Copyright Notice

Machine Learning Operations (MLOps)

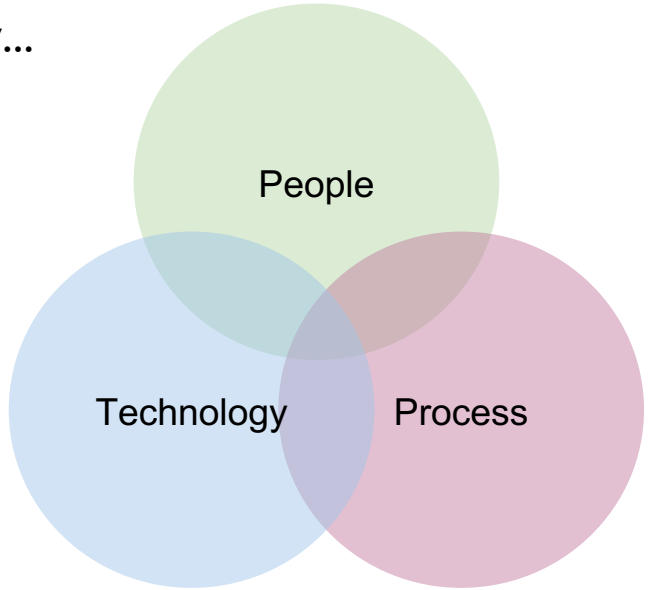
Overview

Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

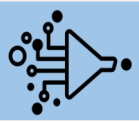
Path to production for ML models

It's not just **technology**...



Path to production for ML models

Considerations



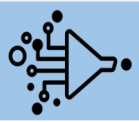
Machine Learning Development Lifecycle (MLDC)
!= Software Development Lifecycle (SDLC)

Model development cycles are not as easy to plan as the Software Development Cycles
It typically takes more time for a Model development cycle.

Further a Model development cycle typically include data tasks like feature engineering and data preparation, etc.
So, you have data preprocessing code and new inputs and artifacts for versioning.

Path to production for ML models

Considerations



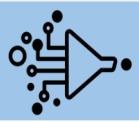
Machine Learning Development Lifecycle (MLDC)
!= Software Development Lifecycle (SDLC)

A Model may be a small part of an overall solution

You may be creating a rest API as a common interface for other applications to integrate with your model or even building applications that can respond to those reviews.
This means creating automation to initiate back-end processes to quickly respond to any negative reviews.

Path to production for ML models

Considerations



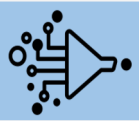
Machine Learning Development Lifecycle (MLDC)
!= Software Development Lifecycle (SDLC)

A Model may be a small part of an overall solution

Multiple personas spanning the MLDC

Path to production for ML models

Considerations



Machine Learning Development Lifecycle (MLDC)
!= Software Development Lifecycle (SDLC)

A Model may be a small part of an overall solution

Multiple personas spanning the MLDC

Integration with traditional IT practices

Manual approval is needed before a new model is deployed.

Path to production for ML models

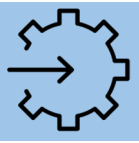
Challenges

Culture & Lack of cross functional teams

Path to production for ML models

Challenges

Culture & Lack of cross functional teams

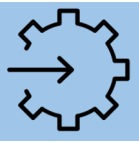


Integration into client applications & existing tooling

Path to production for ML models

Challenges

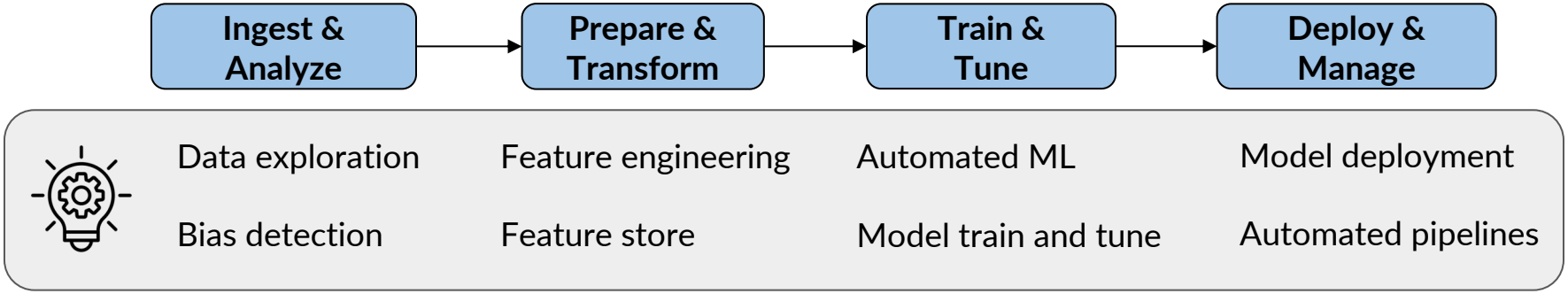
Culture & Lack of cross functional teams



Integration into client applications & existing tooling

Multiple disparate pipelines & dependencies (ex. Code, Data, Training)

Machine Learning Workflow



Operationalizing Machine Learning

Goals

Accelerate the path to production:

- ❑ Reduce manual hand-offs between steps
- ❑ Increase automation within steps
- ❑ Orchestrate the workflow

Operationalizing Machine Learning

Goals

Accelerate the path to production:

- ❑ Reduce manual hand-offs between steps
- ❑ Increase automation within steps
- ❑ Orchestrate the workflow

Improve the quality of deployed models:

- ❑ Implement automated workflows with quality gates

Operationalizing Machine Learning

Goals

Accelerate the path to production:

- ❏ Reduce manual hand-offs between steps
- ❏ Increase automation within steps
- ❏ Orchestrate the workflow

Improve the quality of deployed models:

- ❏ Implement automated workflows with quality gates

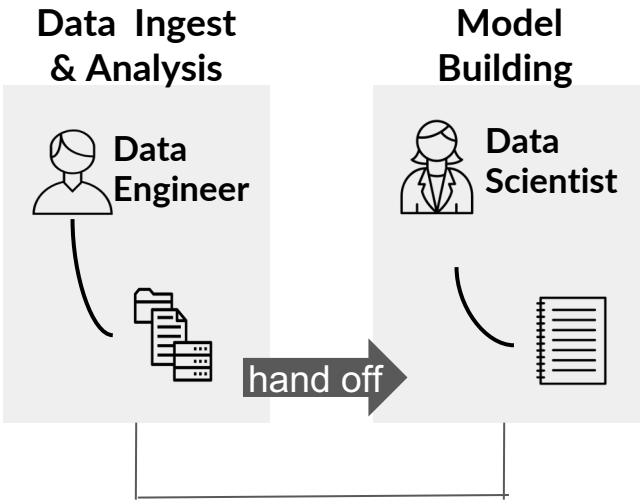
Build resilient, secure, performant, operationally efficient and cost optimized ML solutions

- ❏ Consider aspects unique to ML solutions + Traditional systems engineering considerations

Operationalizing Machine Learning

Path to production

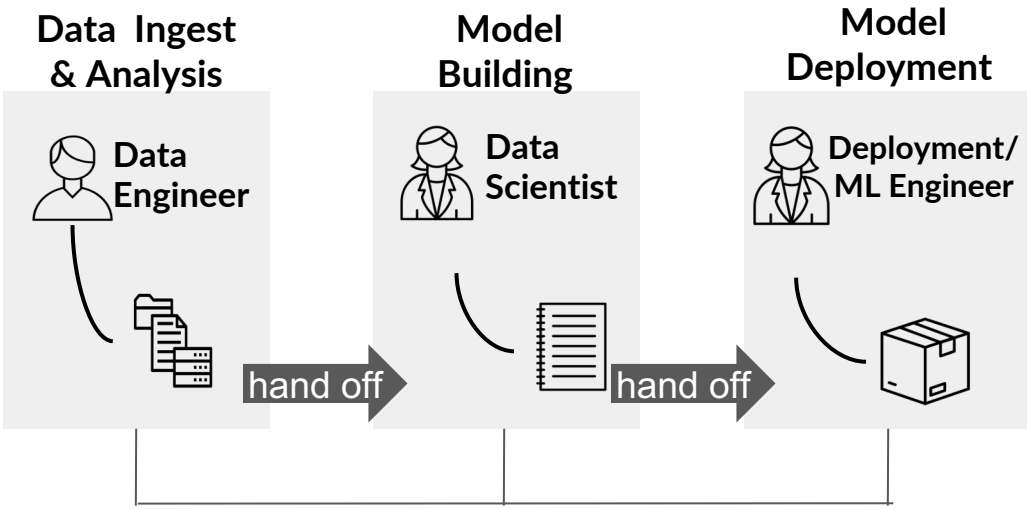
Example workflow with multiple hand offs:



