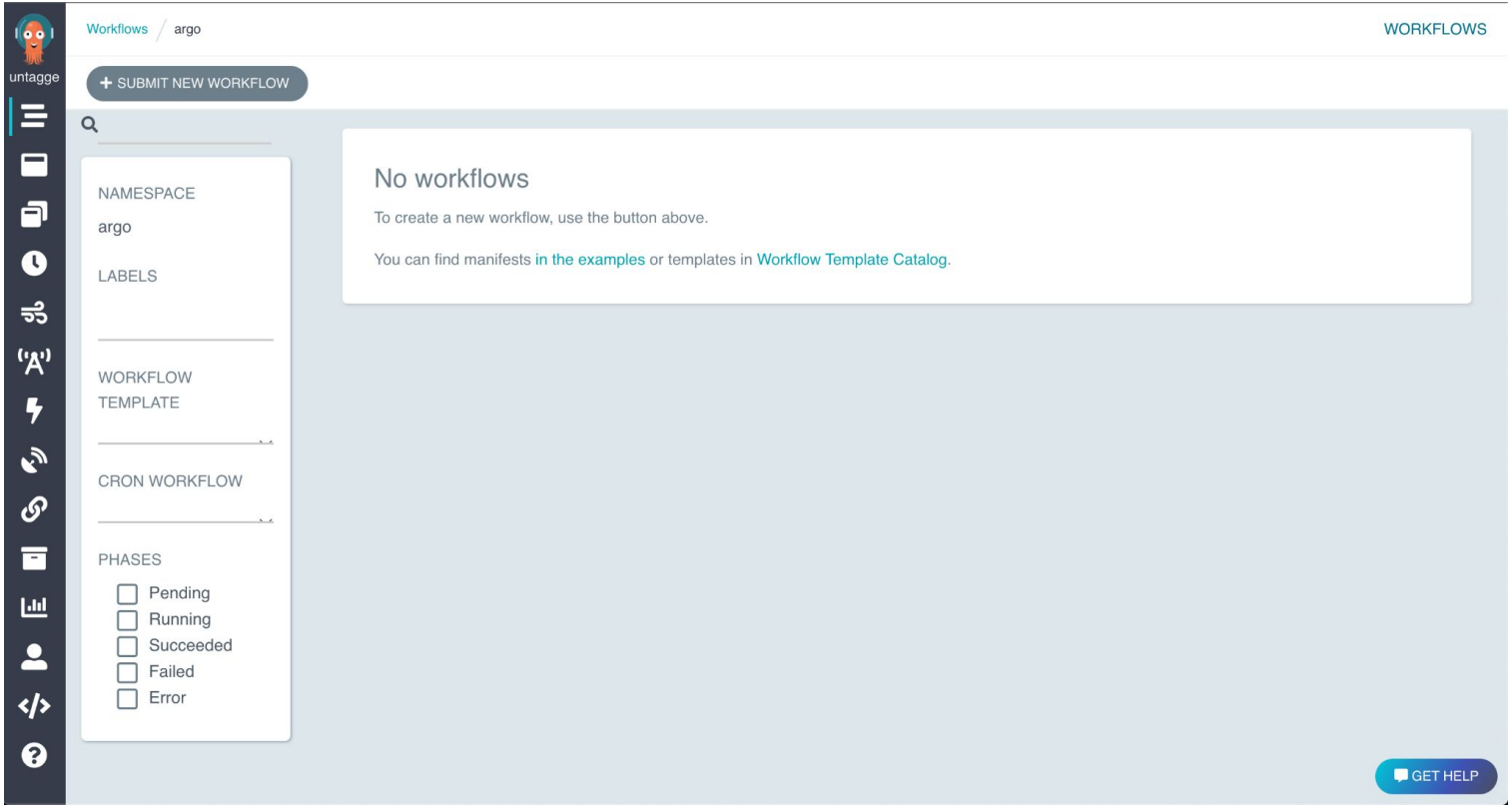
`*Das bauen und pushen des Docker Images würde in production von einer CICD Pipeline übernommen werden, welche anschließend den Workflow startet.`

MLOps level 2: CI/CD pipeline automation

- Github Repository erstellen
  - `git init`
  - `git add .`
  - `git commit -m "First commit"`
  - `git push # Der Code muss auf dem HEAD der remote Git Repository sein`
- Github Token erstellen
  - https://github.com/settings/tokens/new
  - Gebt dem Token alle Permission bei “Repo”, erstellt und kopiert den Token
  - export GITHUB_USER=<YOUR_GITHUB_USER>
  - export GITHUB_TOKEN=<YOUR_GITHUB_TOKEN>
  - export GITHUB_REPO=<YOUR_GITHUB_REPO>
- Argo installieren
  - `kubectl create ns argo`
  - `kubectl apply -n argo -f https://raw.githubusercontent.com/argoproj/argo-workflows/master/manifests/quick-start-postgres.yaml`
  - `cat argo_cred.yaml | envsubst | kubectl apply -f -`
  - In separatem Terminal einen Proxy öffnen
    - `kubectl -n argo port-forward svc/argo-server 2746:2746`

Argo UI auf https://localhost:2746/ aufrufen und Privacy Warnung umgehen

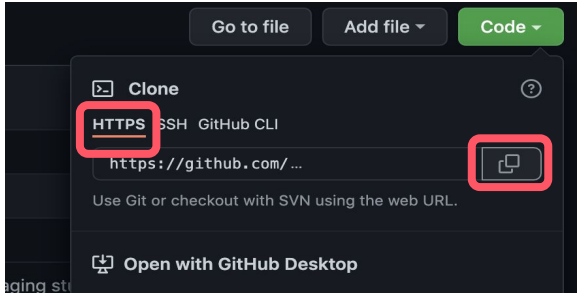
MLOps level 2: CI/CD pipeline automation



MLOps level 2: CI/CD pipeline automation

argo_cicd_workflow.yaml

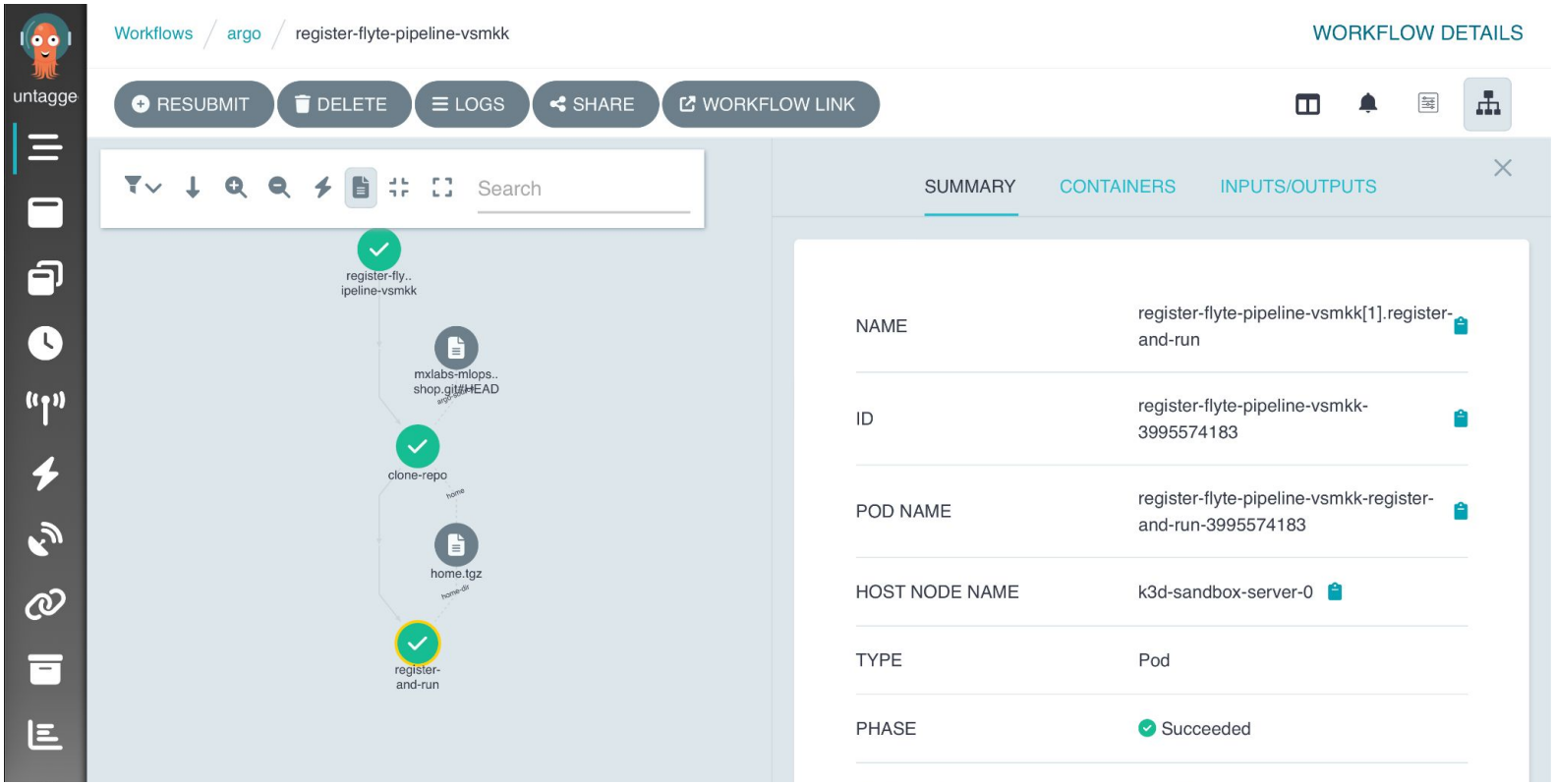
```
- name: git-clone
    inputs:
      artifacts:
      - name: argo-source
        path: /src
        git:
          repo:  $GITHUB_REPO <- https://github.com/.../...git
          revision: HEAD
    script: |
      pyflyte --pkgs flytesnacks.workflows. $WORKFLOW package
```



