MLOps webinar

Nicholas Vachon

April 6, 2022

Agenda

- What is MLOps and why? (5 minutes)
- MLOps Components (25 – 30 minutes)
  - Reproducibility
    - Tracking
    - Environments
    - Data Versioning
  - Deployment / Release / Re-training
    - CI / CD Best practices for software parts
    - Continuous Training options
    - Automated Model promotion
  - Monitoring
    - Feature Drift detection
    - Model Performance when/if ground truth available
  - Pipelining
    - Use command line scripts for your pipeline steps
    - Develop outside Azure ML pipelines for a good developer experience
    - Deployment with Azure ML pipelines: Get all the goodies
- Technical demonstration (10 – 15 minutes)
  - Batch prediction with passive retraining and automated model promotion
  - Drift detection
- Questions (10 - 15 minutes)

NIVELOP

What is MLOps and why?

Traditional DevOps VS MLOps

Traditional DevOps

→ Version control

→ Testing your code

→ Continuous integration and delivery (CI/CD)

→ Monitoring your application

MLOPs

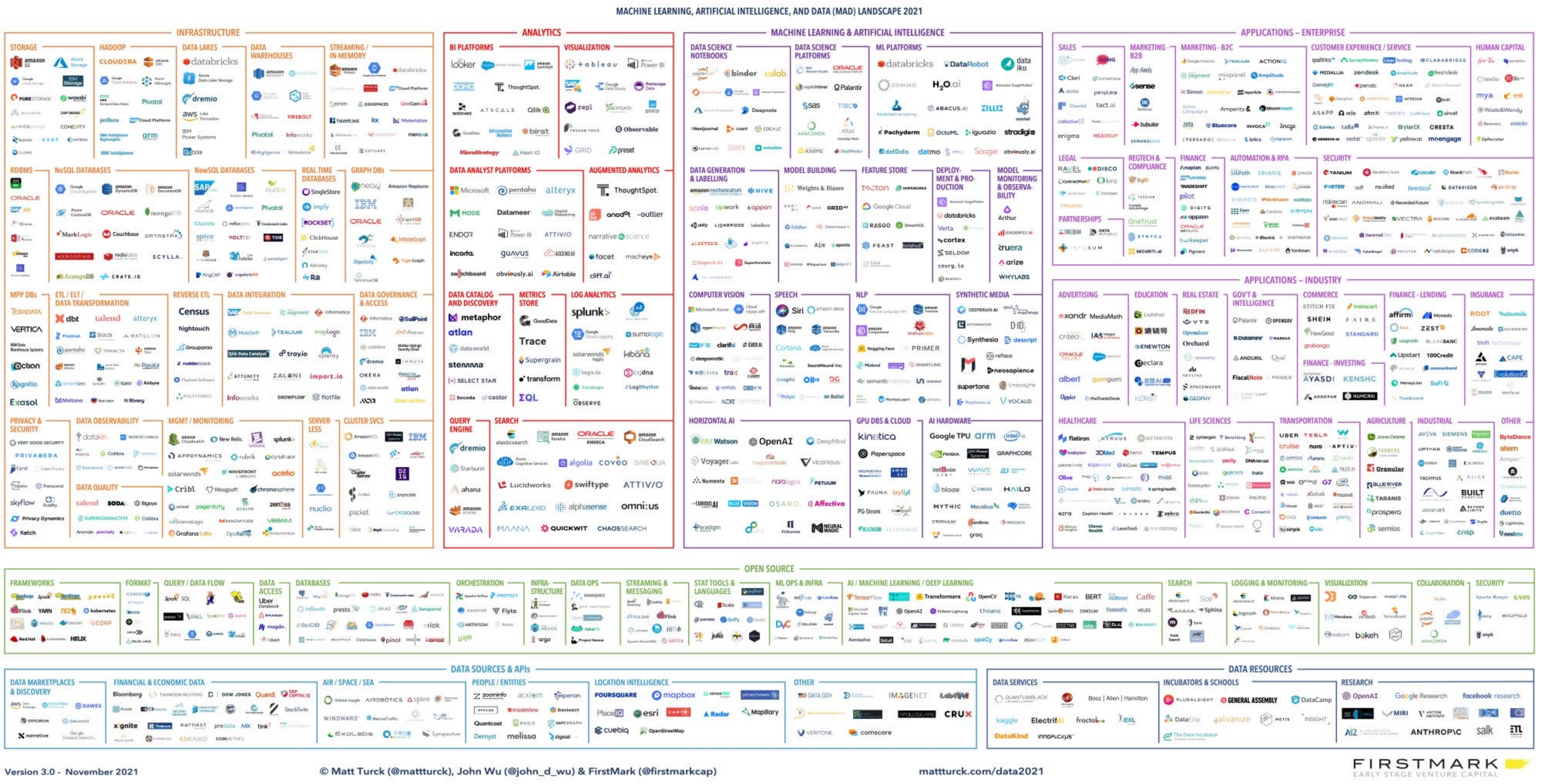
→ Apart from code, we also need to version and track **model artifacts** and **data**

→ ML is less linear and more experimental

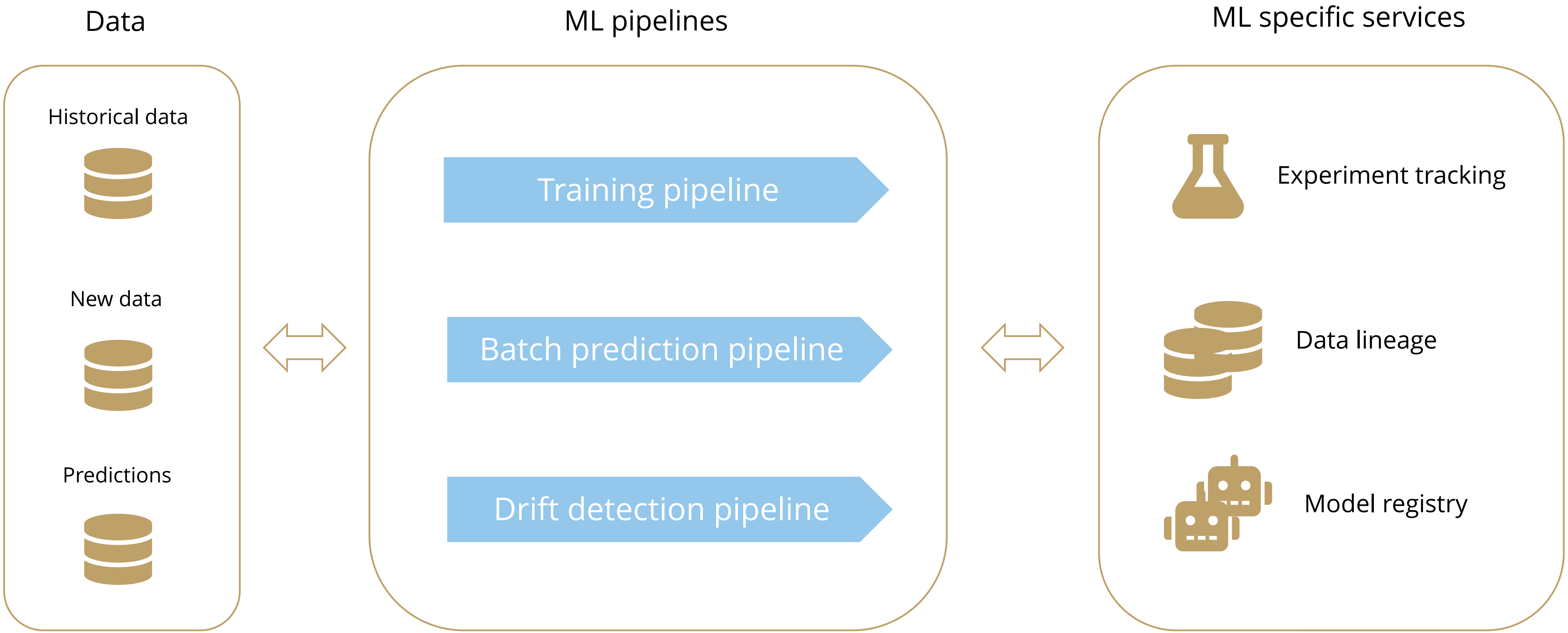
→ It's harder to test an ML model, because of the statistical nature of ML