Operationalizing Machine Learning

Path to production

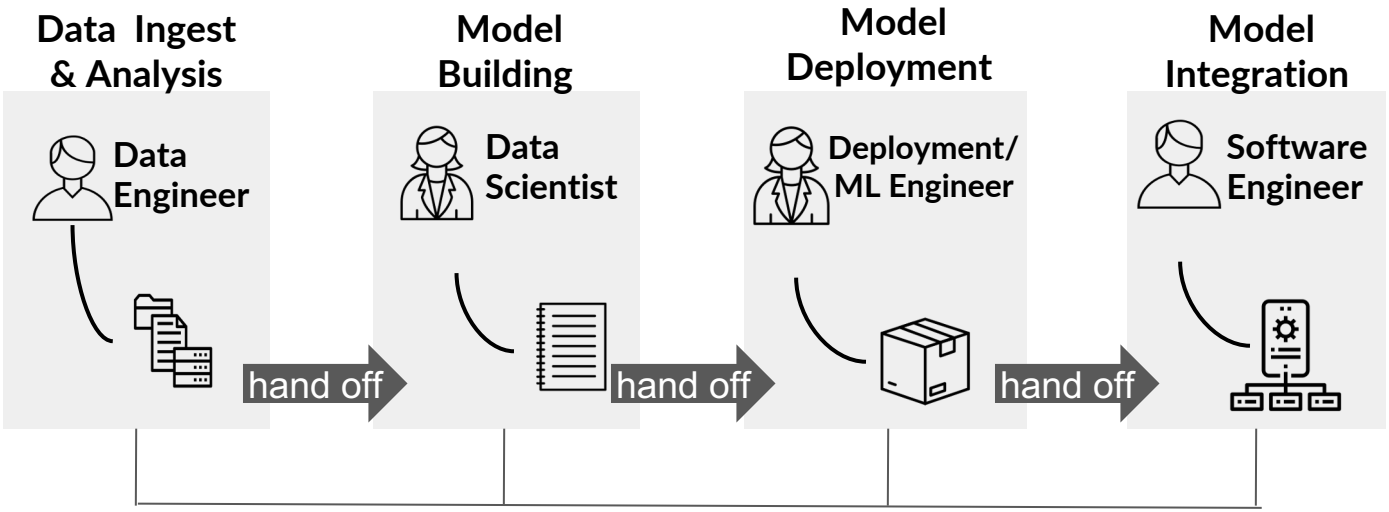
Example workflow with multiple hand offs:



Operationalizing Machine Learning

Path to production

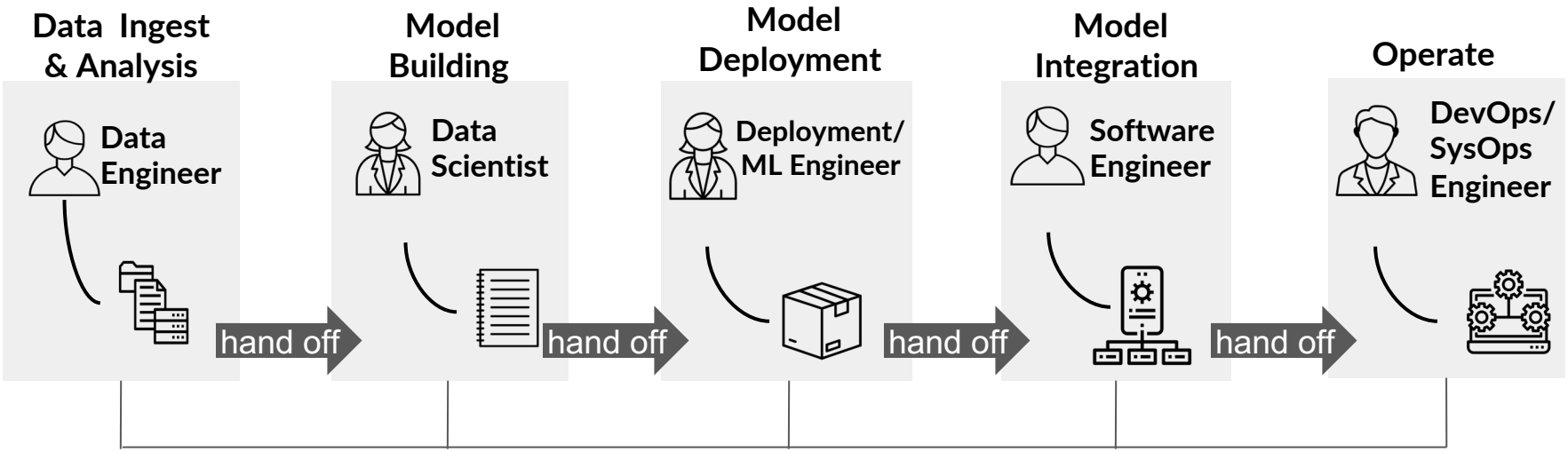
Example workflow with multiple hand offs:



Operationalizing Machine Learning

Path to production

Example workflow with multiple hand offs:

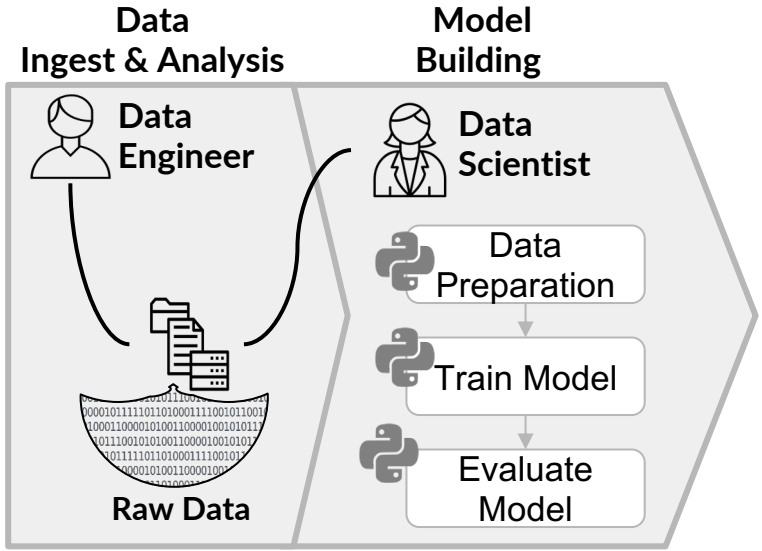


- Multiple handoffs
- Increased rework
- Limited Visibility & Transparency

Operationalizing Machine Learning

Accelerate the path to production

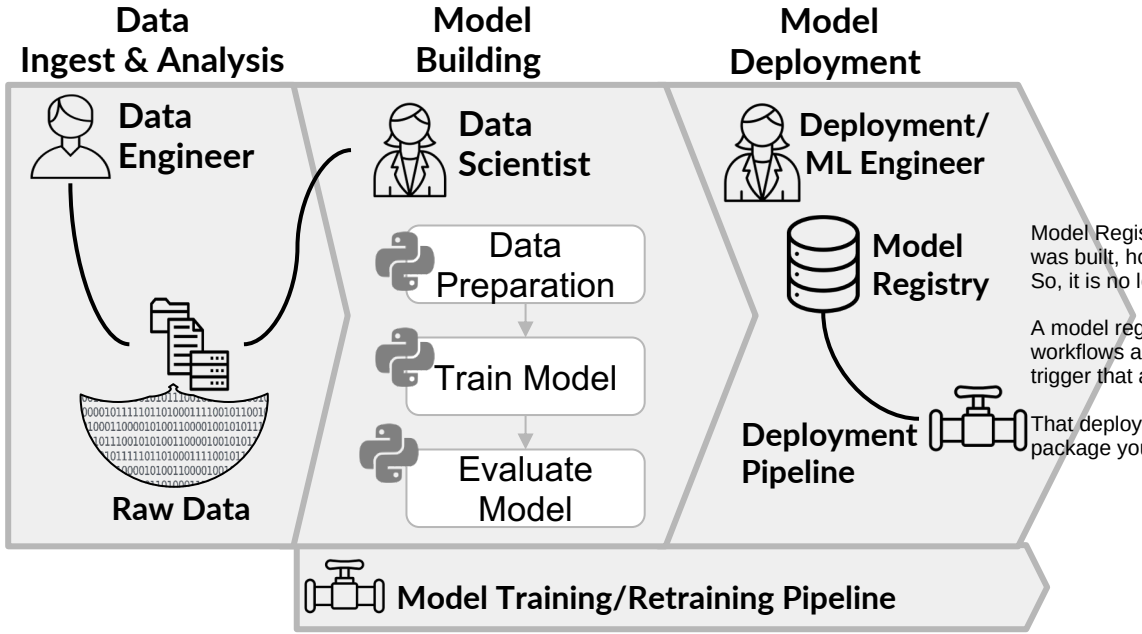
Automating the tasks within a workflow step



Operationalizing Machine Learning

Accelerate the path to production

Automating the tasks within a workflow step



Model Registry holds key metadata and information about how your model was built, how the model performed with evaluation metrics.
So, it is no longer a manual hand-off of a black box model.