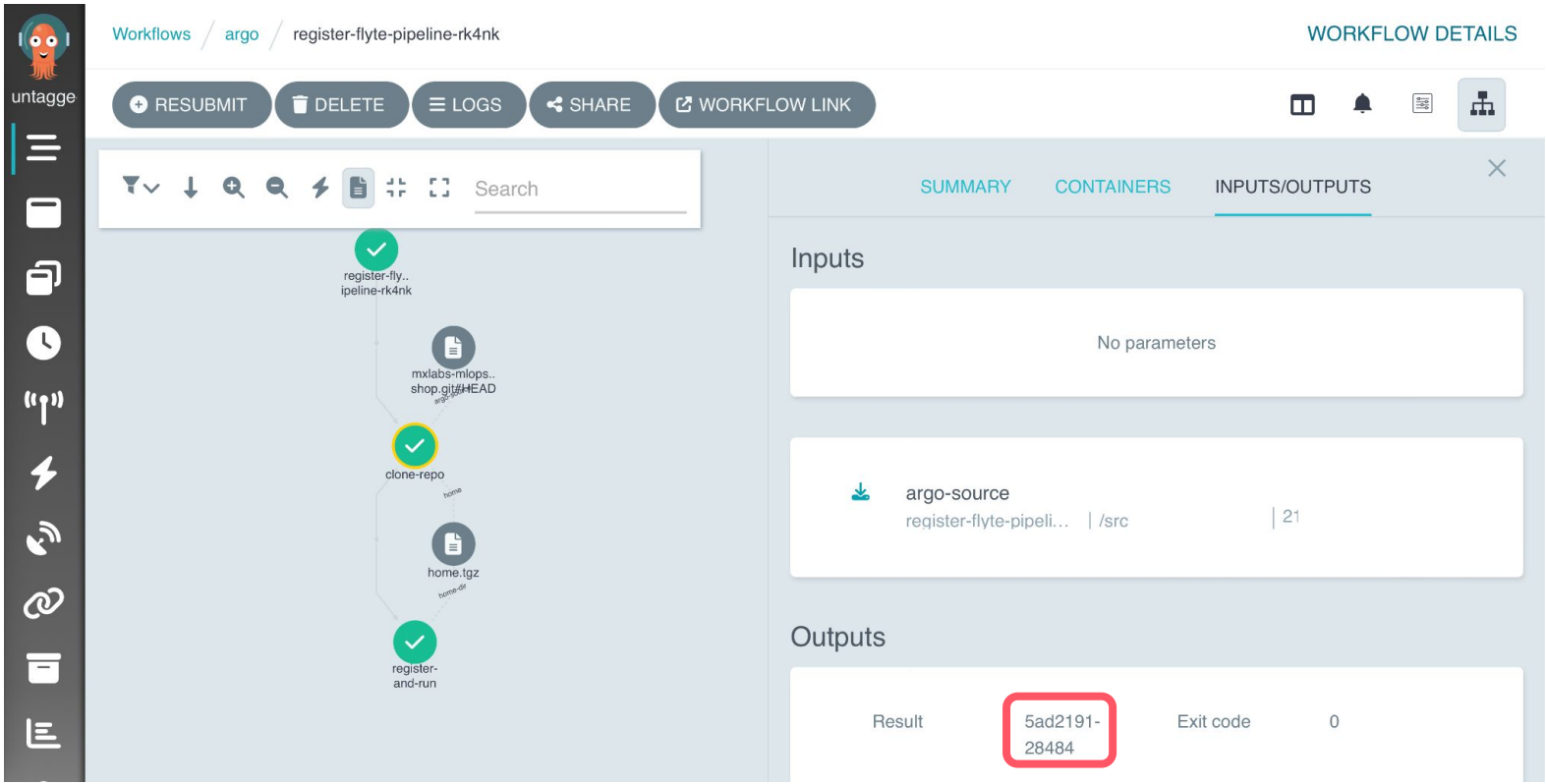
- Stellt Euch vor, ein Web-Hook startet einen CICD Workflow bei einem Commit
- `export WORKFLOW=workflow_part1`
- `cat argo_cicd_workflow.yaml | envsubst | kubectl -n argo create -f -`

$WORKFLOW and $GITHUB_REPO are substituted by envsubst

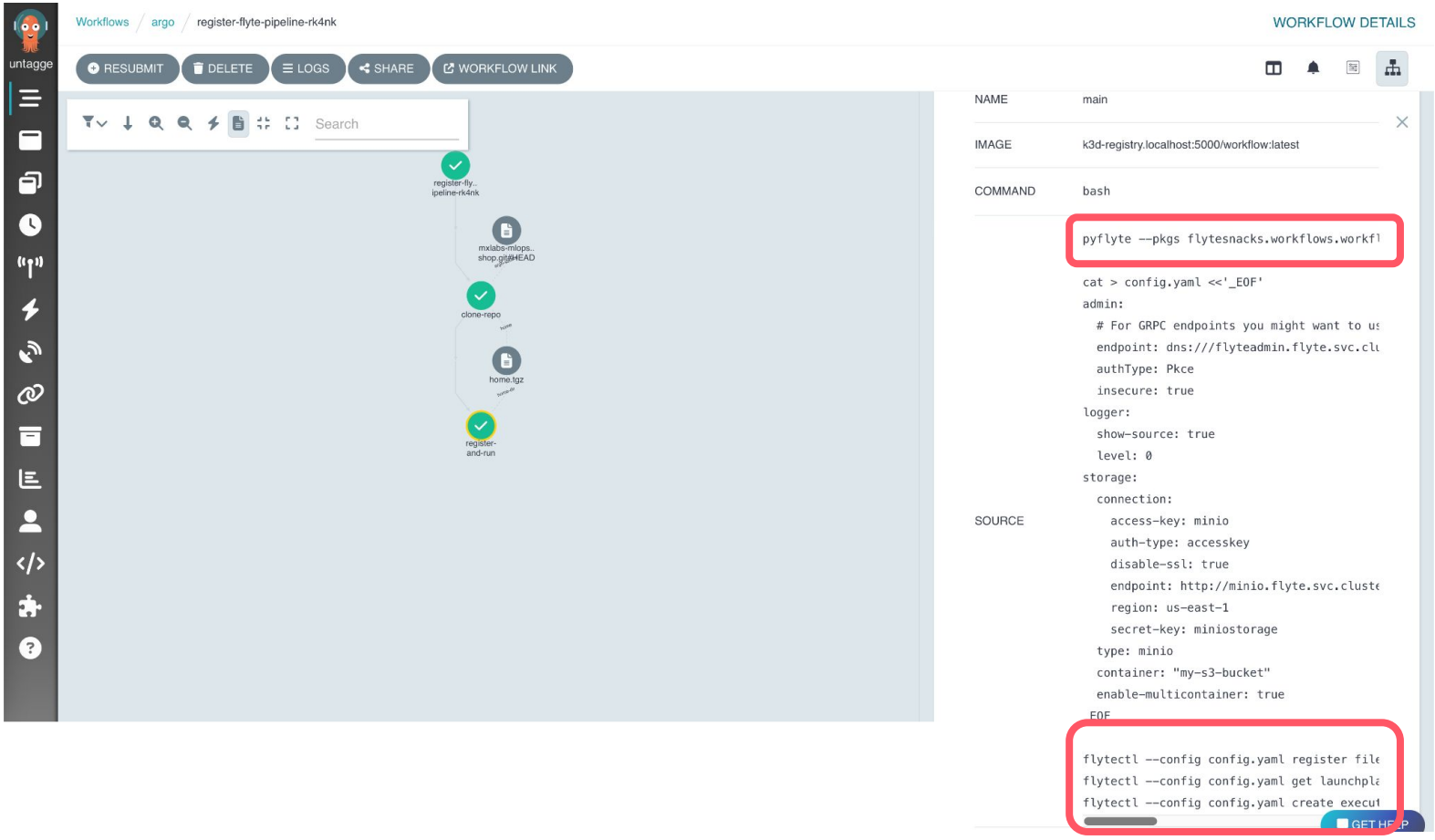
MLOps level 2: CI/CD pipeline automation



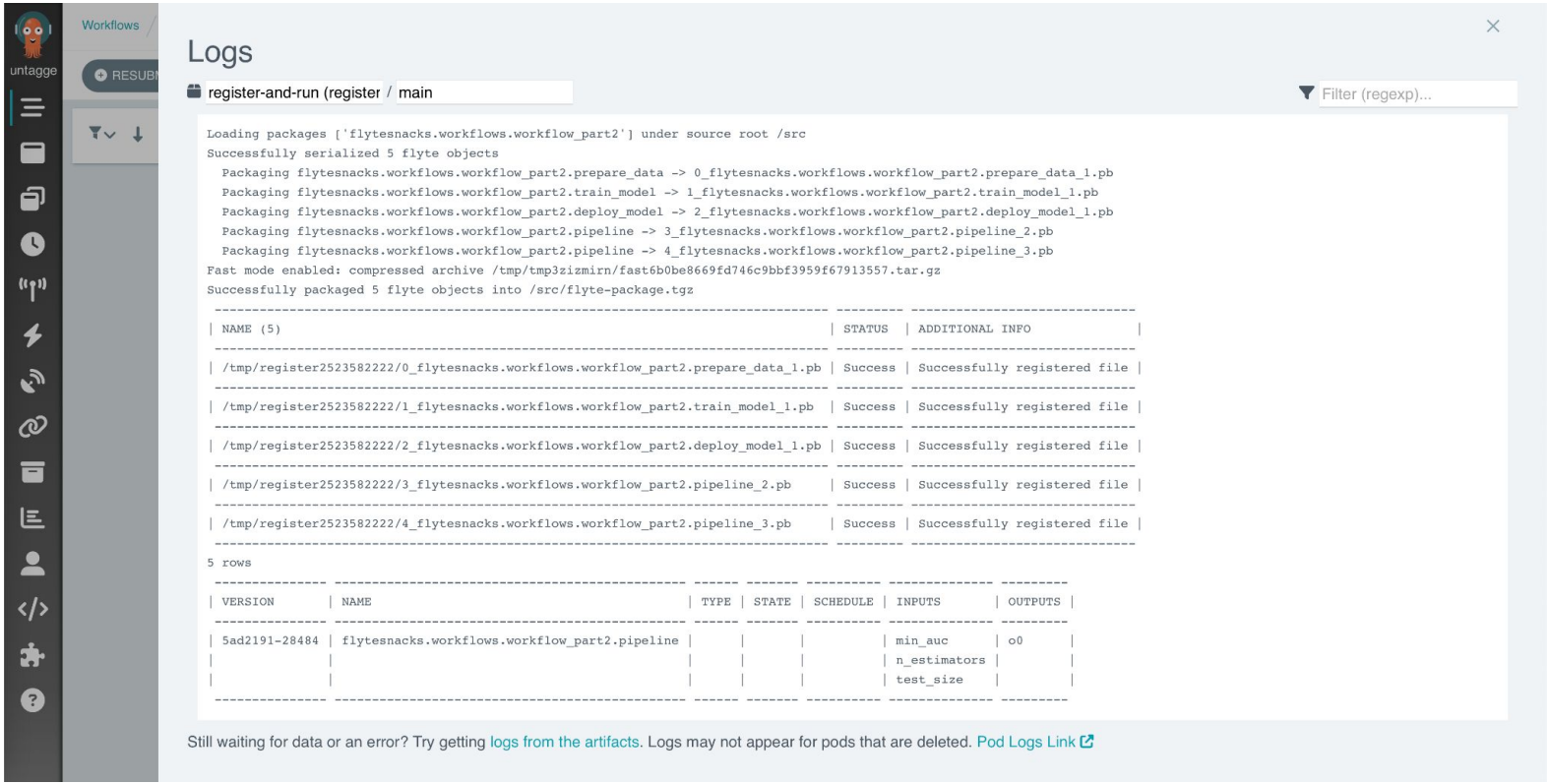
MLOps level 2: CI/CD pipeline automation