→ **Even without code changes,** systems need to be periodically trained and tested with new data

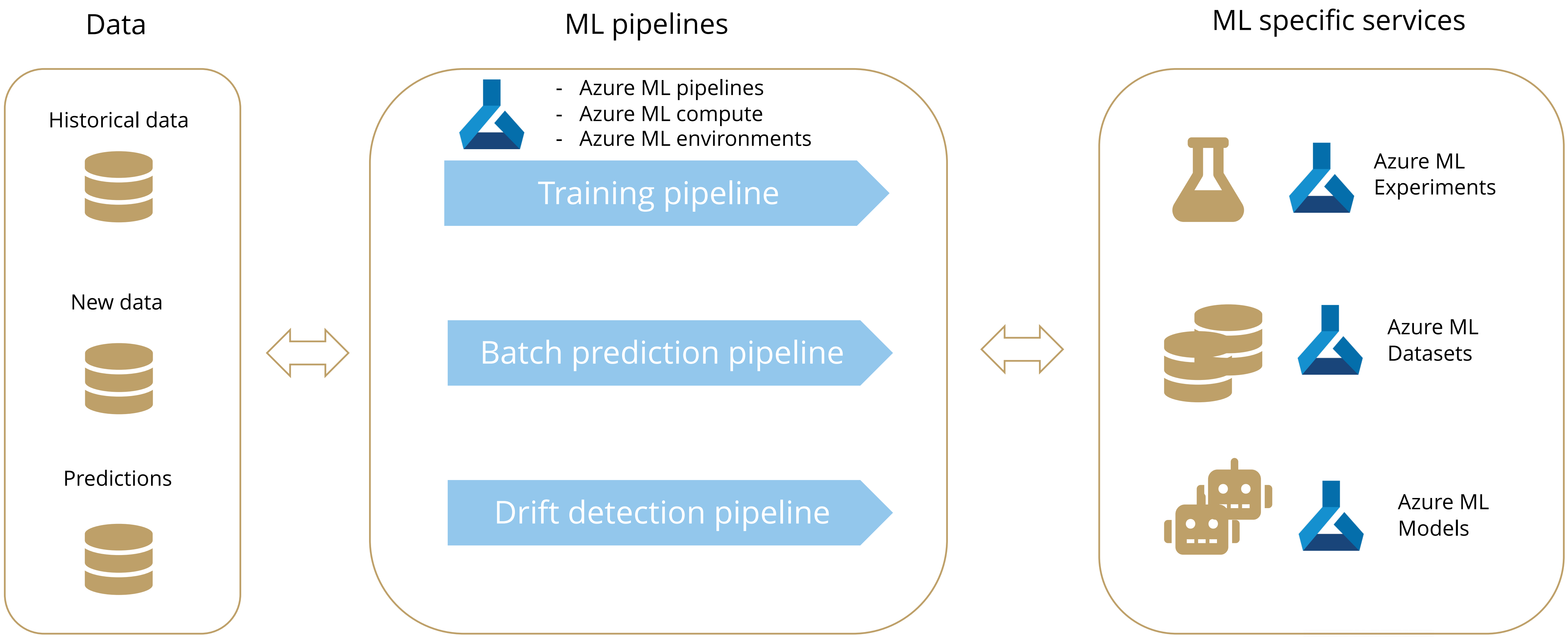
MLOps tooling landscape… HELP!



Batch inference ML system



Batch inference ML system



MLOps Components

Reproducibility

Tracking

Environments

Data Versioning

Tracking

Why do we track experiences?

→ Trust in technology comes from stability, ability to reproduce great results (~~luck~~)

→ Reproducibility is more complicated than in traditional software development

→ Experiments are a central part of data science

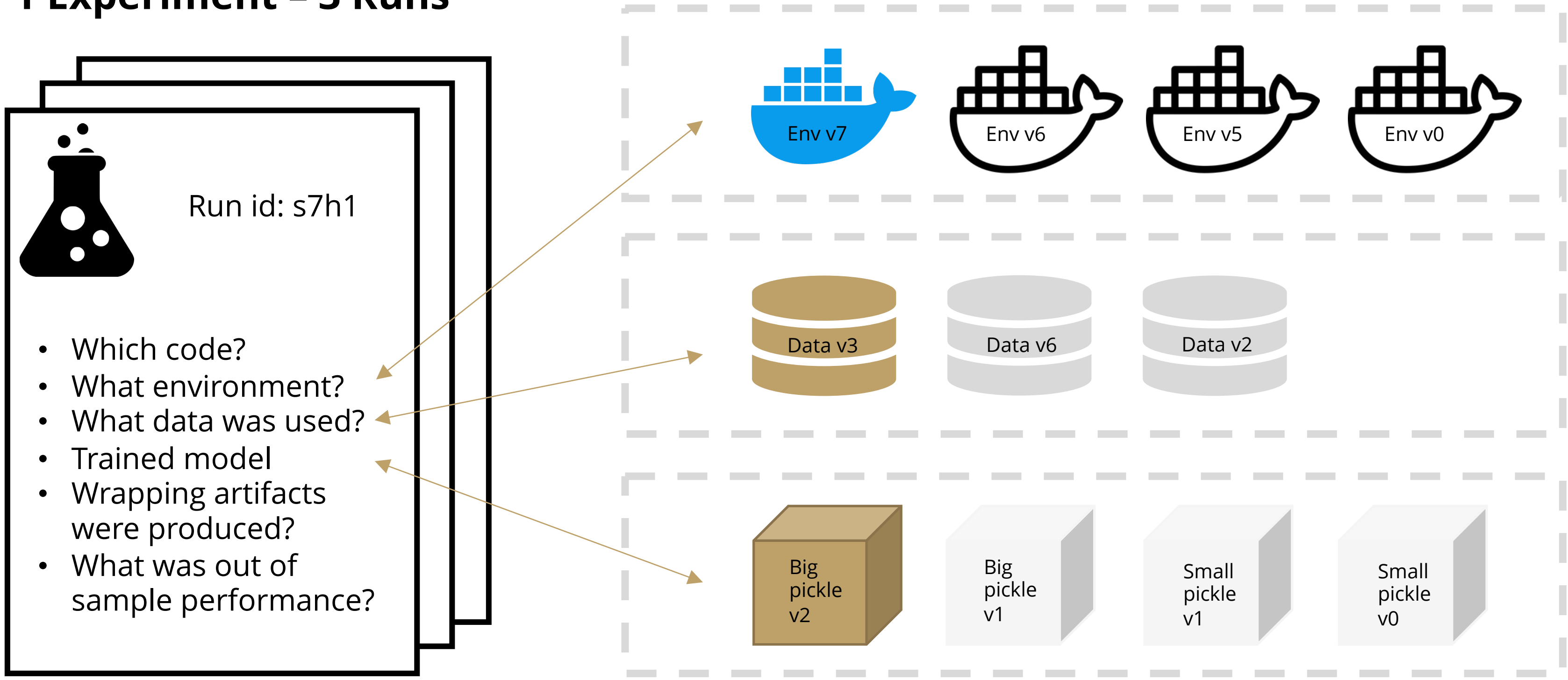
→ Logging experiments forces one to build a working pipeline as early as possible.