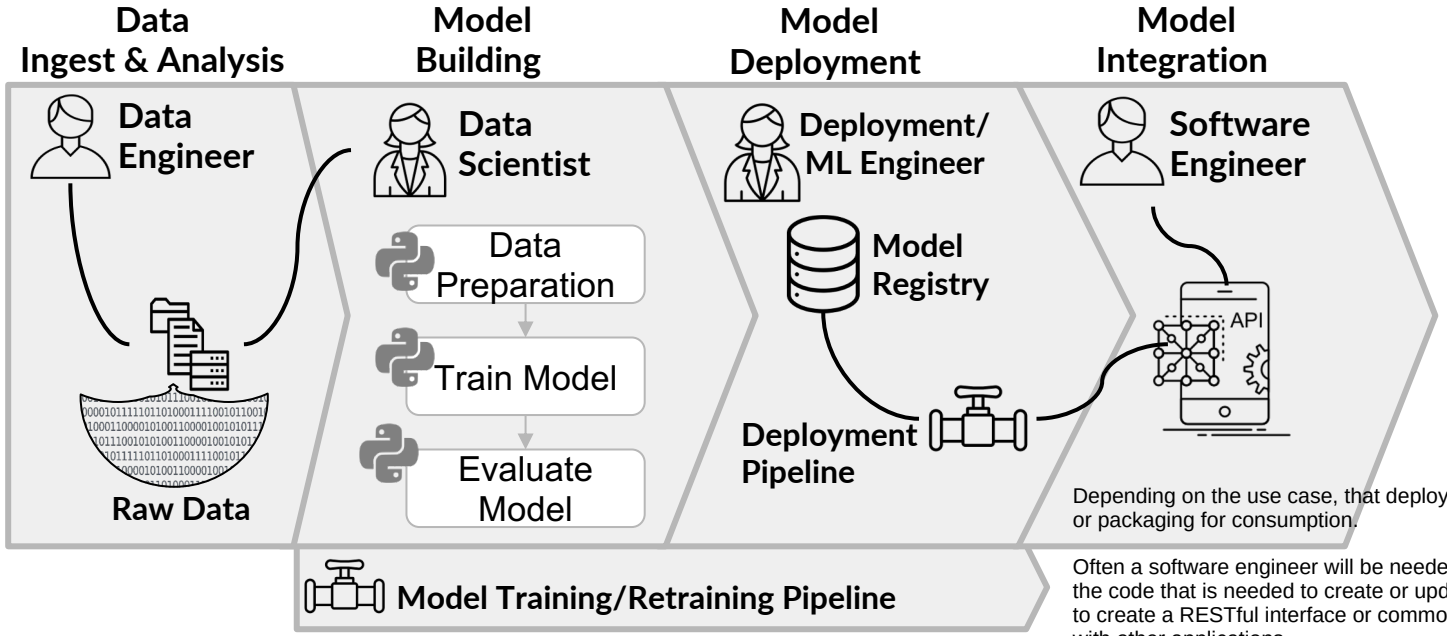
A model registry can also be used to trigger downstream deployment workflows as well. So, once the model is registered and approved it can then trigger that automated downstream pipeline.

That deployment pipeline typically includes the steps that are needed to package your model and then deploy it to one or more environments.

Operationalizing Machine Learning

Accelerate the path to production

Automating the tasks within a workflow step



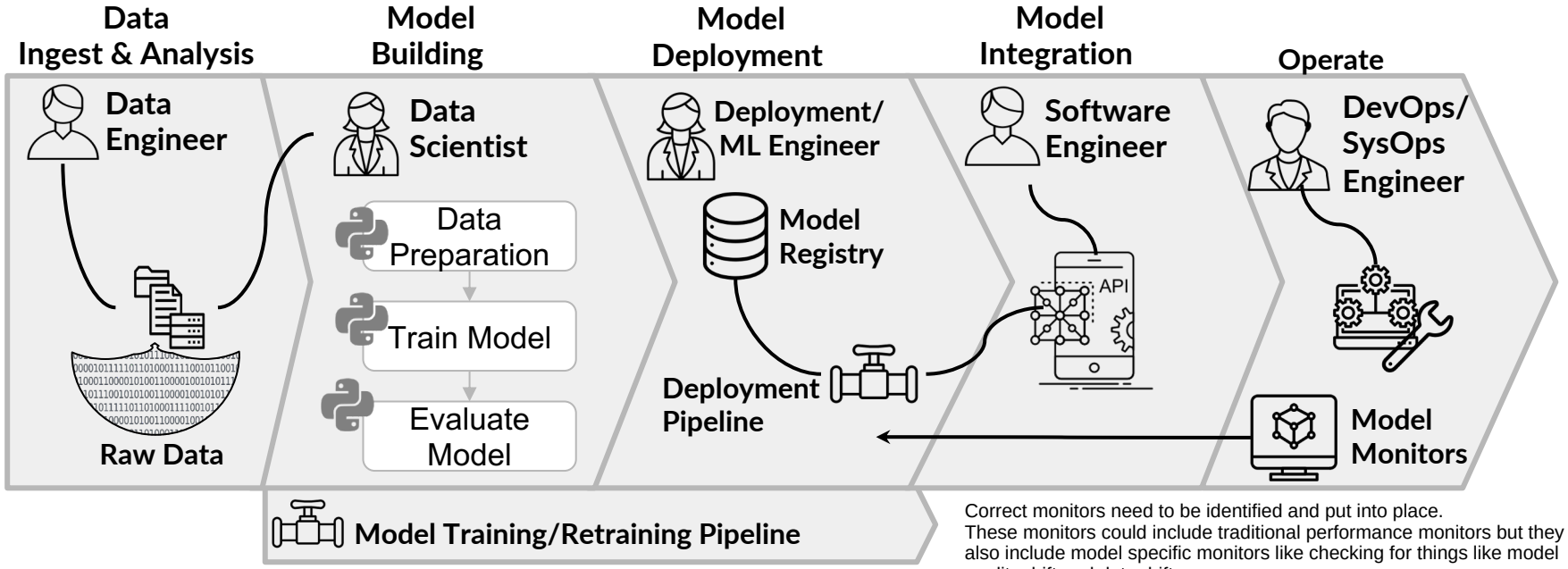
Depending on the use case, that deployment may require additional code or packaging for consumption.

Often a software engineer will be needed here to provide the steps and the code that is needed to create or update the API that will then be used to create a RESTful interface or commonly defined interface for integrating with other applications.

Operationalizing Machine Learning

Accelerate the path to production

Automating the tasks within a workflow step



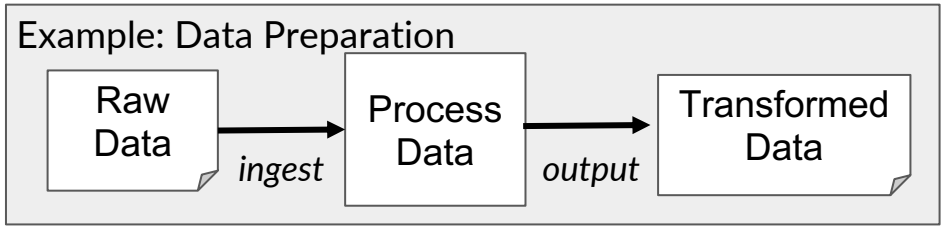
Correct monitors need to be identified and put into place.
These monitors could include traditional performance monitors but they also include model specific monitors like checking for things like model quality drift and data drift.