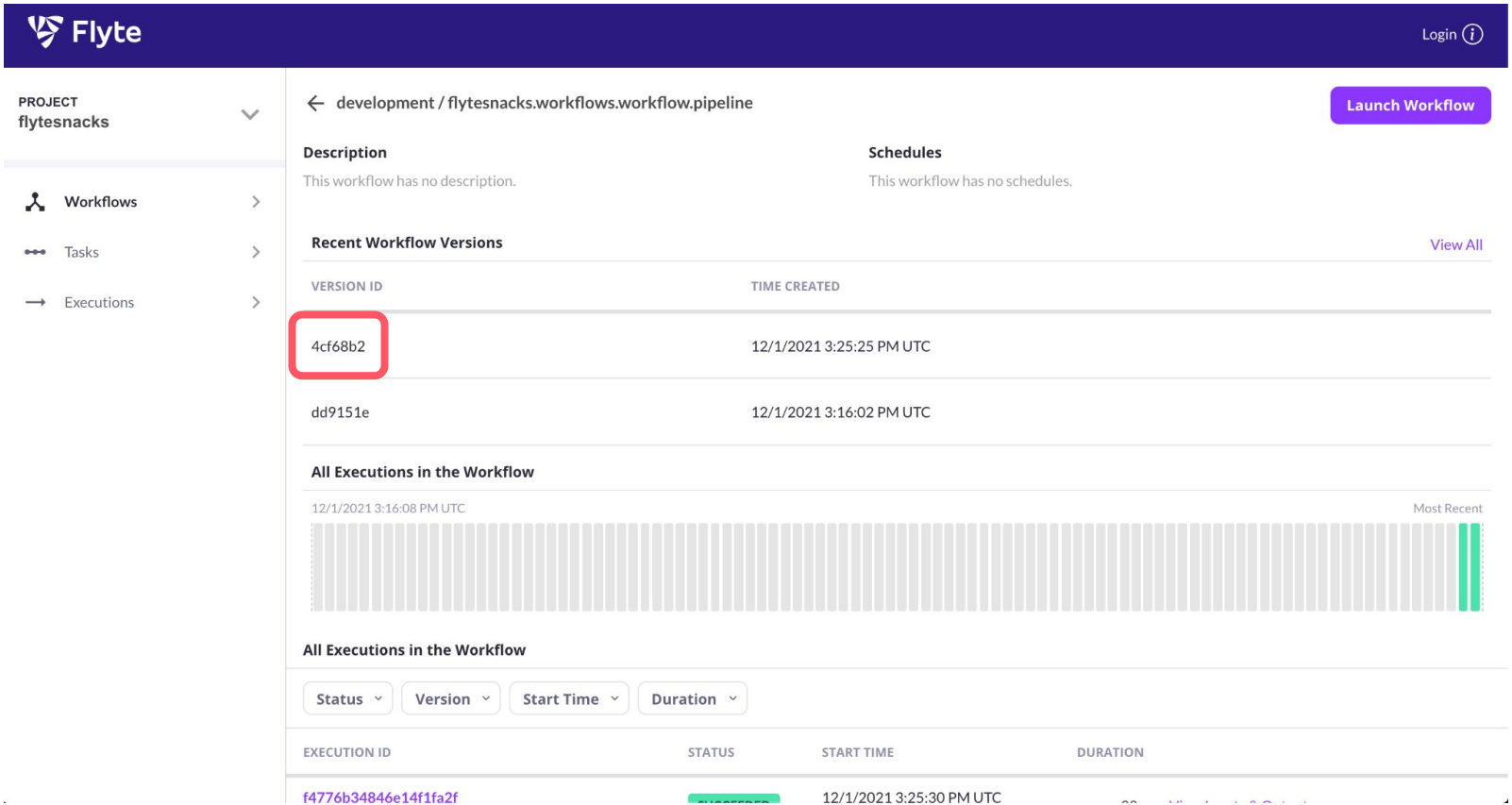
MLOps level 2: CI/CD pipeline automation



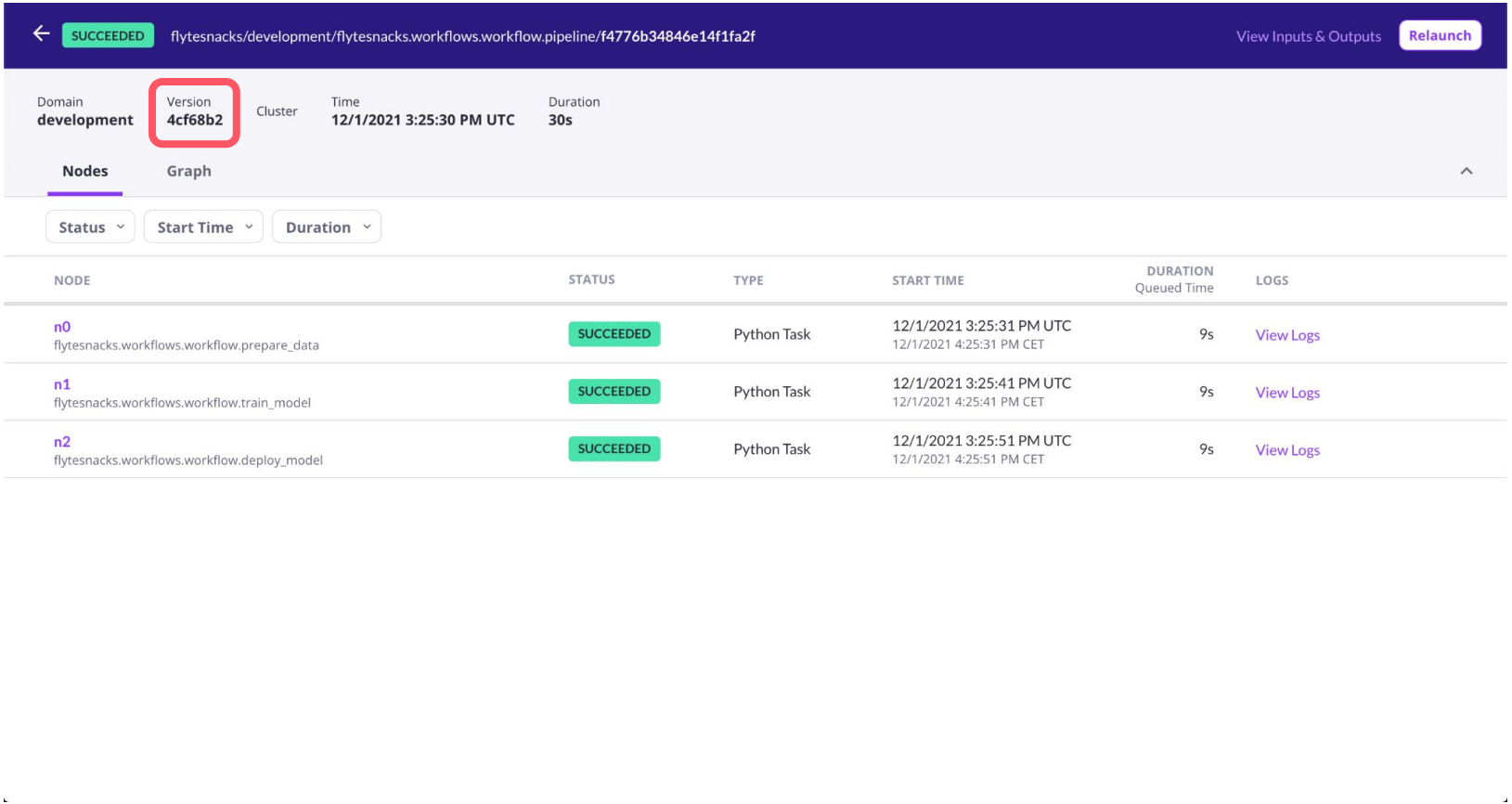
MLOps level 2: CI/CD pipeline automation



MLOps level 2: CI/CD pipeline automation



MLOps level 2: CI/CD pipeline automation

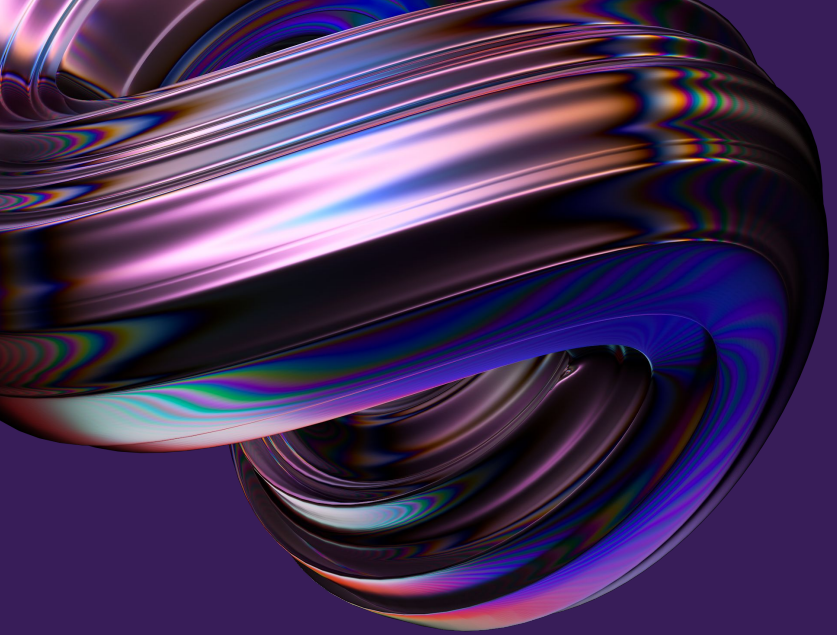


MLOps level 2: CI/CD pipeline automation

- Was haben wir jetzt eigentlich gemacht?
  - Zwei Pipelines
    - ML-Pipeline in Flyte
      - Orchestriert Data Preprocessing, Training, Evaluierung, Deployment
    - CICD Pipeline in Argo
      - Testet die Komponenten der Pipeline (Unit- und Integrations-Tests) [Skipped]
      - Klont Github repository
      - Packt und registriert Pipeline Version bei Flyte
      - Startet Workflow

MLOps level 2: CI/CD pipeline automation

- Überlegungen
  - Flyte stellt unterschiedliche “Domains” bereit
    - Development
    - Staging
    - Production
  - Wir haben nur *Development* verwendet
  - In der Regel habt Ihr unterschiedliche CICD Pipelines für z.B. Feature-Branches, den Main Branch und Tags.
  - Es empfiehlt sich diese mit den Domains von Flyte zu verknüpfen also z.B.:
    - Feature-Branch → Development
    - Main Branch → Staging
    - Tag → Production