NIVELOP

Tracking

Training pipeline

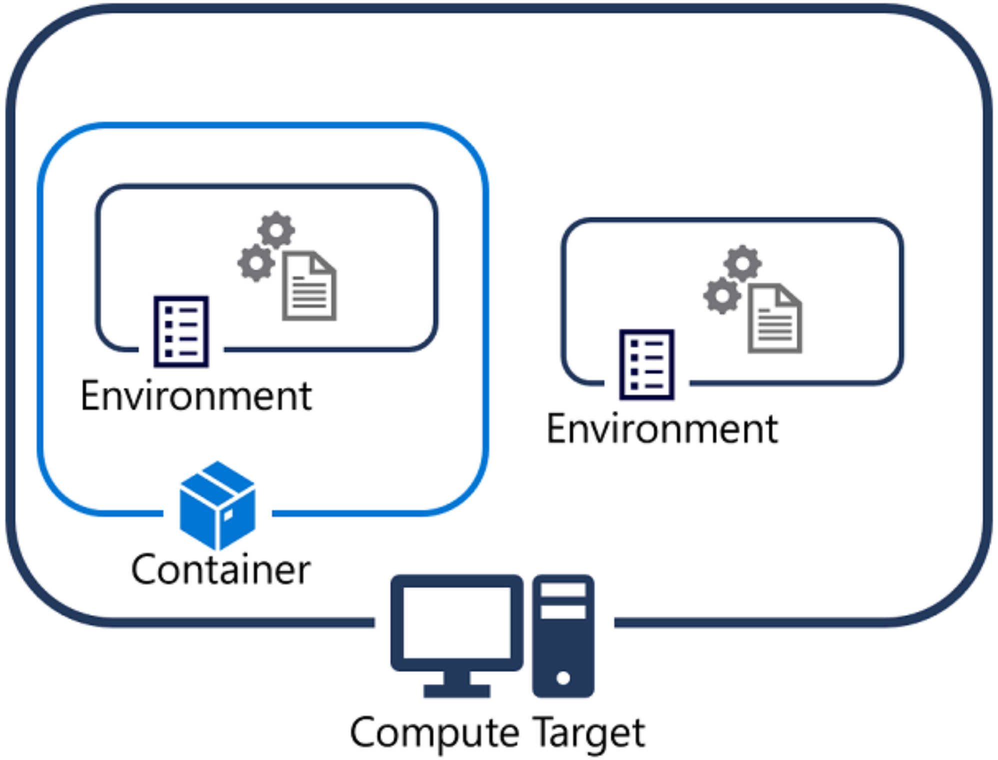
1 Experiment – 3 Runs



Environments

AzureML **Environments** class:

- → Automatic tracking with each run
- → Docker
  - → Abstracted
  - → Fully controlled
    - → Personal images
    - → Dockerfiles
- → Reusable, hence faster



NIVELOP

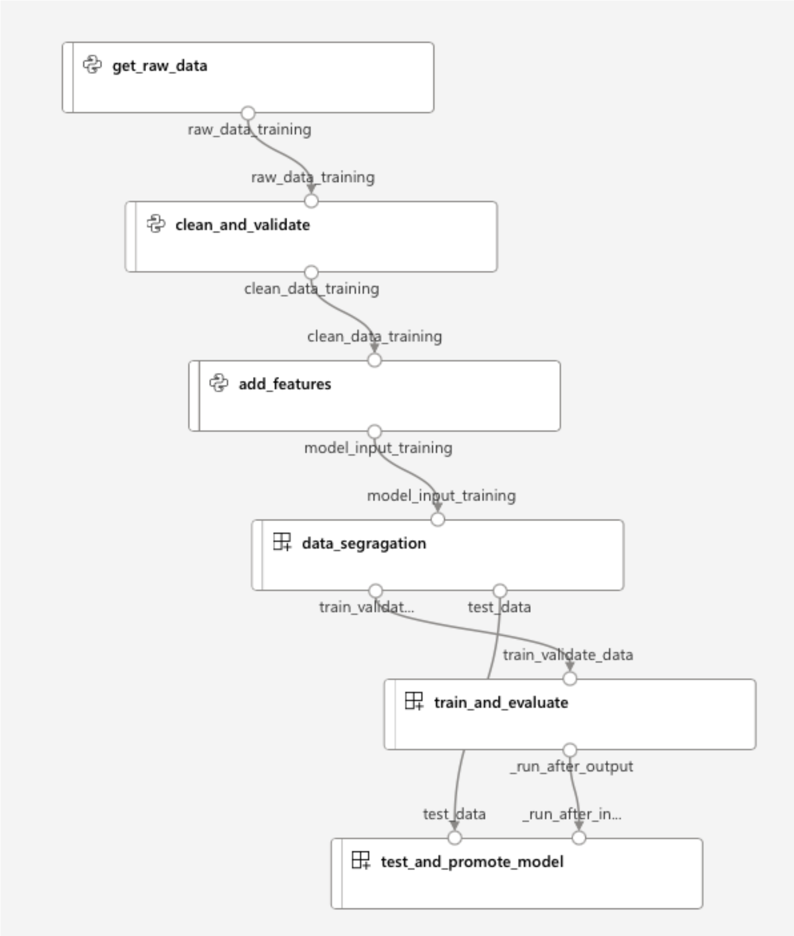
Data Versioning

Tracking data is important:

- → Debugging models
- → Insights into models
- → Comparability

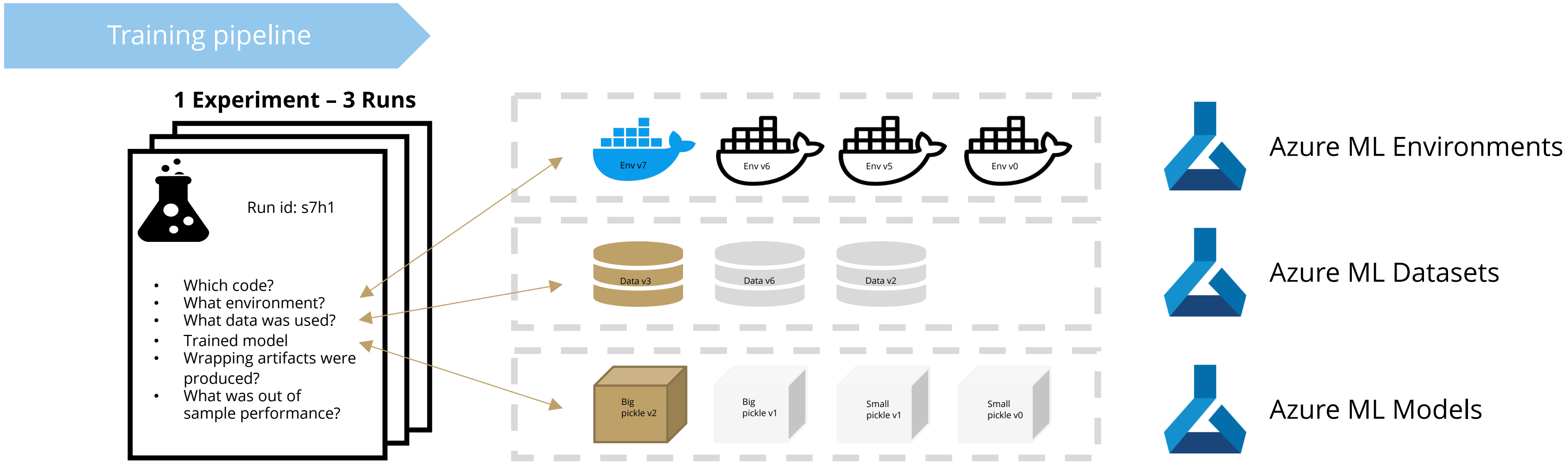
AzureML **Datasets:**

- → Packaged data objects
- → Linked to runs
- → Versioned
- → Easily consumed in experiments and pipelines



NIVELOP

Tracking

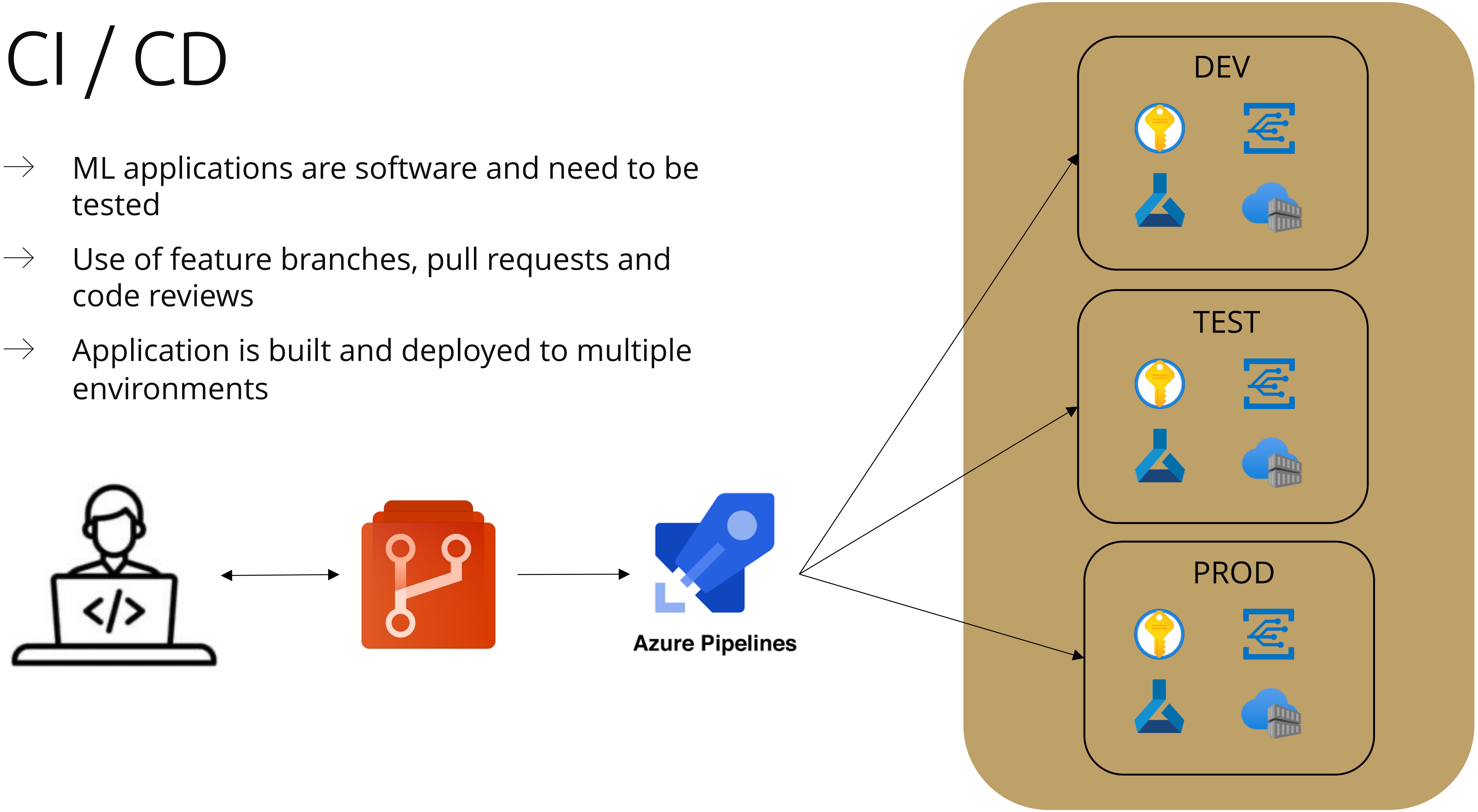

Training pipeline
1 Experiment – 3 Runs
Run id: s7h1
• Which code?
• What environment?
• What data was used?
• Trained model
• Wrapping artifacts were produced?
• What was out of sample performance?
Env v7
Env v6
Env v5
Env v0
Data v3
Data v6
Data v2
Big pickle v2
Big pickle v1
Small pickle v1
Small pickle v0
Azure ML Environments
Azure ML Datasets
Azure ML Models