Operationalizing Machine Learning

Accelerate the path to production

Automation vs Orchestration

Automation: Automate a **task** (Ex. Data Preparation) to perform a specific activity or produce defined artifacts based on the inputs or triggers of that task without human intervention



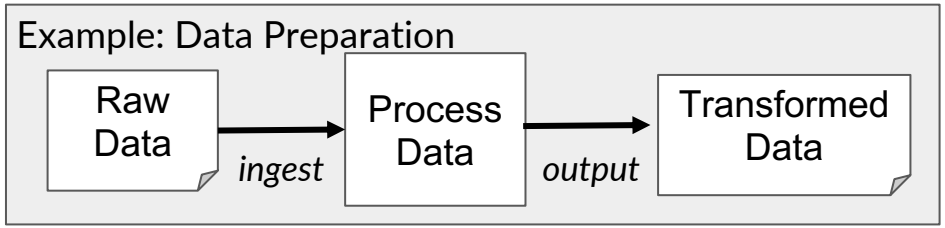
Operationalizing Machine Learning

Accelerate the path to production

Automation vs Orchestration

Automation: Automate a **task** (Ex. Data Preparation) to perform a specific activity or produce defined artifacts based on the inputs or triggers of that task without human intervention

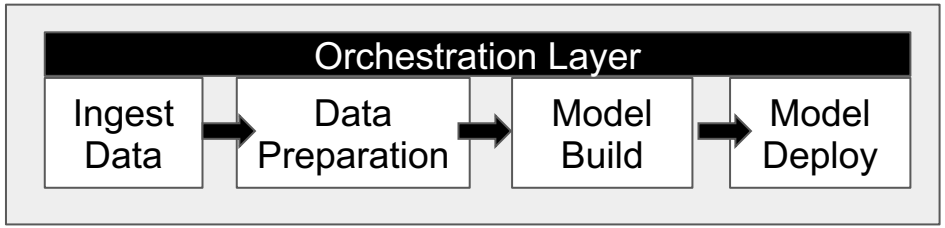
Automation is great for reducing the cycle time and deploying more quickly.



Orchestration: Orchestrate the steps of a workflow that contain a **collection of tasks**

To automate the end-to-end Machine Learning workflow you also need to add a layer that can provide overall orchestration, in defining when and how these individual steps with automated tasks are run.

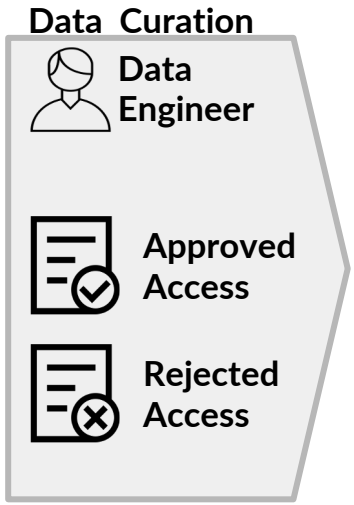
Orchestration is need for improving and ensuring the quality of your model.



Operationalizing Machine Learning

Improve the quality of deployed models

Examples: Automated quality gates

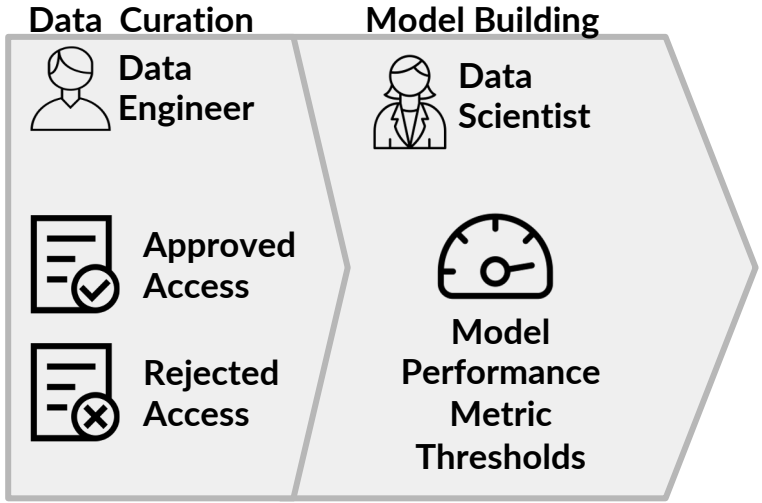


Automated or manual check-points with in your pipeline that gates whether the next step in your pipeline can run based on some type of criteria or conditional step that you have defined.
e.g.
- Data Curation: that governs and restricts access to the data that can be used
- Model Building: you are typically setting up a minimum threshold for your model evaluation matrix. This typically means that you are defining a matrix that you are trying to optimize for. Such as, accuracy or F1 score
- Model Deployment: you can have something like A/B Testing Metrics. So, for a portion of the model you run Model A and for the other portion you use Model B. Then you can evaluate whether Model A or B is working better.
- Model Integration: you want to make sure that the application that's consuming your model is able to get prediction requests bag. In these tests you're often making sure to check for things like your inference code is synchronized with your model and potentially the API.
- Model Monitoring: you want to monitor for standard metrics, so things like CPU, GPU, or memory utilization. However, you also want to set up those monitors that are monitoring specifically for your model. Things like data drift, which can indicate that the data that you used to train your model now looks much different than the actual ground truth data. In this case, you want to se-up a retraining pipeline.