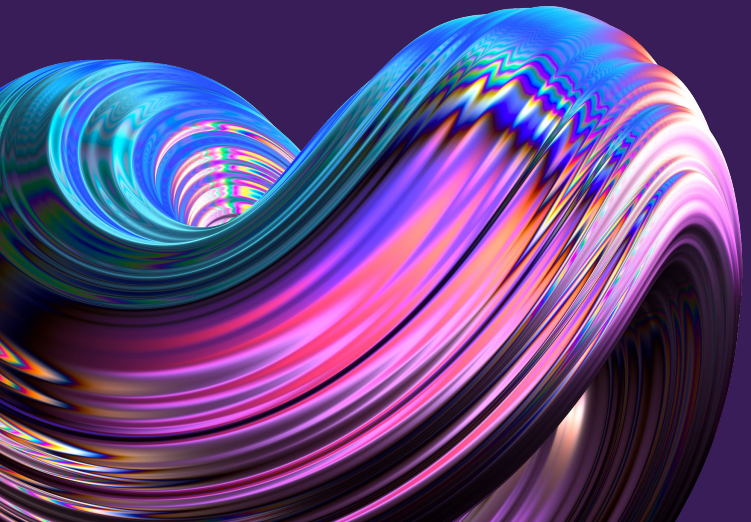
MERANTIX MOMENTUM

Zusammenfassung Session 1

Zusammenfassung Session 1

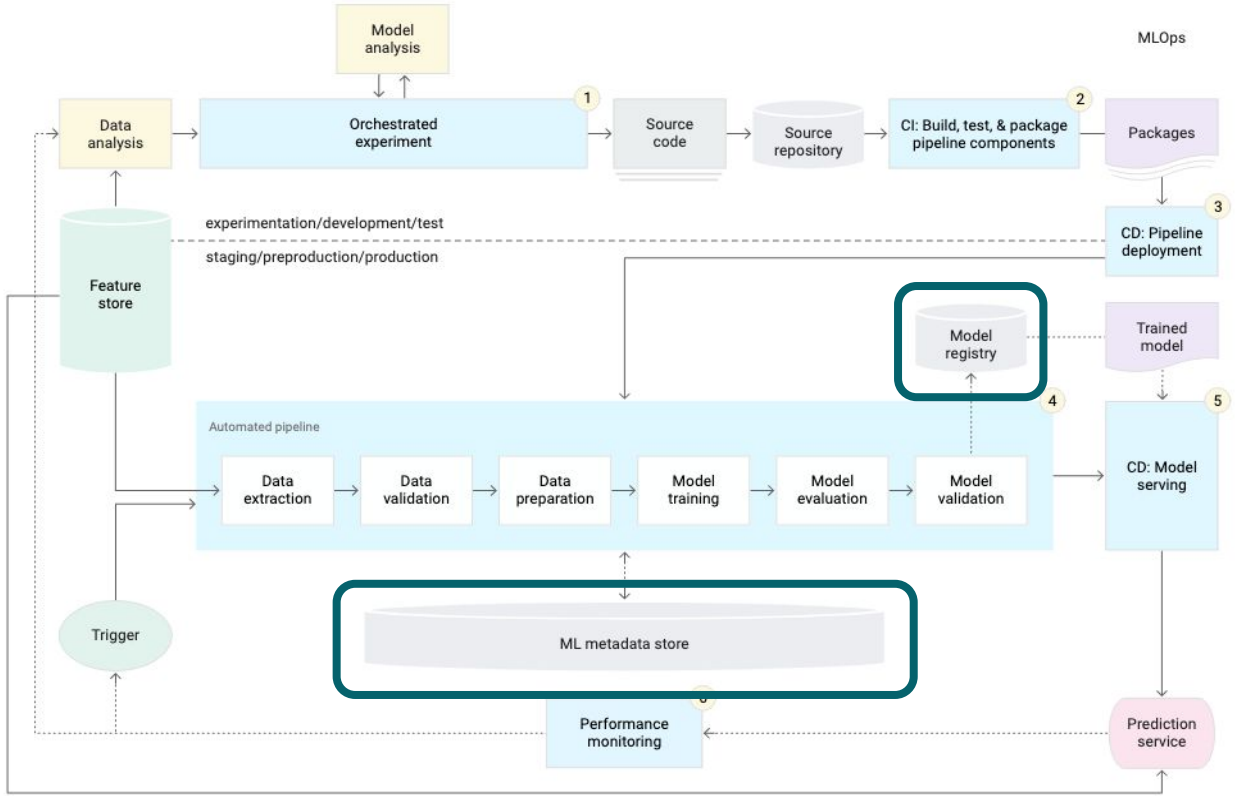
- Was ist MLOps und CD4ML?
  - Maturitätsgrade in Machine Learning Systemen
    - Level 0: Manuell
    - Level 1: ML-Pipeline
    - Level 2: CICD ML-Pipeline
- Vergleich ausgewählter Open Source (ML) Orchestration-Frameworks
- Hands-on Übung: Level 2 CICD Pipeline Automation mit Kubernetes, Argo und Flyte
  - Kurzeinführung Kubernetes
    - Was ist Kubernetes? Was sind Pods, Deployments und Services?
  - Einrichtung der Sandbox
  - Beispiel Level 1 Pipeline
  - Beispiel Level 2 Pipeline mit Continuous Delivery

MERANTIX MOMENTUM



DB Systel MLOps Schulung Session 2

Session 2 - *ML-Metadatastore und Model-Registry*



Session 2 - *ML-Metadatastore und Model-Registry*

- Was ist ein ML-Metadatastore? Was ist eine Model-Registry?
- Übersicht ausgewählter Tools und Frameworks
- Hands-on Übung:
    - Training und Experiment-Tracking mit Mlflow
        - Deployment des Mlflow-Tracking-Severs in Kubernetes
        - Integration in unsere ML-Pipeline in Flyte

Was ist ein ML-Metadatastore? Was ist eine Model-Registry?

- ML-Metadatastore
  - API und UI zum Loggen und Visualisieren von Experimenten
    - Pipeline-Version
    - Datum und Dauer eines Experiments
    - Parameter
    - Metriken
    - Artefakte

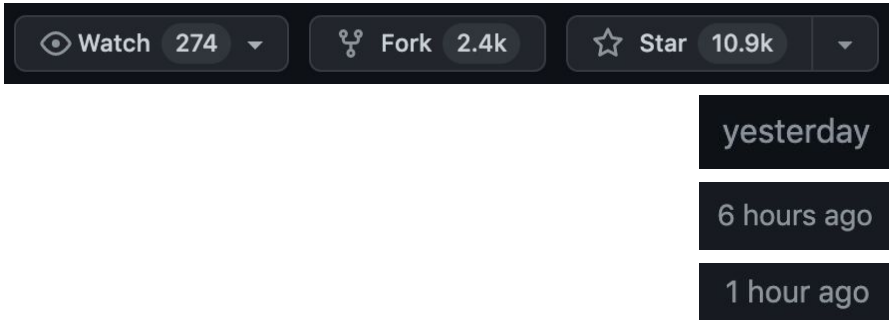
- Model-Registry
  - API und UI zum Managen des Modell-Lifecycles
    - Modelle registrieren
    - Modelle versionieren (z.B. zwecks Verknüpfung zu vorherigen Modell-Versionen für Rollbacks)
    - Lineage (Welches Modell wurde in welchem Experiment trainiert)
    - Statusinformationen (Ist das Modell aktuell in Produktion?)

Übersicht ausgewählter Tools und Frameworks

- Open-Source
  - Mlflow
  - Kubeflow Metadata
  - ML Metadata (Tensorflow Extended)
  - Aim
  - ...
- “Closed-Source”
  - Neptune AI
  - Weights & Biases
  - ...
- Hyper-Scaler
  - AWS Sagemaker Experiments
  - Vertex AI metadata
  - ...

Übersicht ausgewählter Tools und Frameworks - Open-Source

- Mlflow
  - Pro:
    - pip-Install
    - Von Databricks entwickelt
      - Als managed Service verfügbar
    - Platzhirsch/große Community
    - Einfache Python API
    - Sehr ausführliche Dokumentation
    - Viele Features:
      - Mlflow Projects
      - Mlflow Models
      - Mlflow Tracking
      - Mlflow Model Registry
  - Contra:
    - UI zu langsam für Plots mit vielen Datenpunkten
    - Contributions schwierig



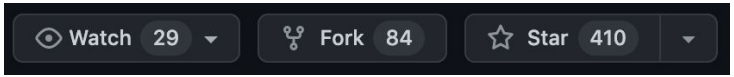

```
import mlflow

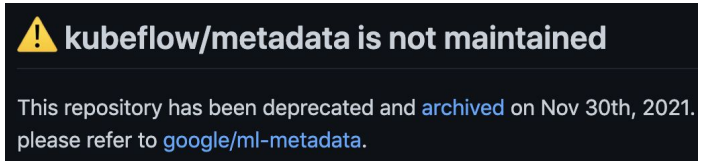
mlflow.set_tracking_uri(“http://…”)

with mlflow.start_run():
    mlflow.log_param("x", 1)
    mlflow.log_metric("y", 2)
    ...
```

Übersicht ausgewählter Tools und Frameworks - Open-Source

- Kubeflow Metadata
  - Contra:
    - Letzter Commit vor 13 Monaten
- ML Metadata (Tensorflow Extended)
  - Pro:
    - pip-Install
    - Wird aktiv maintained
  - Contra:
    - Wenig Dokumentation (besser seit Kohorte 1)
    - Unnötig komplizierte API
    - Kleinere Community als Mlflow






```
from ml_metadata.metadata_store import metadata_store
from ml_metadata.proto import metadata_store_pb2

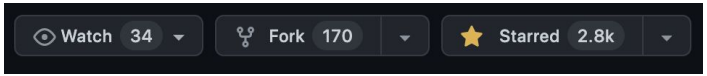
store = metadata_store.MetadataStore(...)

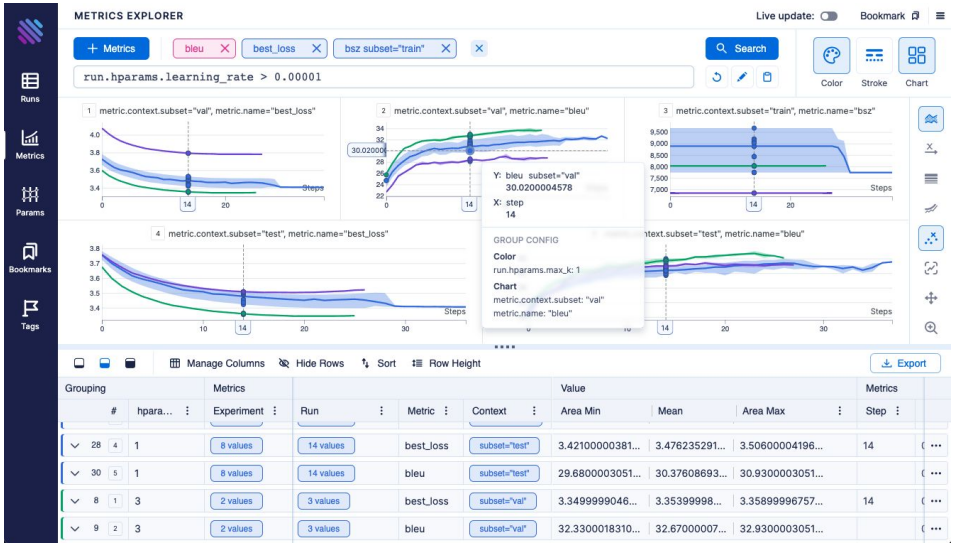
# Create an ExecutionType, e.g., Trainer
trainer_type = metadata_store_pb2.ExecutionType()
trainer_type.name = "Trainer"
trainer_type.properties["state"] = metadata_store_pb2.STRING
trainer_type_id = store.put_execution_type(trainer_type)

# Register the Execution of a Trainer run
trainer_run = metadata_store_pb2.Execution()
trainer_run.type_id = trainer_type_id
trainer_run.properties["state"].string_value = "RUNNING"
[run_id] = store.put_executions([trainer_run])
```

Übersicht ausgewählter Tools und Frameworks - Open-Source

- Aim
  - Pro:
    - UI Scalability angeblich besser als bei Mlflow
  - Contra:
    - Junges Projekt mit kleiner Community






```
# Initialize a new run
run = Run()

# Log run parameters
run["hparams"] = {"learning_rate": 0.001, …}

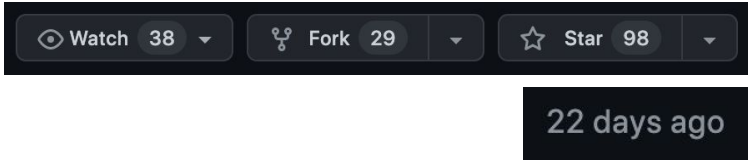
# Log artefacts
for step in range(1000):
    run.track(loss_val, …)
```

Übersicht ausgewählter Tools und Frameworks - “Closed-Source”

- Neptune AI und Weights & Biases
  - Pro:
    - Hosted
    - Bessere UI
      - Im Gegensatz zu Mlflow keine Probleme mit vielen Datenpunkten
    - Visualisierungen für z.B.
      - Modell-Parameter-Verteilungen, Gradienten, ...
      - Throughput und Auslastung der GPUs/Maschinen
    - User Management
    - ...
  - Contra:
    - Nicht Open Source