Deployment / Re-training

- CI / CD Best practices for software parts
- Continuous Training options
- Automated Model promotion

CI / CD

- → ML applications are software and need to be tested
- → Use of feature branches, pull requests and code reviews
- → Application is built and deployed to multiple environments



Continous Training

Maturity

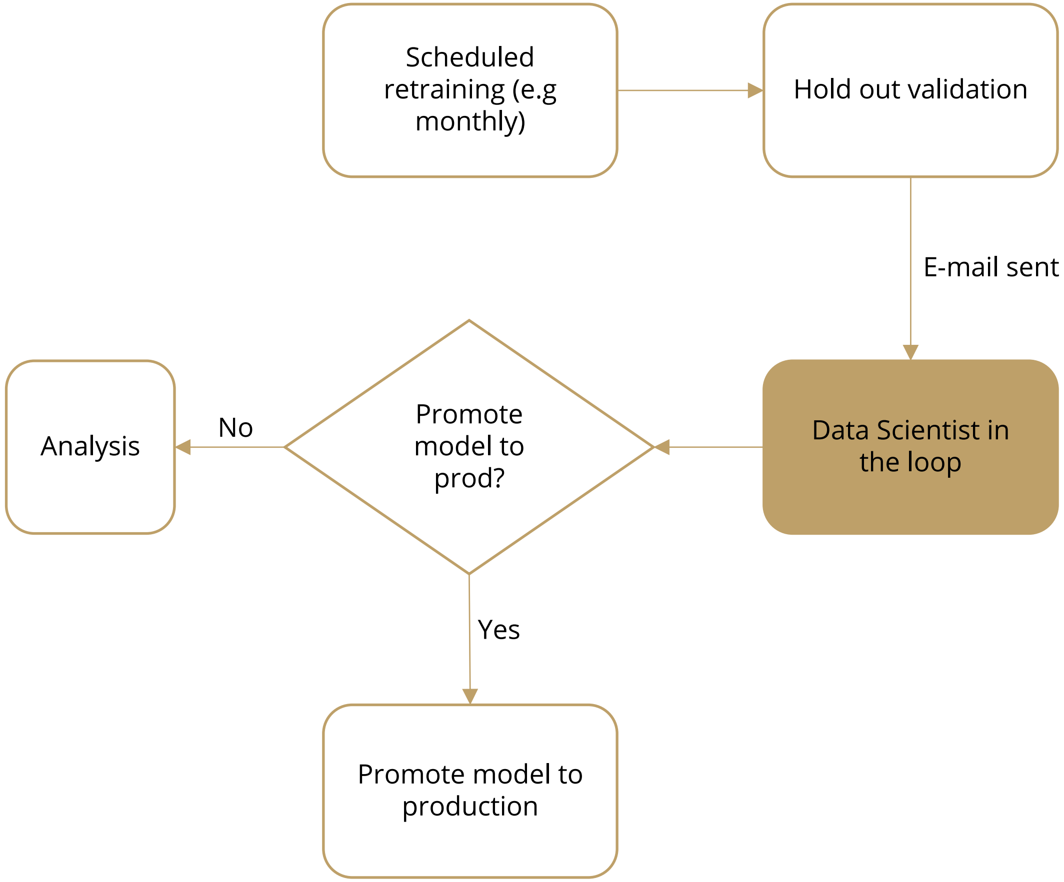
Passive retraining
Human in the loop

Passive retraining
Automated testing of model

Active retraining
Triggered by event

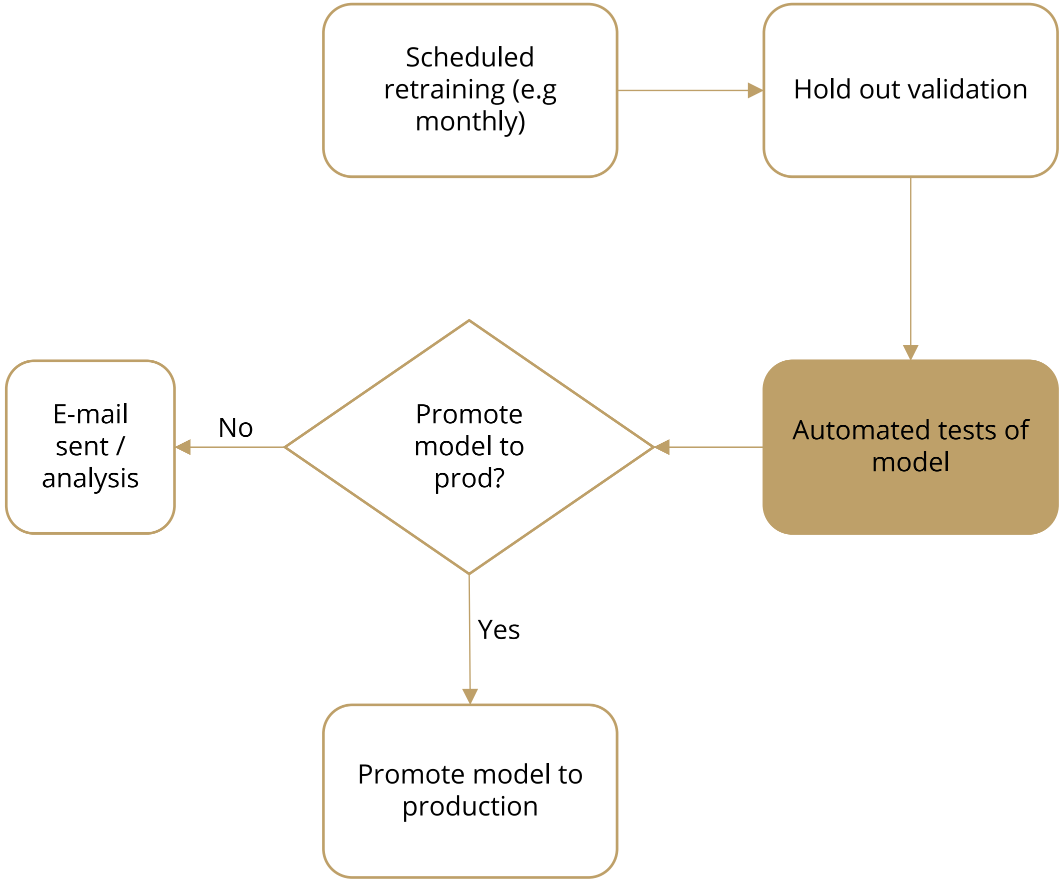
Passive retraining – human in the loop

- Training pipeline runs on a schedule
- Email sent to data scientist when model is trained and registered in the model registry
- Data scientist manually decides to promote the model.