Operationalizing Machine Learning

Improve the quality of deployed models

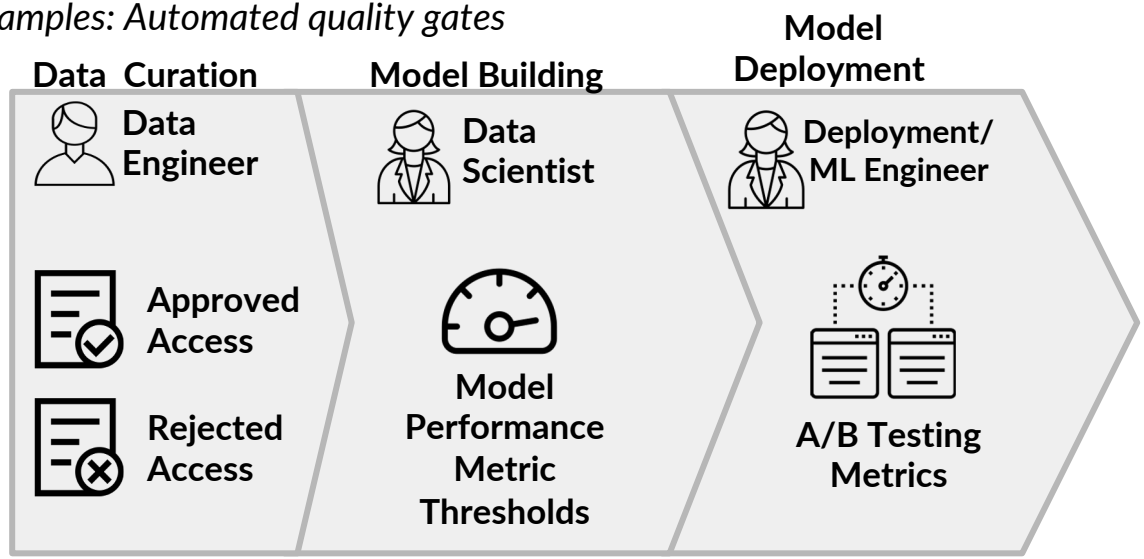
Examples: Automated quality gates



Operationalizing Machine Learning

Improve the quality of deployed models

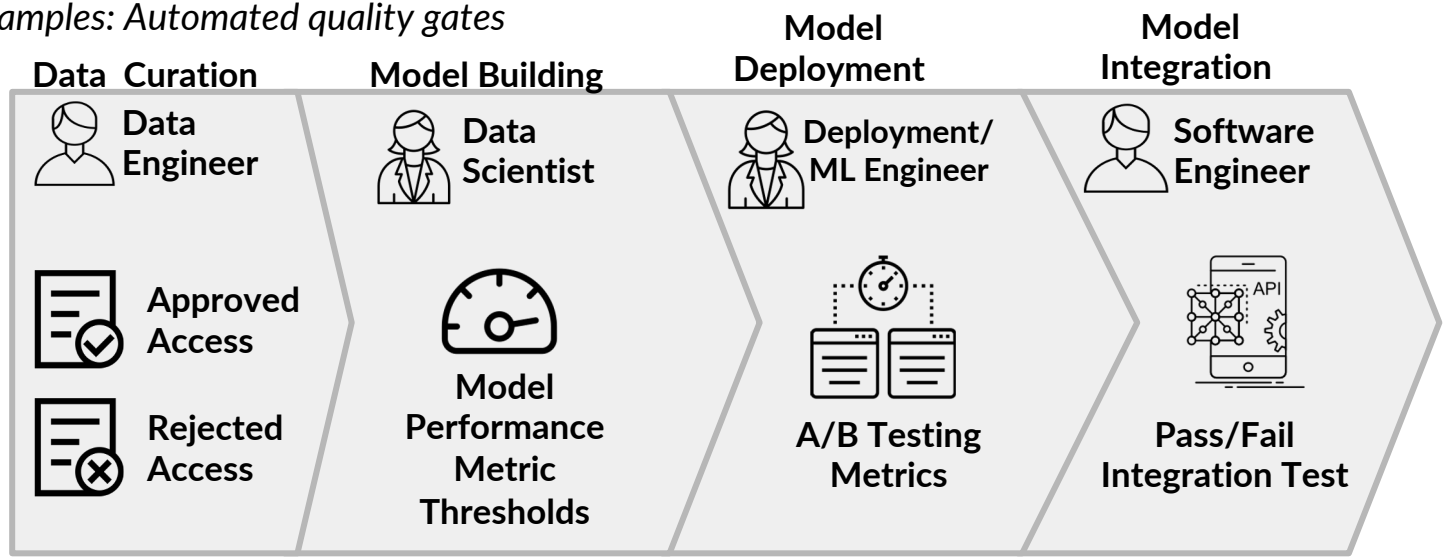
Examples: Automated quality gates



Operationalizing Machine Learning

Improve the quality of deployed models

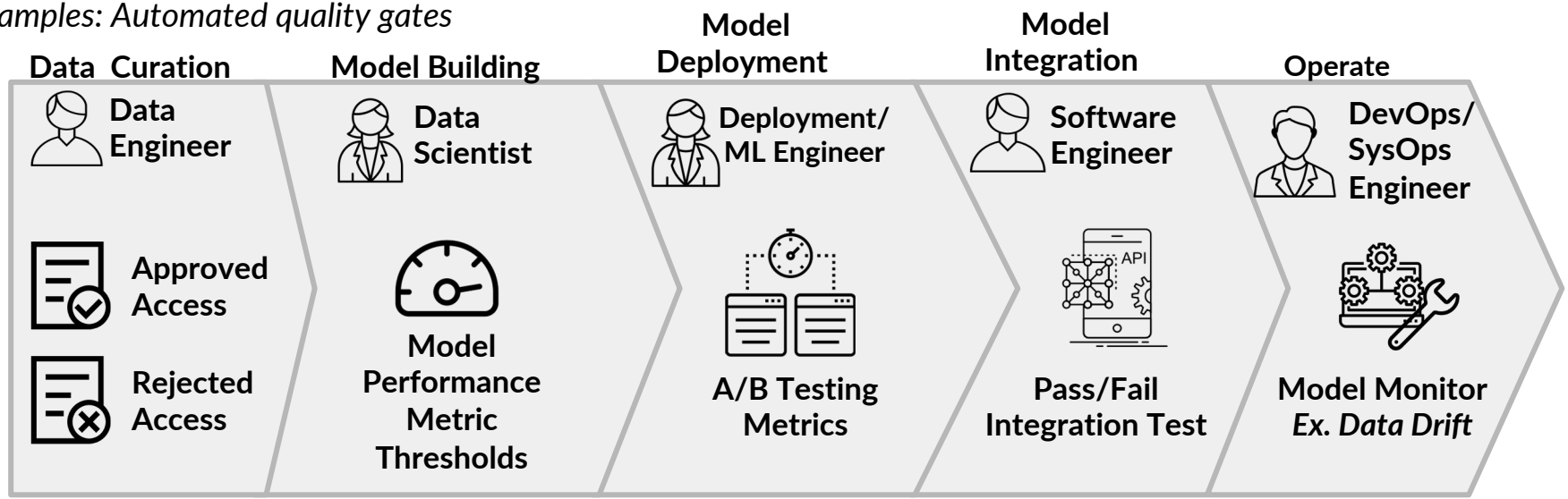
Examples: Automated quality gates



Operationalizing Machine Learning

Improve the quality of deployed models

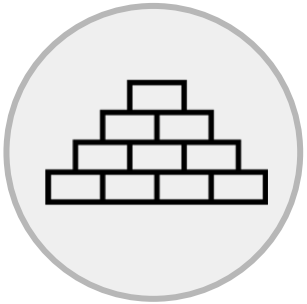
Examples: Automated quality gates



Operationalizing Machine Learning

Key Considerations

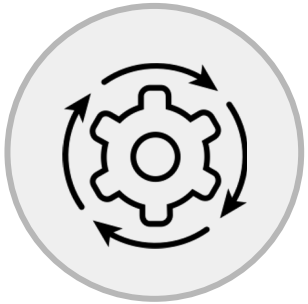
Security



Reliability

Cost Optimization

Performance Efficiency



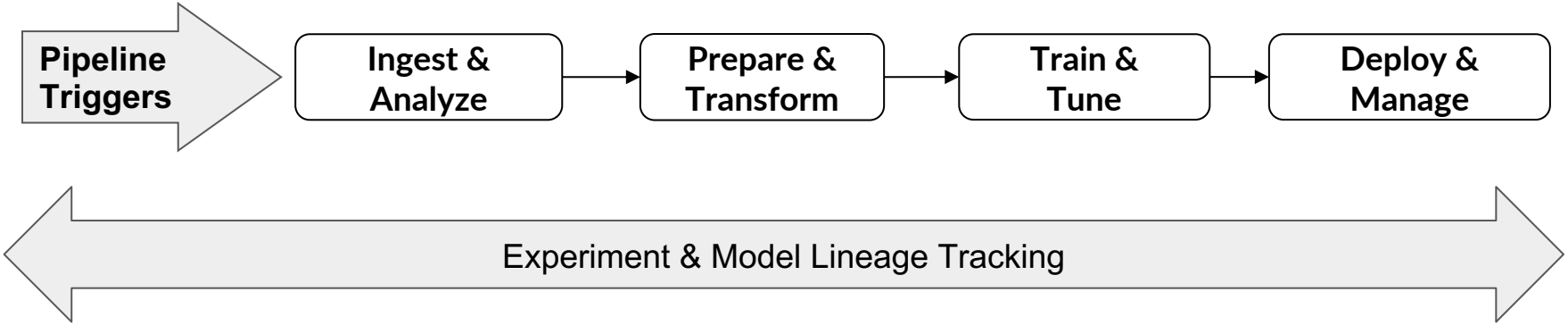
Operational Excellence

Creating Machine Learning Pipelines

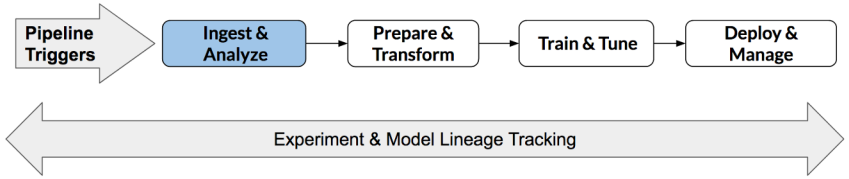
Creating Machine Learning Pipelines

Building Effective Pipelines

In the case of software development life cycles, it's usually pretty clear when you have a commit to a source code repository, you're going to automatically start an automated bill. However in the case of a machine learning pipeline you have multiple potential triggers. Such as a change to algorithm, hyper parameters or code, or alternatively having new training data, which could also be another potential trigger for your machine learning pipeline.

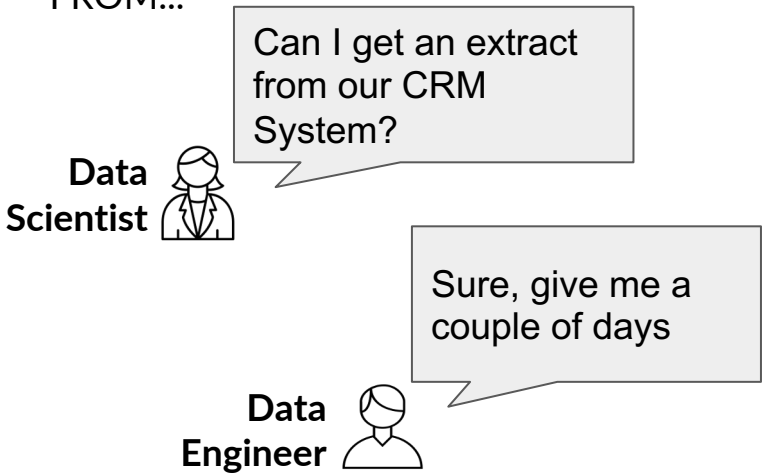


Data Tasks



Data Ingestion for Model Development

FROM...

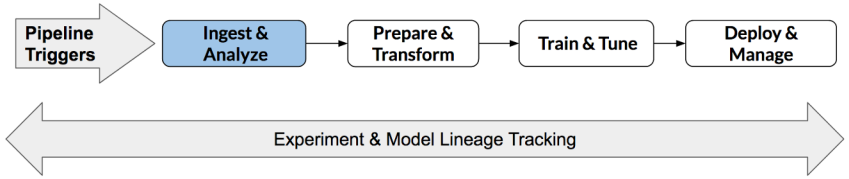


In Traditional approach you reach out to your data engineer, you ask them for some data, sometimes you need to go through a series of security approvals as well. And then ultimately get a data set that you can use for your model building activities.

Challenges with this model as you can imagine:
- First, it can really slow down model development time.
- Second, it can also result in limited traceability with these manual handoffs of data.
- Third, it also makes model development difficult in terms of automating any kind of retraining workflows.