Übersicht ausgewählter Tools und Frameworks - Hyper-Scaler

- AWS Sagemaker Experiments
  - Pro:
    - Managed Service
    - Sagemaker *scheint* der “most-mature” managed Service zu sein
- Vertex AI metadata
  - Pro:
    - Managed Service
  - Contra:
    - Dokumentation empfohl bis vor Kurzem die Nutzung der Rest-API
    - PythonSDK Dokumentation ist viel besser geworden seit Kohorte 1




```
my_experiment = smexperiments.experiment.Experiment.create(...)
my_trial = my_experiment.create_trial(trial_name='linear-learner')

linear = sagemaker.estimator.Estimator(...)

linear.set_hyperparameters(...

linear.fit(inputs={'train': s3_train_data}, experiment_config={
            "ExperimentName": my_experiment.experiment_name,
            "TrialName": my_trial.trial_name,
            "TrialComponentDisplayName": "MNIST-linear-learner",
        },)
```

Übersicht ausgewählter Tools und Frameworks

- Empfehlung? Keine *silver bullet*
  - Open-Source
    - Mlflow hat die größte Community und gute Features.
      - Funktioniert bei uns im großen und ganzen gut
      - Gibt es auch als managed Service
  - “Closed-Source”
    - Bessere Visualisierungen und UI
    - Gehostet
    - Benutzen manche Researcher bei uns
  - Managed Service
    - Macht Sinn wenn man eine managed AI-Platform benutzt

- Takeaway
  - Am besten so *“wegabstrahieren”*, dass leichter austauschbar ist
    - z.B. mit Hilfe von Pytorch-Lightning Loggers

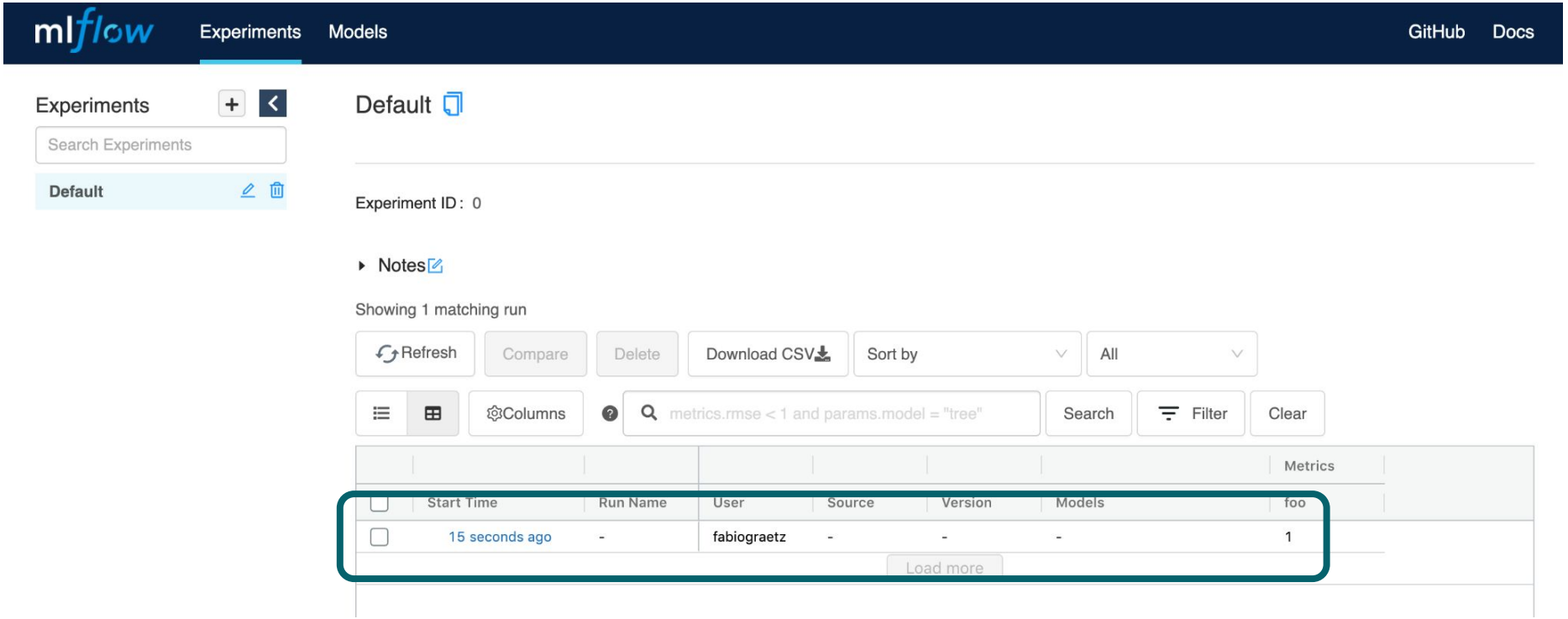
Hands-on Übung: Training und Experiment-Tracking mit MIflow

- Minimal working example
  - Tracking-Server starten
    - `pip install mlflow`
    - `mlflow server --port 5001`
  - In Python

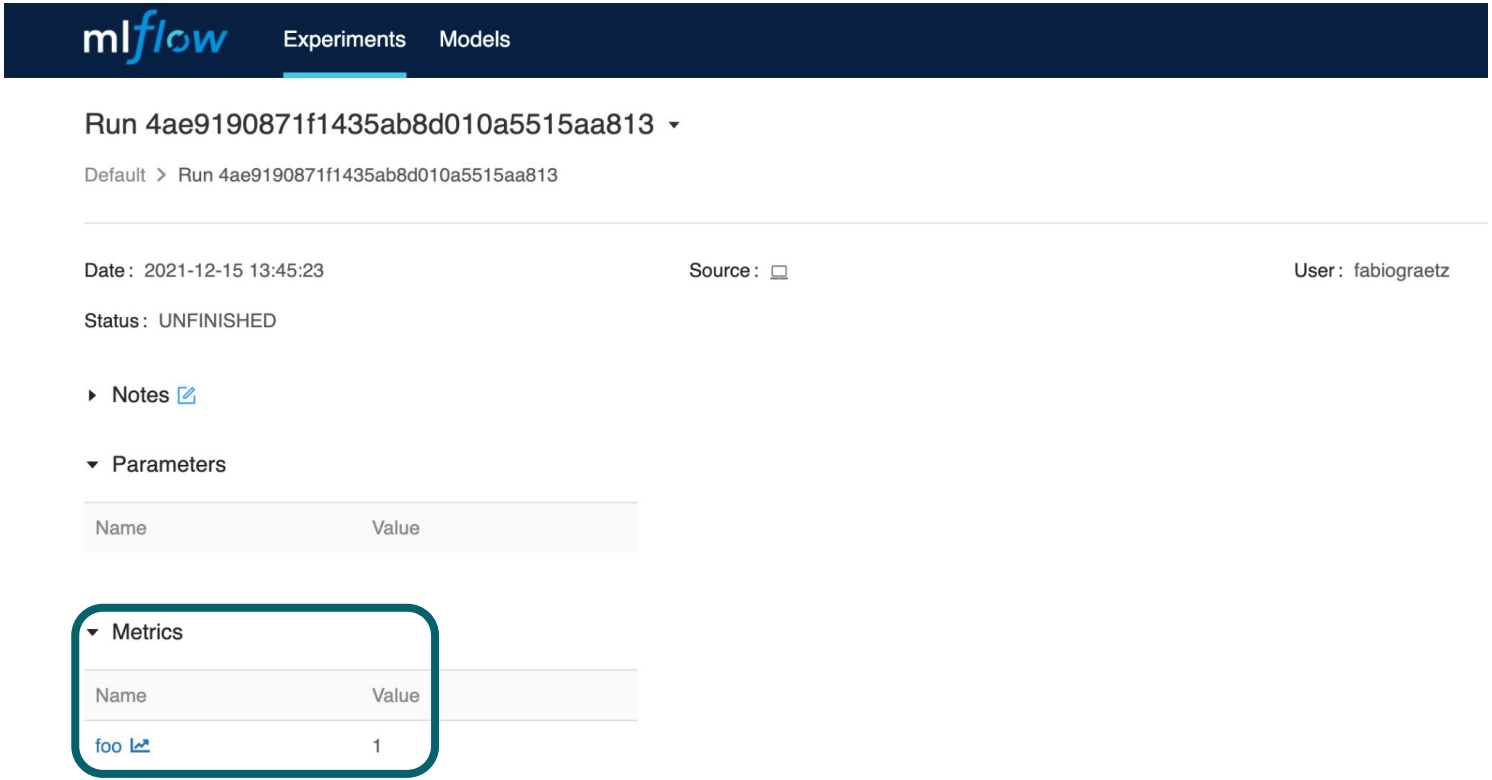
```
>>> import mlflow
>>> mlflow.set_tracking_uri("http://localhost:5001")
>>> mlflow.start_run()
>>> mlflow.log_metric("foo", 1)
```