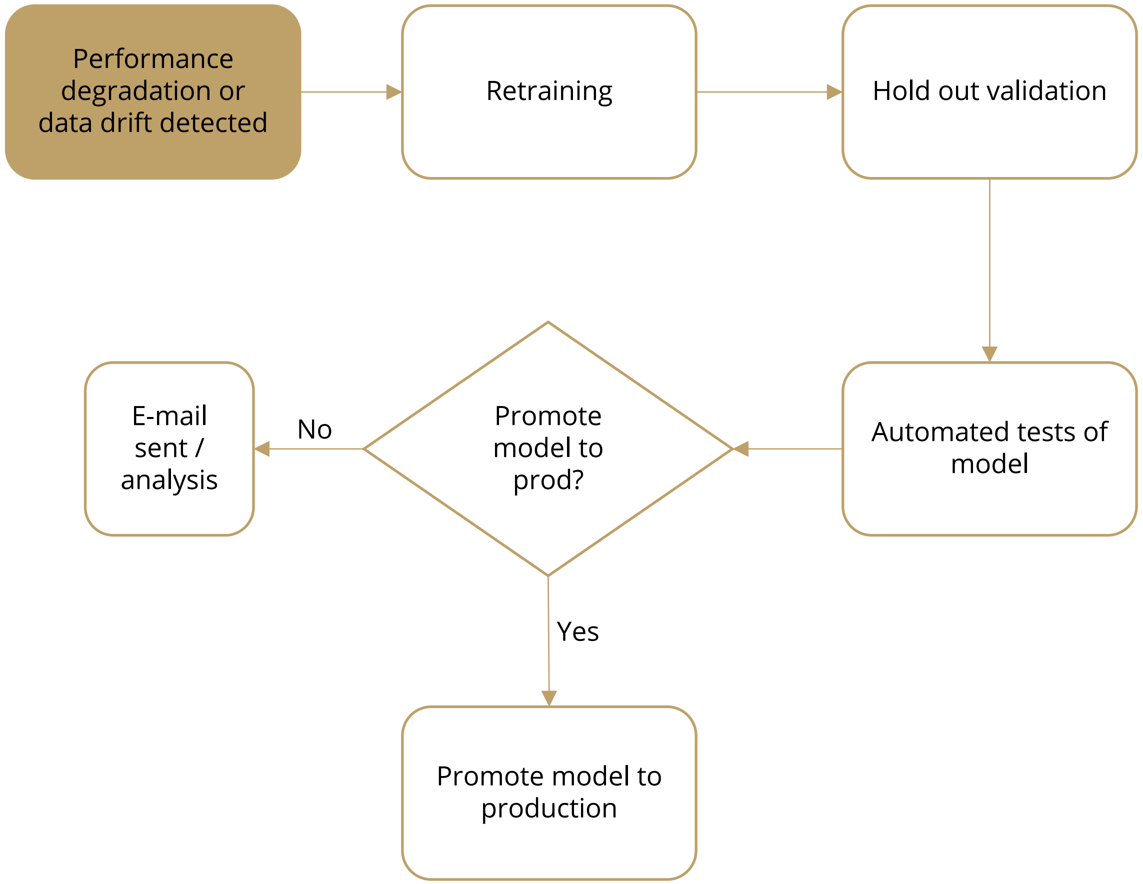
Passive retraining – Automated

- Training pipeline runs on a schedule
- Automated model tests and model promotion
  - Test of performance metric
  - Tests of edge cases
  - Fairness tests
  - Compare to current production model
- Email is sent to data scientist if new model fails tests



Active retraining

→ Training pipeline is triggered by model performance degradation or data drift



Monitoring

Traditional monitoring

Data quality

Feature Drift detection

Model Performance when/if ground truth available

Monitoring

→ Traditional monitoring:
  → Did the pipeline run fail?
  → What where in the logs?

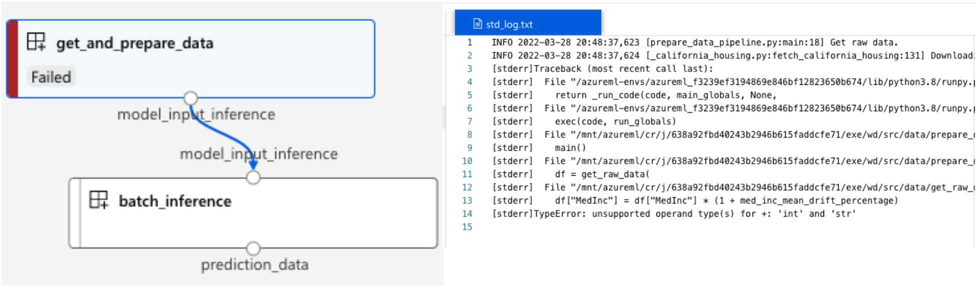
Alerting

Subscribe to Azure ML system topic event

Send email relevant Data Scientist with Logic App

Investigating

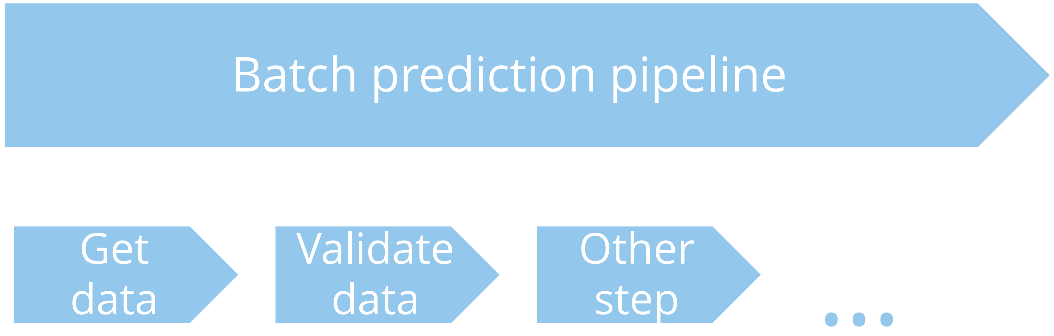
Data Scientist debugs the problem in Azure ML workspace portal