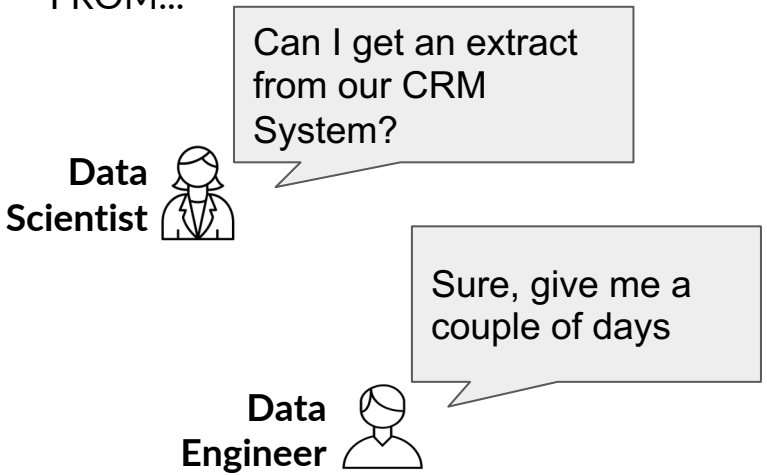
While you may be able to do some of those initial model build activities, retraining pipelines in this particular context are very difficult to support. It's generally recommended to establish a data lake that provides governed access to data scientists or automated processes. This allows data scientists to quickly start a model development activities, and it also ensures there's traceability for that data because you know which data scientists have checked out specific data sets.

Data Tasks

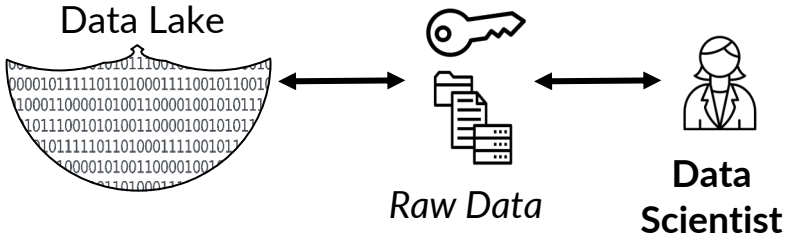


Data Ingestion for Model Development

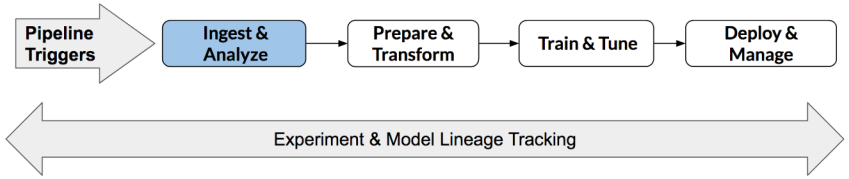
FROM...



TO...



Data Tasks



Data Ingestion for Model Retraining

FROM...

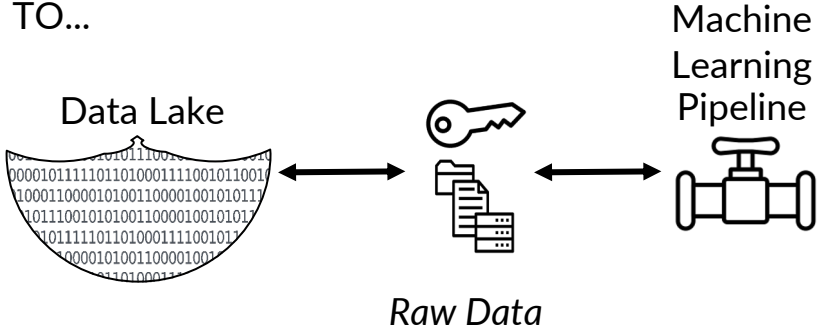
Can I get an extract from our CRM System?

Data Scientist

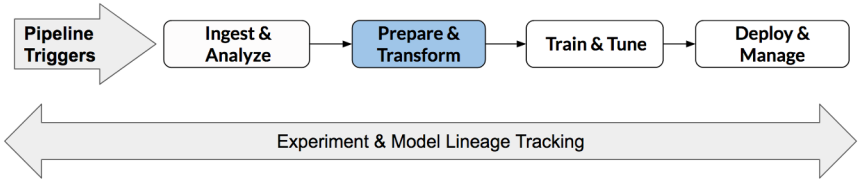
Sure, give me a couple of days

Data Engineer

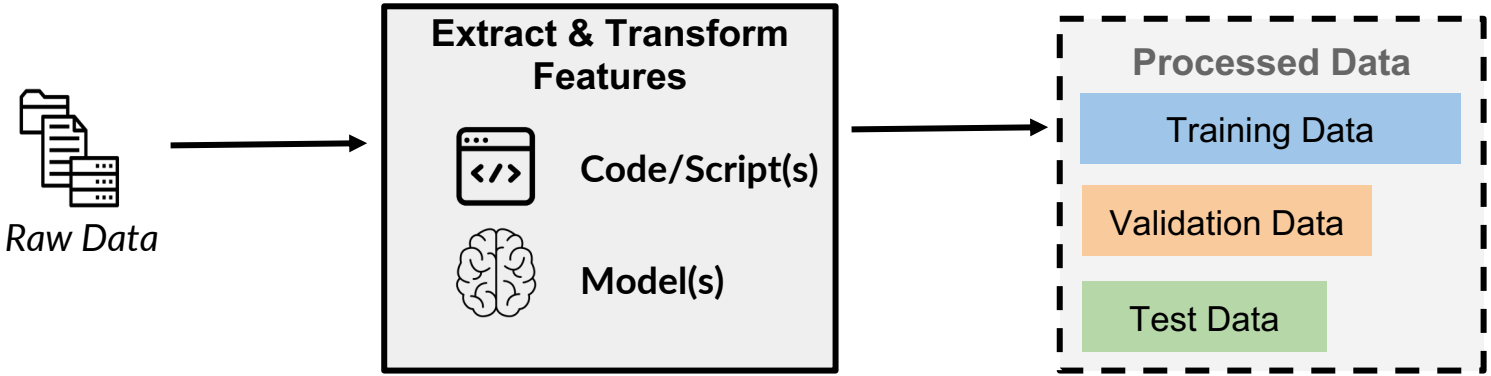
TO...



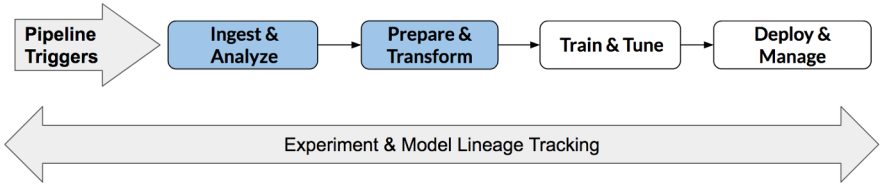
Data Tasks



Data Pre-Processing & Feature Engineering



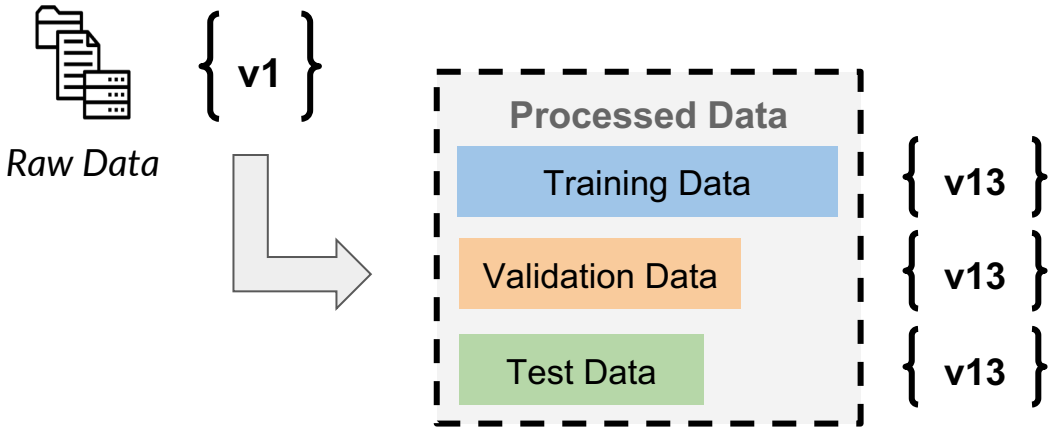
Data Tasks



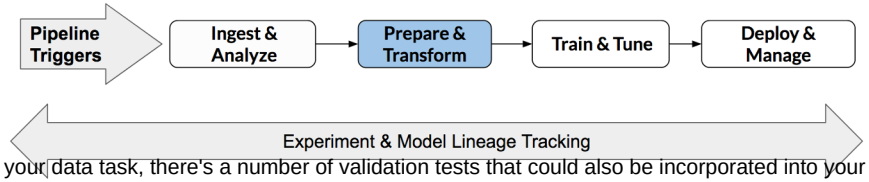
Data Versioning

When you automate your data processing and feature engineering, you have a key input and that's your raw data. And this typically gets fed in through automation, and then that automation will extract and transform those features, ultimately producing artifacts that will be used in training. The artifacts that are produced in your data preparation step include your training, validation, and test data sets.