  - Tracking UI auf http://localhost:5001 erreichbar

Hands-on Übung: Training und Experiment-Tracking mit MIflow

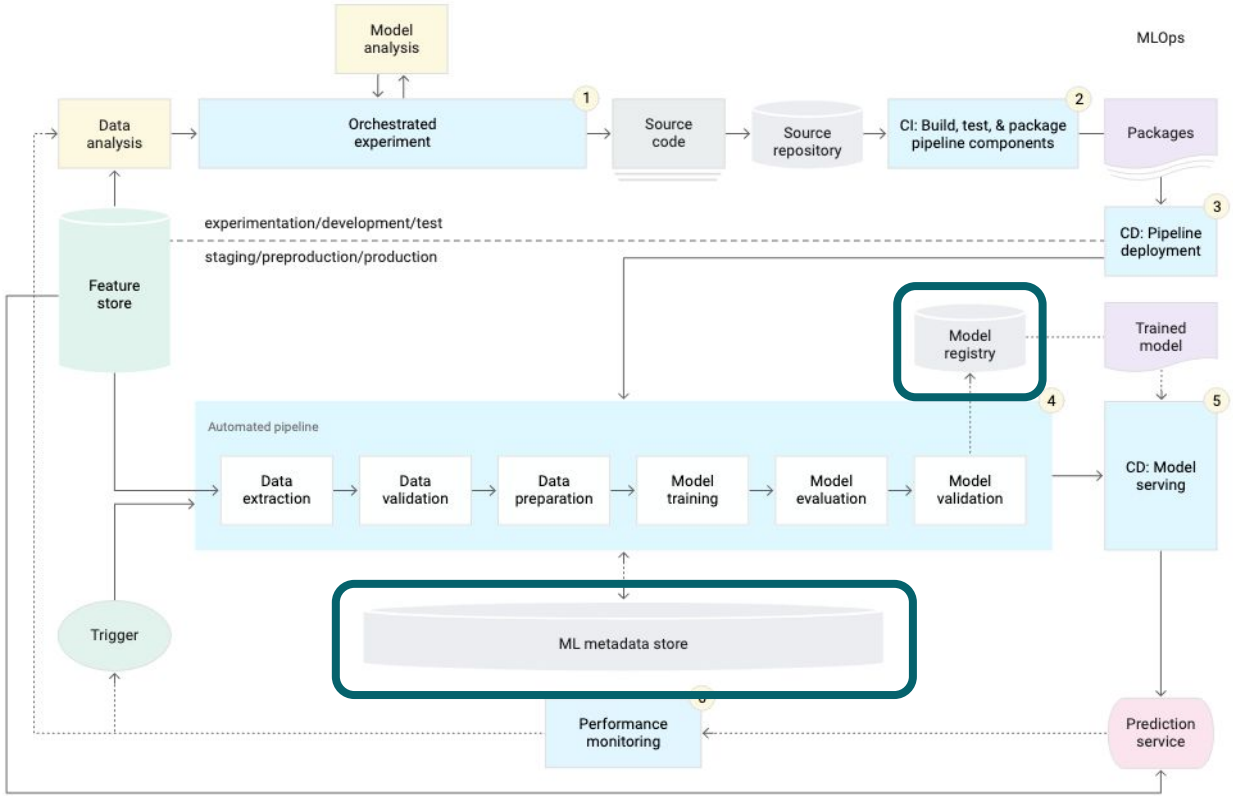


Hands-on Übung: Training und Experiment-Tracking mit MIflow

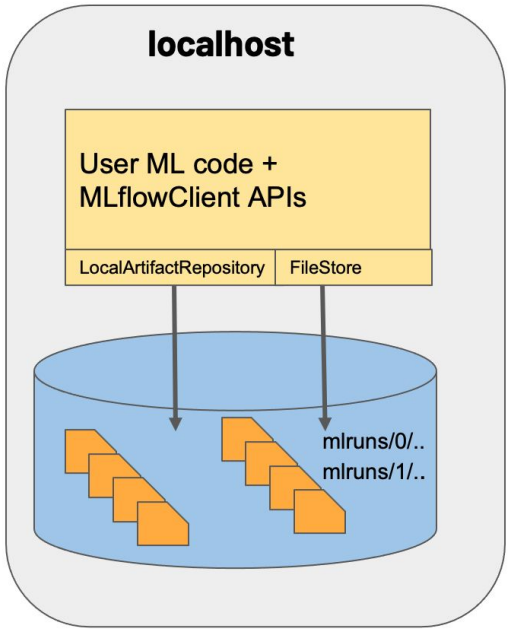


Hands-on Übung: Training und Experiment-Tracking mit MIflow

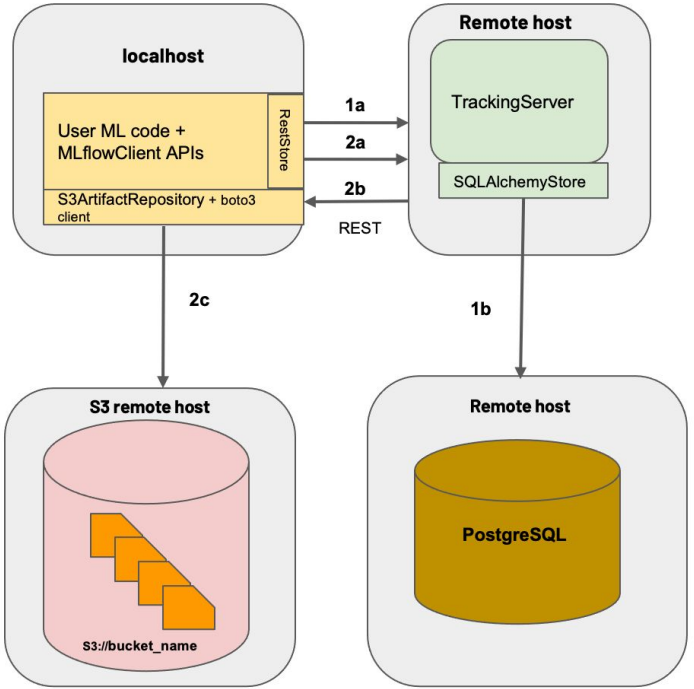
- Todos
  - In MLOps-Sandbox in K8s integrieren
  - Dummy Modell trainieren, Metriken loggen
  - Modell-Artefakt registrieren



Hands-on Übung: MIflow-Tracking-Server Deployment



Scenario 1: MLflow on the localhost

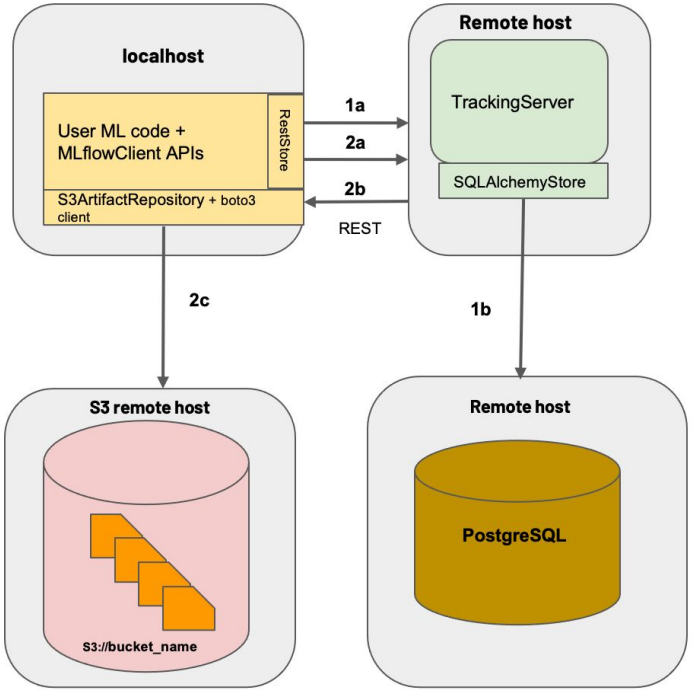


Scenario 4: `mlflow server --backend-store-uri postgresql://URI --default-artifact-root S3:/bucket_name --host remote_host`

Source

Hands-on Übung: MIflow-Tracking-Server Deployment

```
.
├── infrastructure
│   ├── blob_storage
│   │   ├── …
│   │   └── minio.yaml
│   └── mlflow_tracking_server
│       ├── Dockerfile
│       └── manifests
│           ├── mlflow_server_deployment.yaml
│           ├── mlflow_server_service.yaml
│           ├── namespace.yaml
│           ├── postgres_deployment.yaml
│           └── postgres_service.yaml
```



Scenario 4: `mlflow server --backend-store-uri postgresql://URI --default-artifact-root S3:/bucket_name --host remote_host`

Hands-on Übung

- `export GITHUB_REPO=https://github.com/<owner>/<repo_name>.git`
- `export WORKFLOW=workflow_part2`
- `cat argo_cicd_workflow.yaml | envsubst | kubectl -n argo create -f -`

```python
@workflow
def pipeline(test_size: float = 0.3, n_estimators: int = 100, min_auc: float = 0.95) -> str:
    """Preprocess data, validate data, train model, validate model, and deploy model."""
    data_uri = prepare_data(test_size=test_size)
    model_name, model_version = train_model(data_uri=data_uri, n_estimators=n_estimators, min_auc=min_auc)
    endpoint_uri = deploy_model(model_name=model_name, model_version=model_version)

    return endpoint_uri
```

```python
@task(cache=True, cache_version="1.0")
def prepare_data(test_size: float = 0.3) -> str:
    """Download a dataset, validate it, and return blob storage uri."""

    from sklearn import datasets
    from sklearn.model_selection import train_test_split

    X, y = datasets.load_breast_cancer(return_X_y=True, as_frame=True)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)

    """
    Validate data:
    - schema skews
      - E.g. any missing or unexpected columns?
    - values skews
      - Any unexpected `None` values?
      - Any unexpected feature distributions?

    Stop the pipeline if something odd is detected.
    """

    bucket = f"data/breast_cancer_test_size_{test_size}"
    for df, fn in zip([X_train, X_test, y_train, y_test], ["X_train.csv", "X_test.csv", "y_train.csv", "y_test.csv"]):
        uri = f"{bucket}/{fn}"
        save_df_in_bucket(df, uri)

    return bucket
```

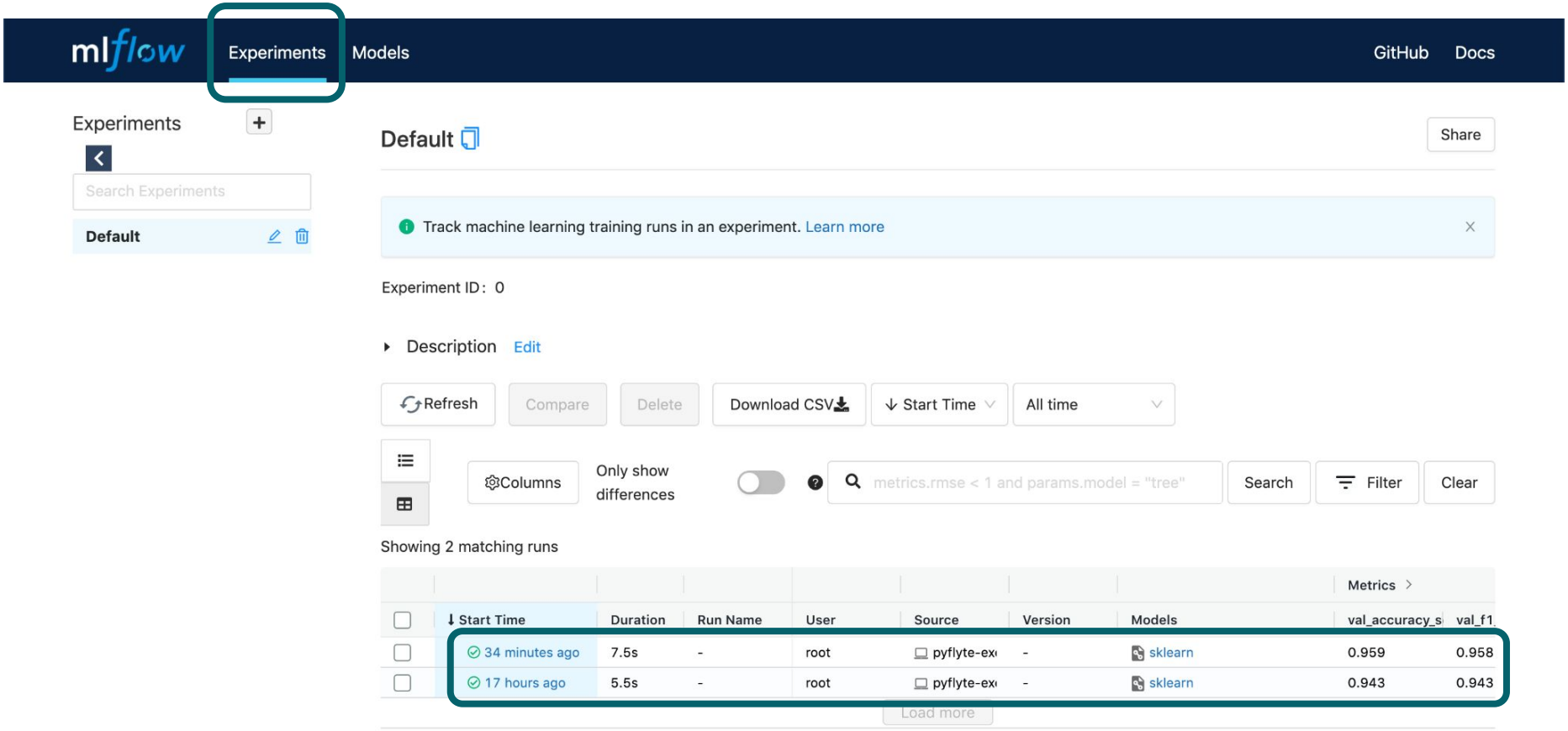
```
kubectl port-forward svc/minio 9000:9000 -n flyte
```