NIVELOP

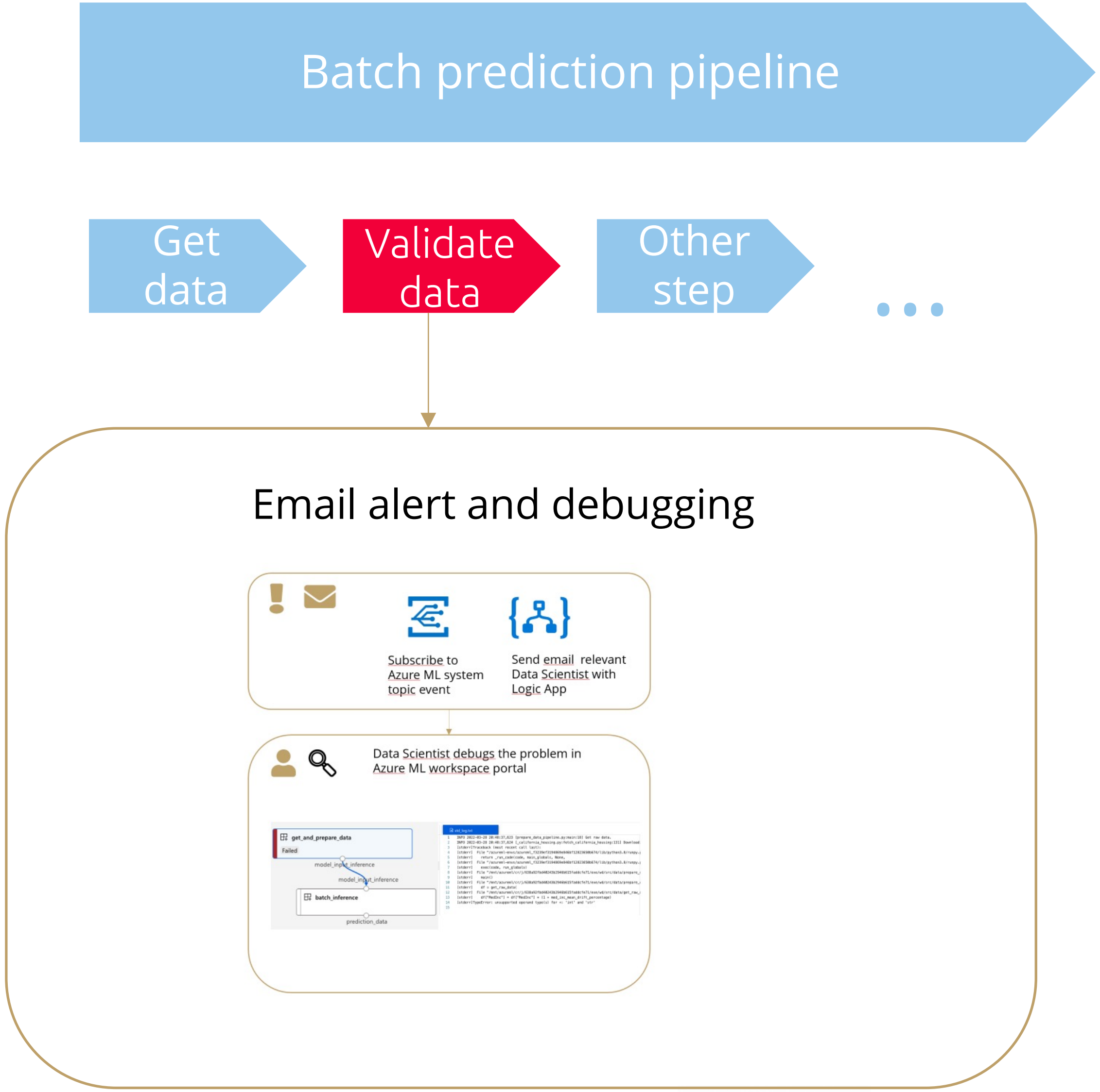
Monitoring

- Traditional monitoring:
  - Did the pipeline run fail?
  - What where the logs?
- Data monitoring
  - Runtime validation
  - Was the schema as expected?
  - Other expectations of the data: No missing values, over zero, etc.



Monitoring

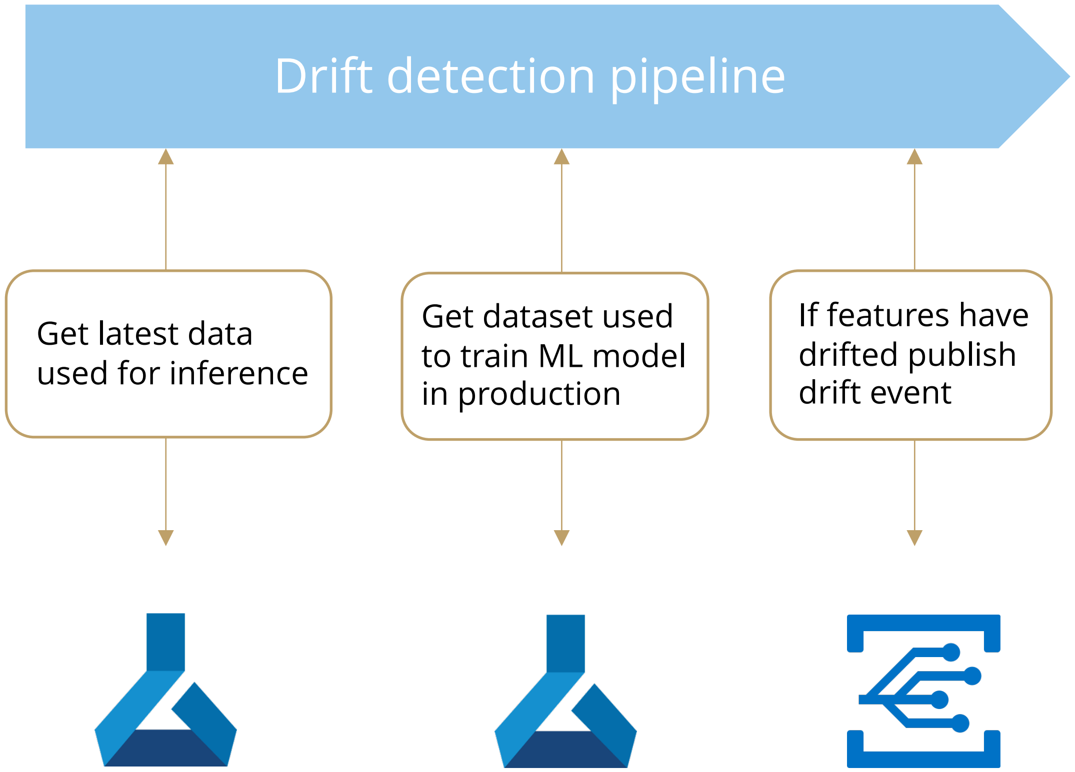
- Traditional monitoring:
  - Did the pipeline run fail?
  - What where the logs?
- Data monitoring
  - Runtime validation
  - Was the schema as expected?
  - Other expectations of the data: No missing values, over zero, etc.



NIVELOP

Monitoring

- Traditional monitoring:
  - Did the pipeline run fail?
  - What where the logs?
- Data monitoring
  - Was the schema as expected?
  - Other expectations of the data: No missing values, over zero, etc.
- ML specific monitoring
  - Did the performance of the model degrade?
  - Did the statistical distribution of the data change / drift?