Examples:



Data Tasks



Data Validation

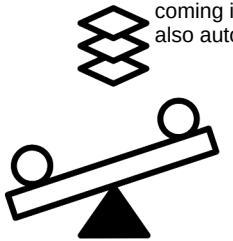
Examples:

Finally for your data task, there's a number of validation tests that could also be incorporated into your machine learning pipeline and automated to improve quality.

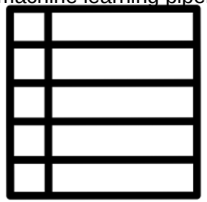
A few examples here include data quality checks where you can implement automated steps that gather statistics about your input data to look for any signals of potential data quality issues. So things like a specific attribute having an unexpectedly large number of missing values on input. Another example here would be checking for indicators of statistical bias.

Finally data schema is something that you can include in your automated checks as well. So in this case you can embed a quality check inside your pipeline that ensures that the data structure and schema that's coming in on input, is in the format is expected to be. The key here is not only performing these tasks but also automating them as part of your machine learning pipeline.

Data Quality

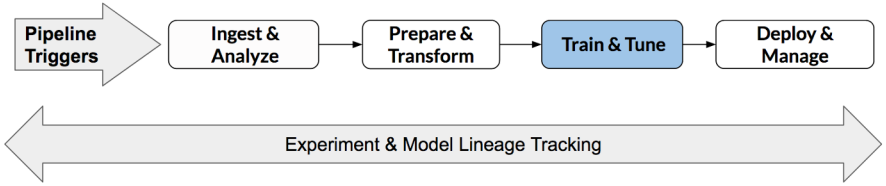


Statistical Bias

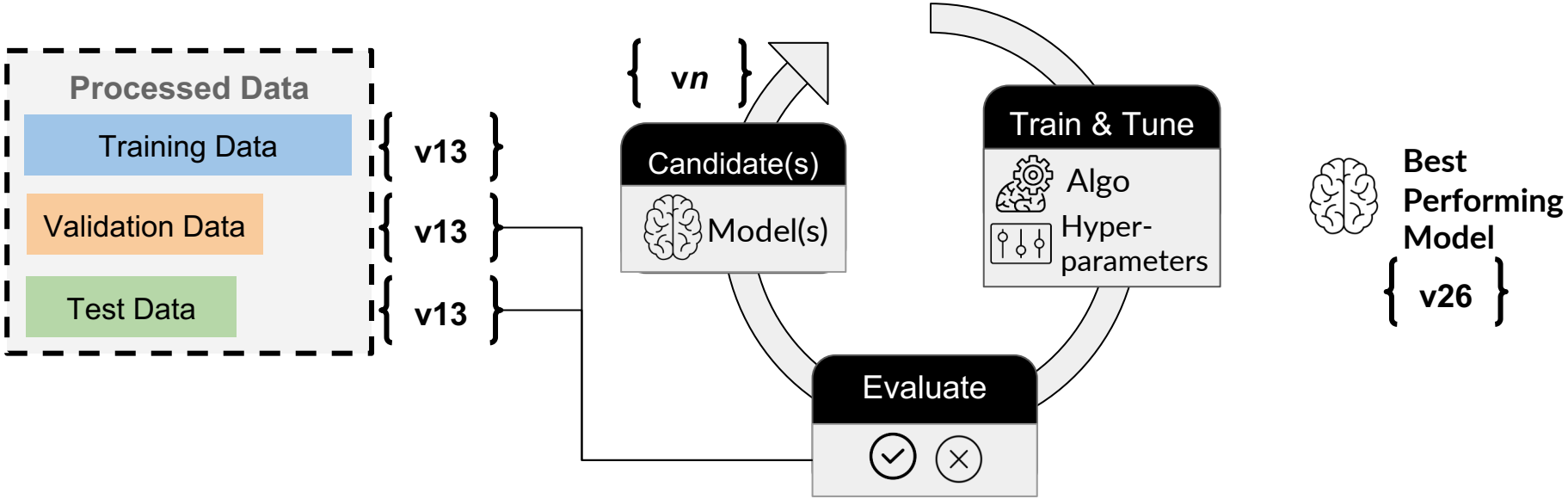


Data Schema

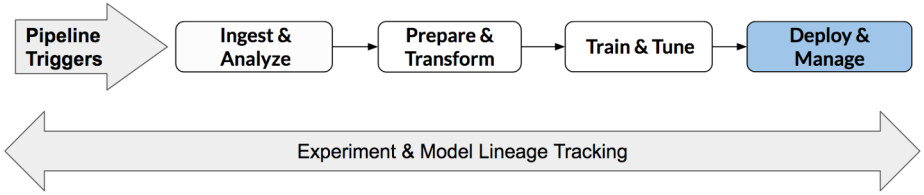
Model Building Tasks



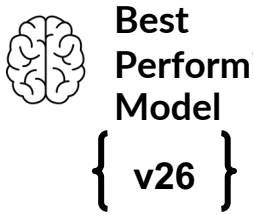
Model Training, Evaluation & Versioning

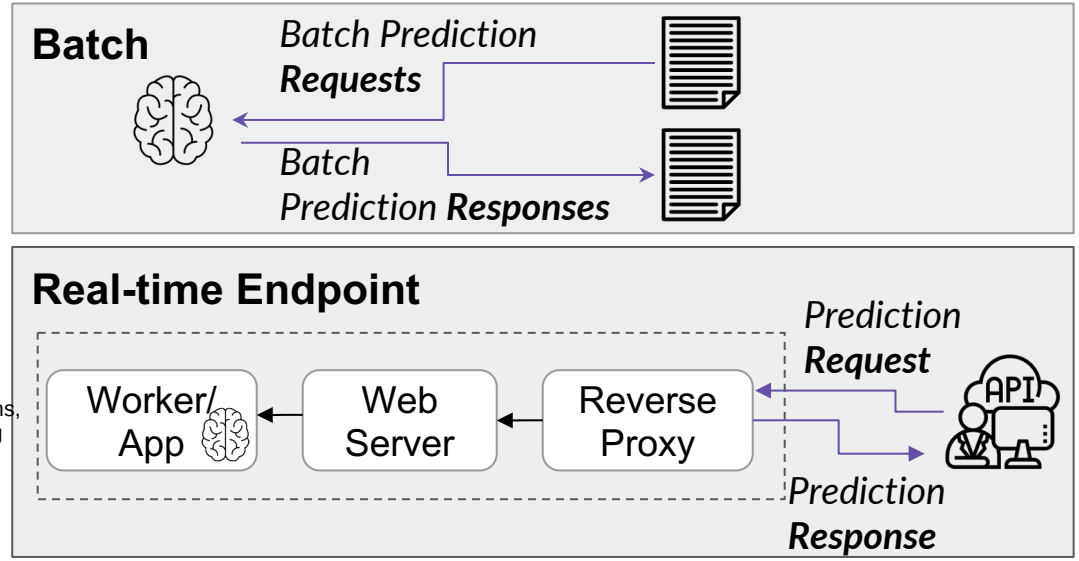


Model Deployment Tasks



Model Deployment & Consumption

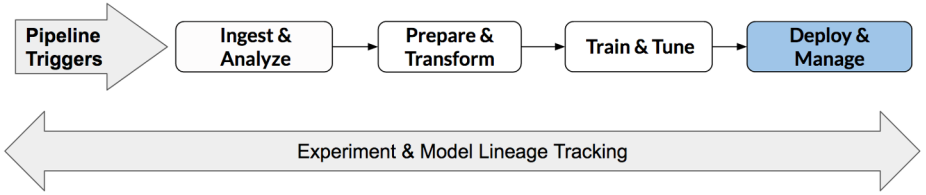




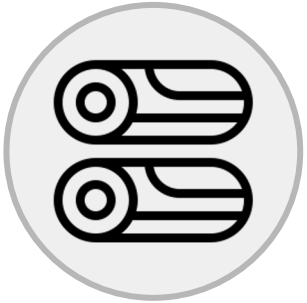
For a model deployment task, you're taking that model artifact and you're deploying it for consumption. This can take two different forms, you can either deploy in batch mode where you're basically wanting to send in batch records for prediction, and then receive batch responses back.