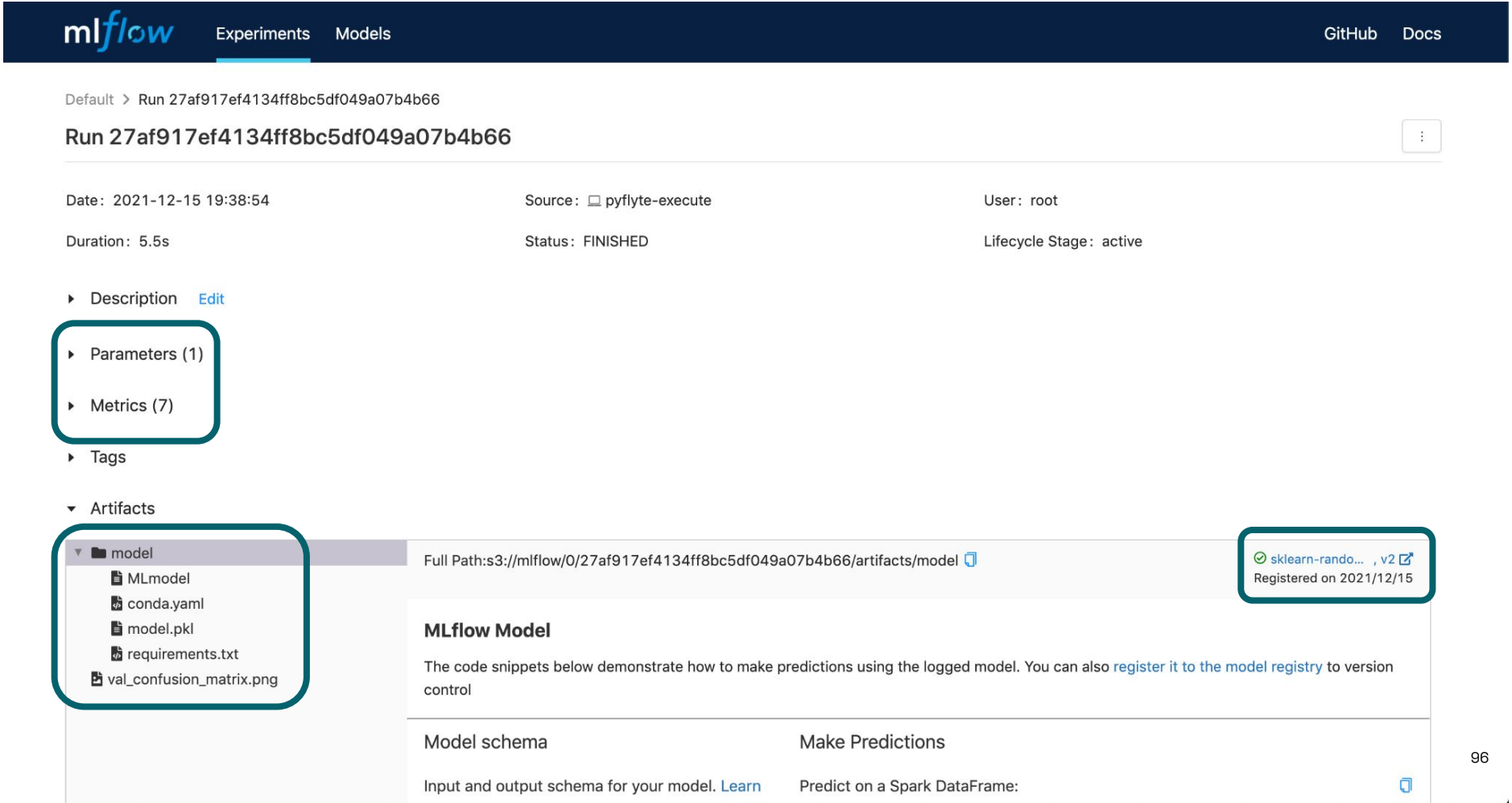
```
(sandbox) ➜  session2 git:(master) ✗ mc alias set minio http://localhost:9000 minio miniostorage --api S3v4
Added `minio` successfully.
(sandbox) ➜  session2 git:(master) ✗ mc ls minio/data
[2021-12-16 13:04:38 CET]     0B breast_cancer_test_size_0.3/
[2021-12-16 13:04:38 CET]     0B breast_cancer_test_size_0.4/
(sandbox) ➜  session2 git:(master) ✗ mc ls minio/data/breast_cancer_test_size_0.3/
[2021-12-15 18:53:32 CET]  36KiB X_test.csv
[2021-12-15 18:53:32 CET]  84KiB X_train.csv
[2021-12-15 18:53:32 CET]  1002B y_test.csv
[2021-12-15 18:53:32 CET] 2.3KiB y_train.csv
(sandbox) ➜  session2 git:(master) ✗
```

```
mlflow.set_tracking_uri("http://mlflow-server-service.mlflow.svc.cluster.local:5000")
```

```
@task(cache=True, cache_version="1.0")
def train_model(data_uri: str, n_estimators: int = 100, min_auc: float = 0.95) -> Tuple[str, str]:
    """
    Train and evaluate model on prepared data at `data_uri`.

    Args:
        data_uri (str): uri of prepared data in blob storage.
        n_estimators (int): number of trees in the random forest
        min_auc (float): minimum required area under the roc curve

    Returns:
        model_name (str): name of the model
        model_version (str): version of the model
    """
    from sklearn.ensemble import RandomForestClassifier

    X_train = load_df_from_bucket(f"{data_uri}/X_train.csv")
    X_test = load_df_from_bucket(f"{data_uri}/X_test.csv")
    y_train = load_df_from_bucket(f"{data_uri}/y_train.csv")
    y_test = load_df_from_bucket(f"{data_uri}/y_test.csv")

    with mlflow.start_run() as run:
```

```
    with mlflow.start_run() as run:
        clf = RandomForestClassifier(n_estimators=n_estimators)
        clf.fit(X_train, y_train)
        mlflow.sklearn.eval_and_log_metrics(clf, X_test, y_test, prefix="val_")
        mlflow.log_param("n_estimators", n_estimators)

        """
        Validate model and stop pipeline if performance is lower than expected.
        """
        client = MlflowClient()
        metric = client.get_metric_history(run.info.run_id, "val_roc_auc_score")
        if metric[0].value < min_auc:
            raise Exception("Stop training due to insufficient performance.")

        model_dir = "model"
        mlflow.sklearn.log_model(
            clf,
            model_dir,
        )
        model_details = mlflow.register_model(
            model_uri=mlflow.get_artifact_uri(model_dir), name="sklearn-random-forest"
        )

    logging.info("Model trained and evaluated")

    return model_details.name, model_details.version
```


mlflow
Experiments
Models
GitHub
Docs
Experiments
Search Experiments
Default
Default
Share
Track machine learning training runs in an experiment. Learn more
Experiment ID: 0
Description
Edit
Refresh
Compare
Delete
Download CSV
Start Time
All time
Columns
Only show differences
metrics.rmse < 1 and params.model = "tree"
Search
Filter
Clear
Showing 2 matching runs
Metrics
Start Time
Duration
Run Name
User
Source
Version
Models
val_accuracy_s
val_f1
34 minutes ago
7.5s
-
root
pyflyte-ex
-
sklearn
0.959
0.958
17 hours ago
5.5s
-
root
pyflyte-ex
-
sklearn
0.943
0.943
Load more


mlflow
Experiments
Models
GitHub
Docs
Default > Run 27af917ef4134ff8bc5df049a07b4b66
Run 27af917ef4134ff8bc5df049a07b4b66
Date: 2021-12-15 19:38:54
Source: pyflyte-execute
User: root
Duration: 5.5s
Status: FINISHED
Lifecycle Stage: active
Description
Edit
Parameters (1)
Metrics (7)
Tags
Artifacts
model
MLmodel
conda.yaml
model.pkl
requirements.txt
val_confusion_matrix.png
Full Path:s3://mlflow/0/27af917ef4134ff8bc5df049a07b4b66/artifacts/model
sklearn-rando... , v2
Registered on 2021/12/15
MLflow Model
The code snippets below demonstrate how to make predictions using the logged model. You can also register it to the model registry to version control
Model schema
Input and output schema for your model. Learn
Make Predictions
Predict on a Spark DataFrame:

Date: 2021-12-15 19:38:54

Source: pyflyte-execute

User: root

Duration: 5.5s

Status: FINISHED

Lifecycle Stage: active

▸ Description Edit

▾ Parameters (1)

Name	Value
n_estimators	100

▾ Metrics (7)

Name	Value
val_accuracy_score	0.943
val_f1_score	0.943
val_log_loss	0.257
val_precision_score	0.943
val_recall_score	0.943
val_roc_auc_score	0.986
val_score	0.943