ML pipelines

Use command line scripts for your pipeline steps

Develop outside Azure ML pipelines for a good developer experience

Deployment with Azure ML pipelines: Get all the goodies

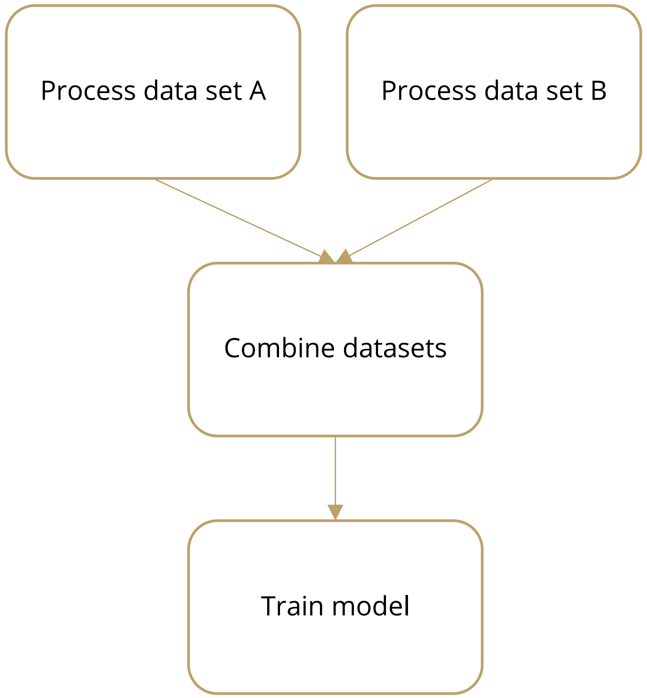
Why use a pipeline?

→ Modular design / separation of concerns

→ Inspect intermediate results

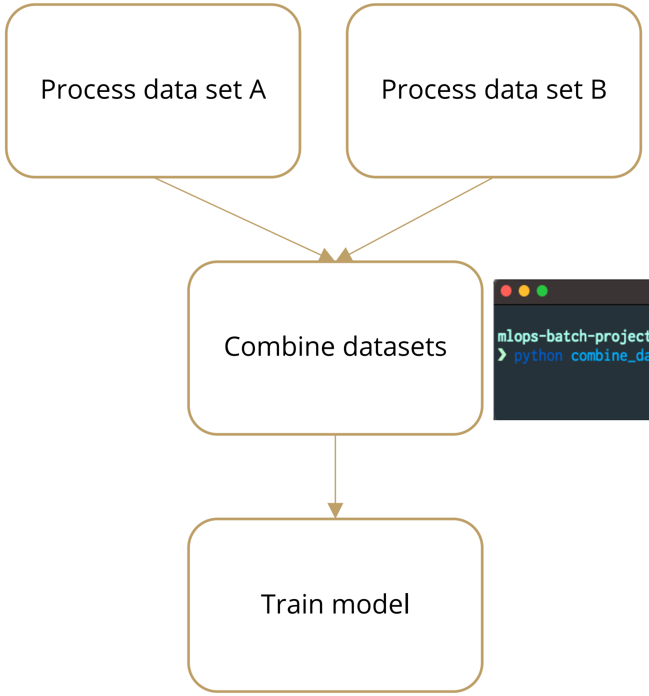
→ Rerun only part of pipeline

→ Run independent steps in parallel



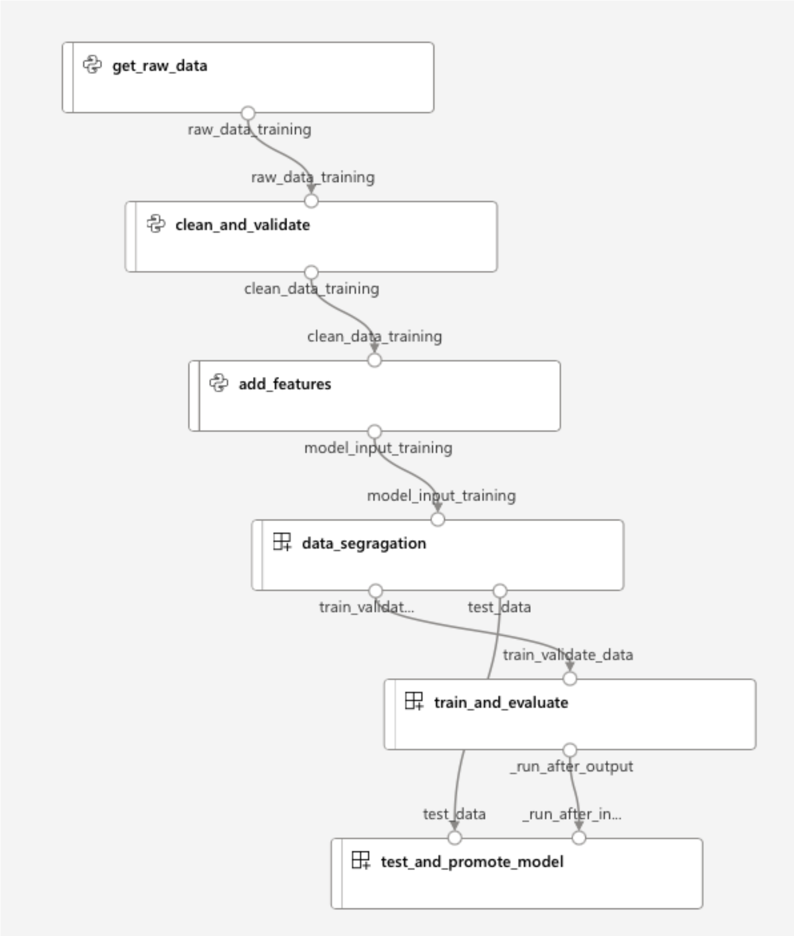
Why use command line scripts?

- → Universal interface
- → Can be orchestrated by a wide variety of tools
- → Easy to orchestrate locally, using something like Make when developing



Why run / orchestrate with AML pipelines?

- Highly scalable compute
  - Possibility of spark cluster
- Data lineage / versioning
- Versioning of pipelines
- Versioning of environments
  - Each step can have it's own environment
- Monitoring / great debugging
- Parallelize steps
- Multiple ways of triggering the prediction and training pipelines
  - API
  - Data Factory
  - Blob event / aka new data



Defining AML pipeline steps

→ Pipelines are defined and published using python

→ You wrap each step in the pipeline in code that define the data dependencies

```
################################################
# Clean and validate step
################################################
clean_training_data = OutputFileDatasetConfig(name='clean_data_training')
clean_training_data = clean_training_data.register_on_complete(name='clean_data_training')
raw_data_as_input = raw_training_data.as_input(name="raw_data_training")

clean_and_validate_step = PythonScriptStep(
    name="clean_and_validate",
    script_name="src/data/clean_and_validate.py",
    source_directory=".",
    arguments=[
        f"data.raw_data.folder={raw_data_as_input.arg_val}",
        f"data.clean_data.folder={clean_training_data.arg_val}",
    ],
    inputs=[raw_data_as_input],
    outputs=[clean_training_data],
    compute_target=compute_target,
    runconfig=aml_run_config,
    allow_reuse=True
)
```

NIVELOP

Demo