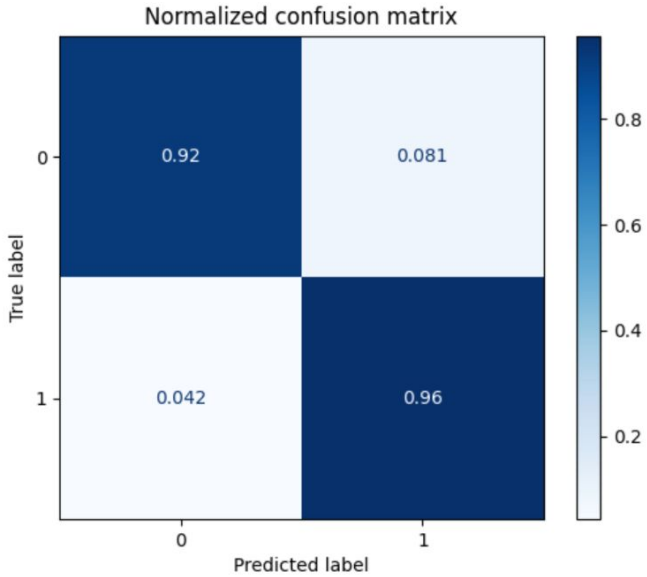
val_score	0.943

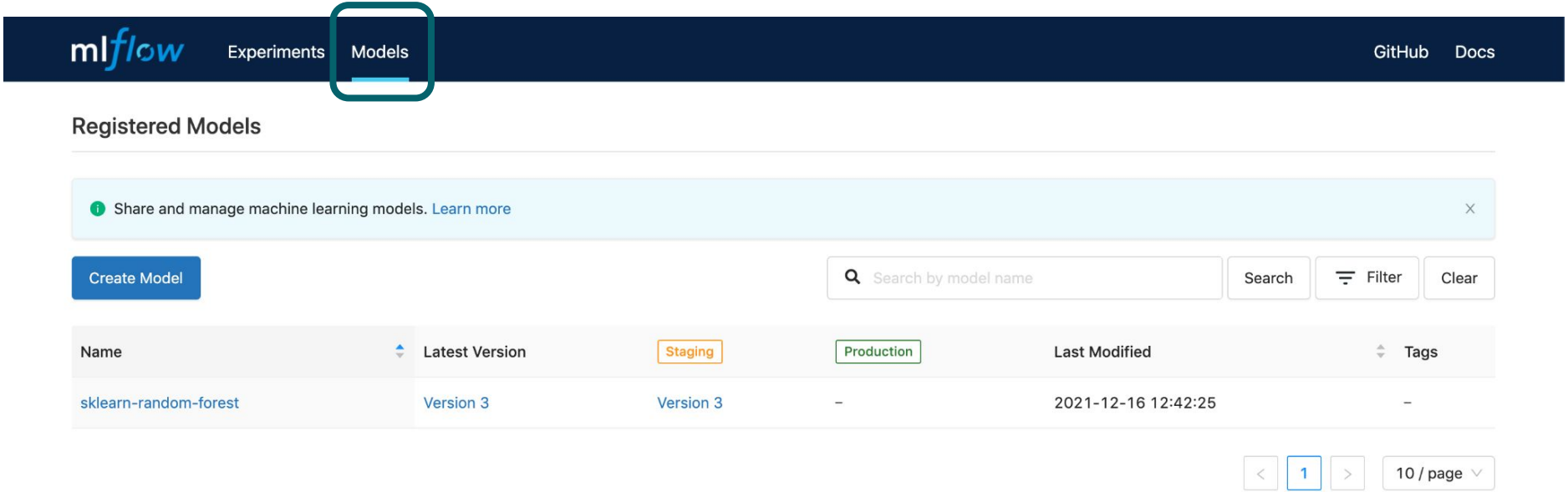
▸ Tags

▾ Artifacts

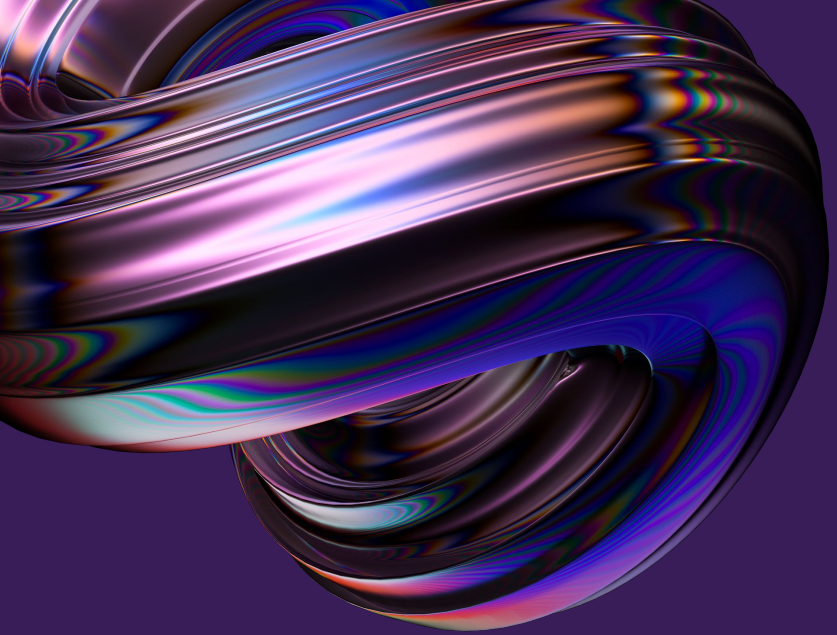
- model
  - MLmodel
  - conda.yaml
  - model.pkl
  - requirements.txt
- val_confusion_matrix.png

Full Path:s3://mlflow/0/27af917ef4134ff8bc5df049a07b4b66/artifacts/val_confusion_matrix.png
Size: 17.96KB




mlflow
Experiments
Models
GitHub
Docs
Registered Models
Share and manage machine learning models. Learn more
Create Model
Search by model name
Search
Filter
Clear
Name
Latest Version
Staging
Production
Last Modified
Tags
sklearn-random-forest
Version 3
Version 3
–
2021-12-16 12:42:25
–
1
10 / page

```python
@task
def deploy_model(model_name: str, model_version: str) -> str:
    """
    Deploy the model with name `model_name` and version `model_version`.

    Args:
        model_name (str): name of the model in the mlflow model registry
        model_version (str): version of the model in the mlflow model registry
    Returns:
        endpoint_uri (str): the path of the deployed model endpoint.
    """
    client = MlflowClient()

    model_details = client.get_model_version(
        name=model_name,
        version=model_version,
    )

    """
    Deploy model and validate that the deployment worked.
    """

    client.transition_model_version_stage(name=model_details.name, version=model_details.version, stage="Staging")

    return "endpoint_uri"
```

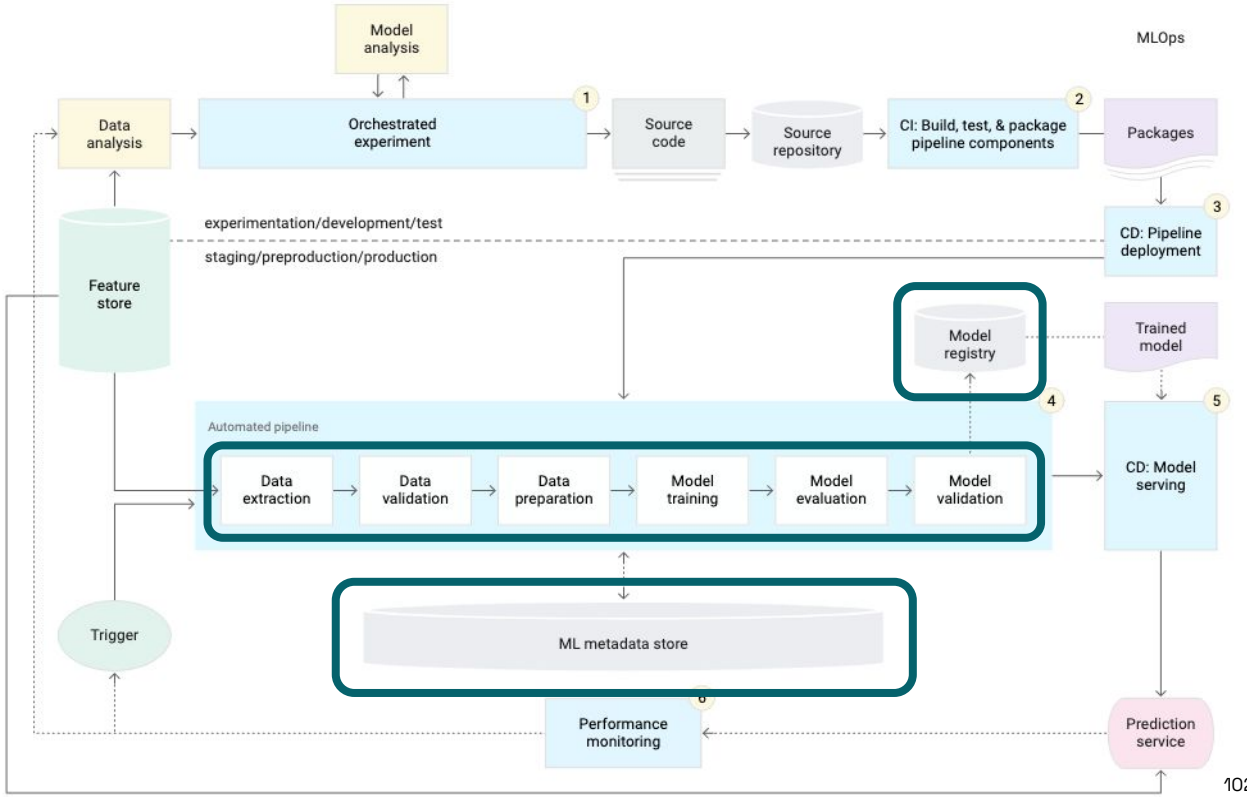


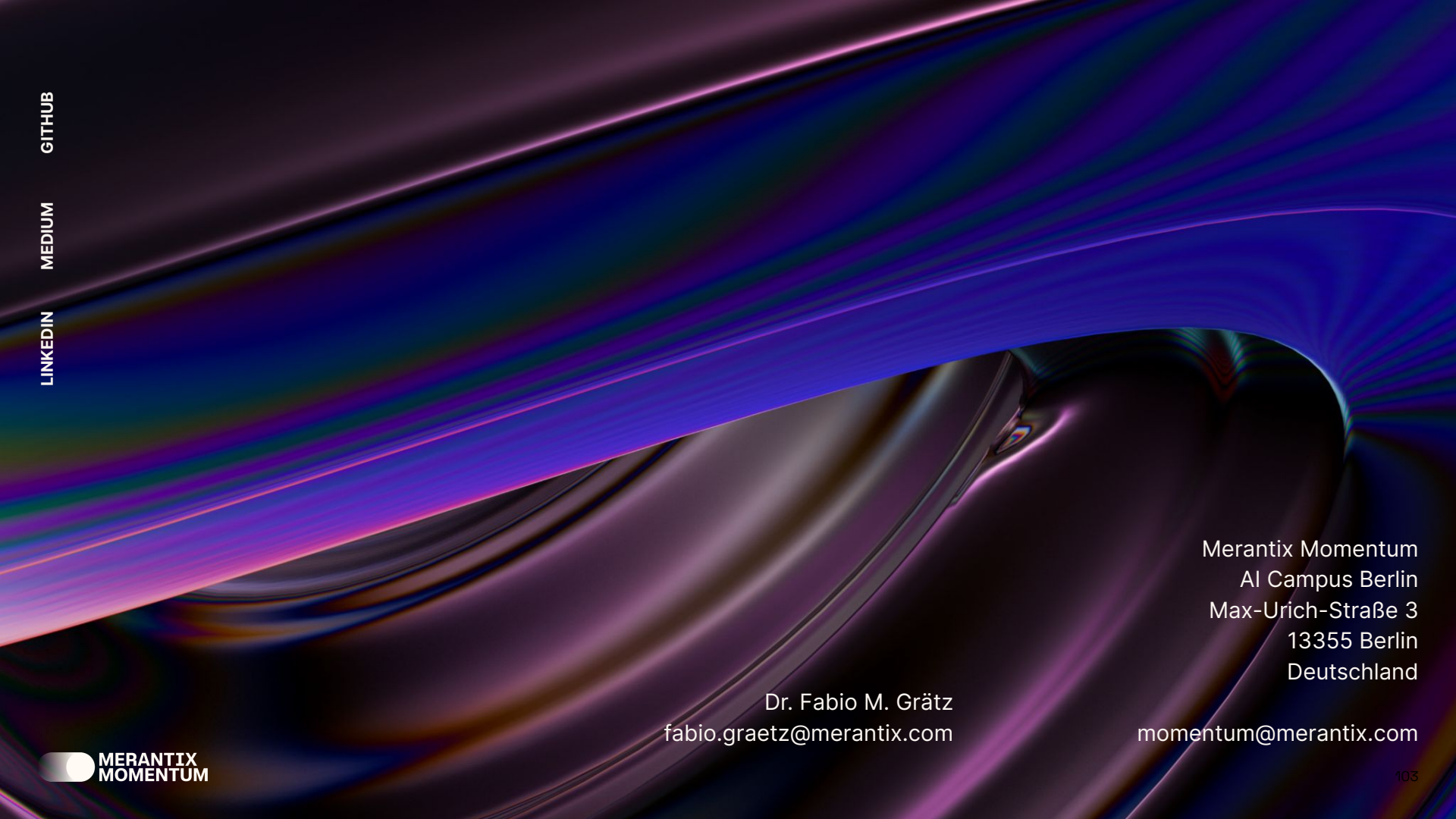
MERANTIX MOMENTUM

Zusammenfassung Session 2

Zusammenfassung Session 2

- Was ist ein ML-Metadatastore? Was ist eine Model-Registry?
- Übersicht ausgewählter Tools und Frameworks
- Hands-on Übung:
  - Deployment des Mlflow-Tracking-Severs in Kubernetes
  - Integration in unsere ML-Pipeline in Flyte





Merantix Momentum
AI Campus Berlin
Max-Urich-Straße 3
13355 Berlin
Deutschland

Dr. Fabio M. Grätz
fabio.graetz@merantix.com

momentum@merantix.com

MERANTIX
MOMENTUM