Or you can deploy your model for a real-time or persistent endpont. An endpoint can consistently serve prediction requests and responses through a serving stack. And the serving stack typically includes a proxy, a web server, that can then accept and respond to your requests coming in on input.

Operating Tasks



Logging & Monitoring



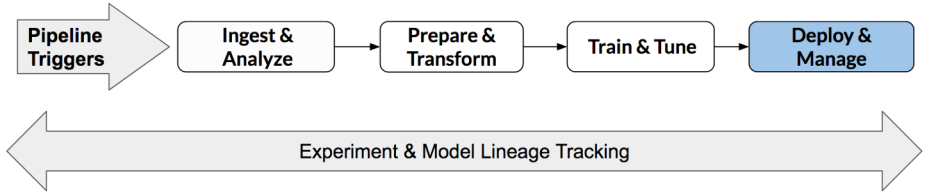
Logging →

- Model Data
- System Data

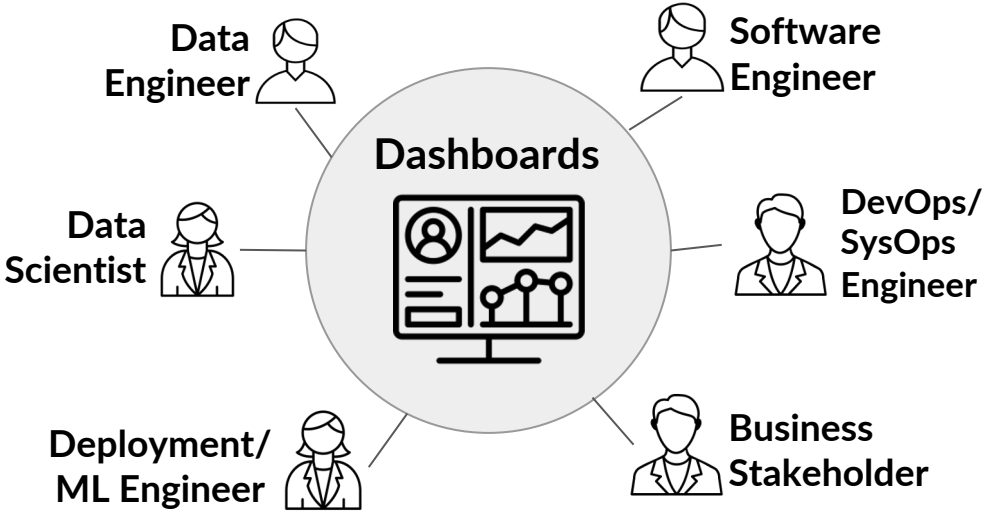
Monitoring →

- Collect Metrics
- Setup Alerts
- Trigger Automated Flows

Operating Tasks



Additional Feedback Mechanisms

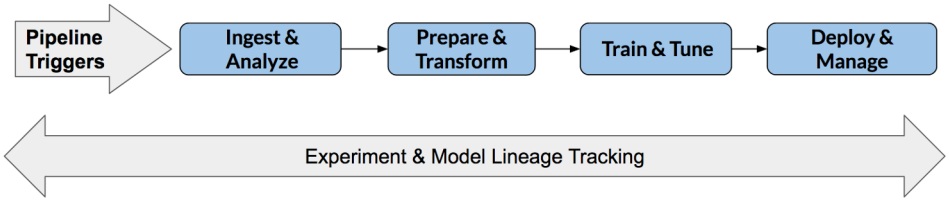


- Each persona can have different motivations and needs for monitors, logging and dashboards.
- Examples:
  - Pipeline Status
  - System Performance
  - Model Performance

Machine Learning Pipelines

Pipeline Orchestration: Bringing It Together

- ❑ Steps within **Task** can be automated
- ❑ Each set of tasks has **Inputs & Artifacts** produced as part of those steps



- ❑ **Orchestration** is required to coordinate the execution of tasks and steps within the tasks.

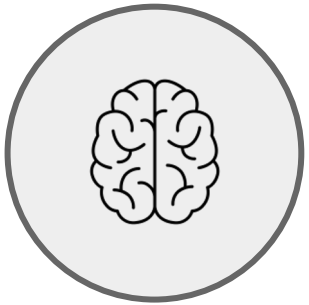
Model Lineage & Artifact Tracking

Model Lineage

What is Model Lineage?

Model lineage essentially refers to understanding and tracking all of the inputs that were used to create a specific version of a model. There are typically many inputs that go into training a specific version of a model.

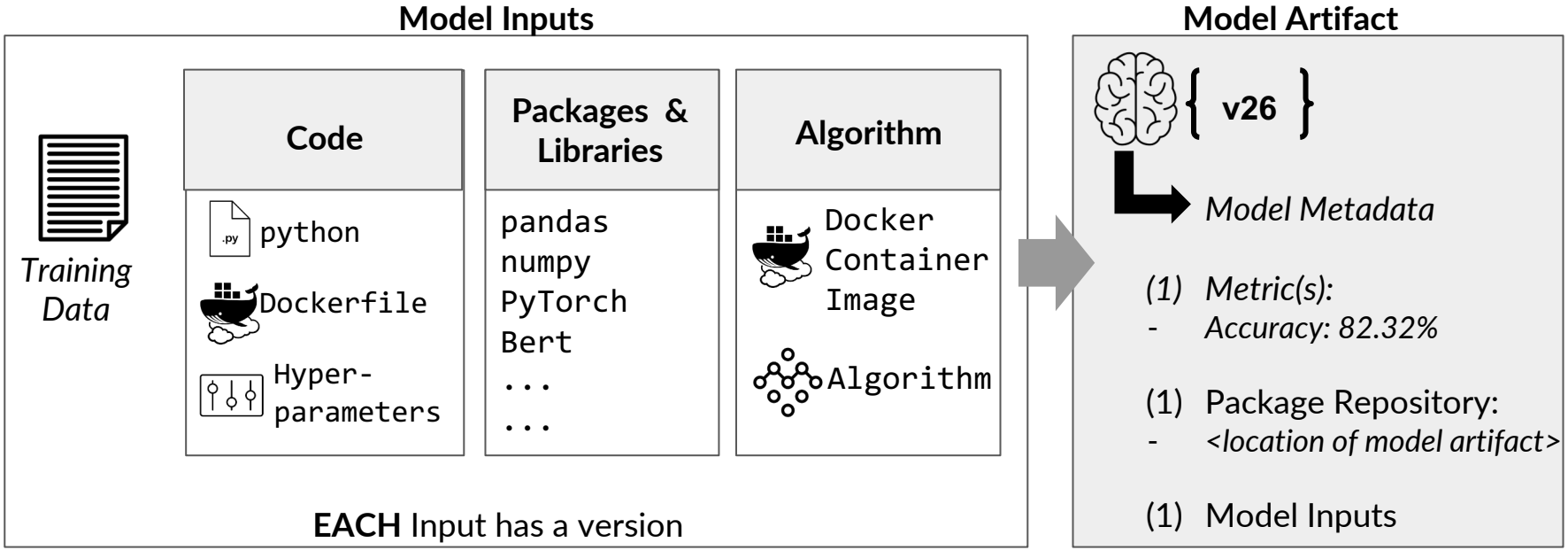
For **EACH** version of a trained model:



- ❑ Version(s) of data used
- ❑ Version(s) of code/hyperparameters used
- ❑ Version(s) of algorithm/framework
- ❑ Version(s) of training docker image
- ❑ Version(s) of packages/libraries

Model Lineage

Model Lineage Example

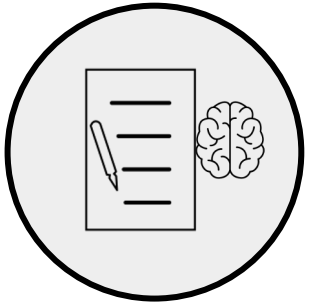


Model Lineage

Model Registry

A model registry is a central store for managing model metadata and model artifacts. When you incorporate a model registry into your automated pipeline, it can provide traceability and auditability for your models, allowing you to effectively manage models, especially when you begin to manage at scale and you have tens or hundreds or even thousands of models.

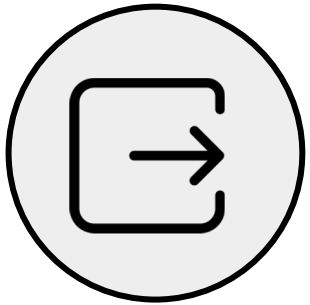
What is a Model Registry?



- ❏ Centrally manage model metadata and model artifacts
- ❏ Track which models are deploy across environments

Artifact Tracking

What is Artifact Tracking?

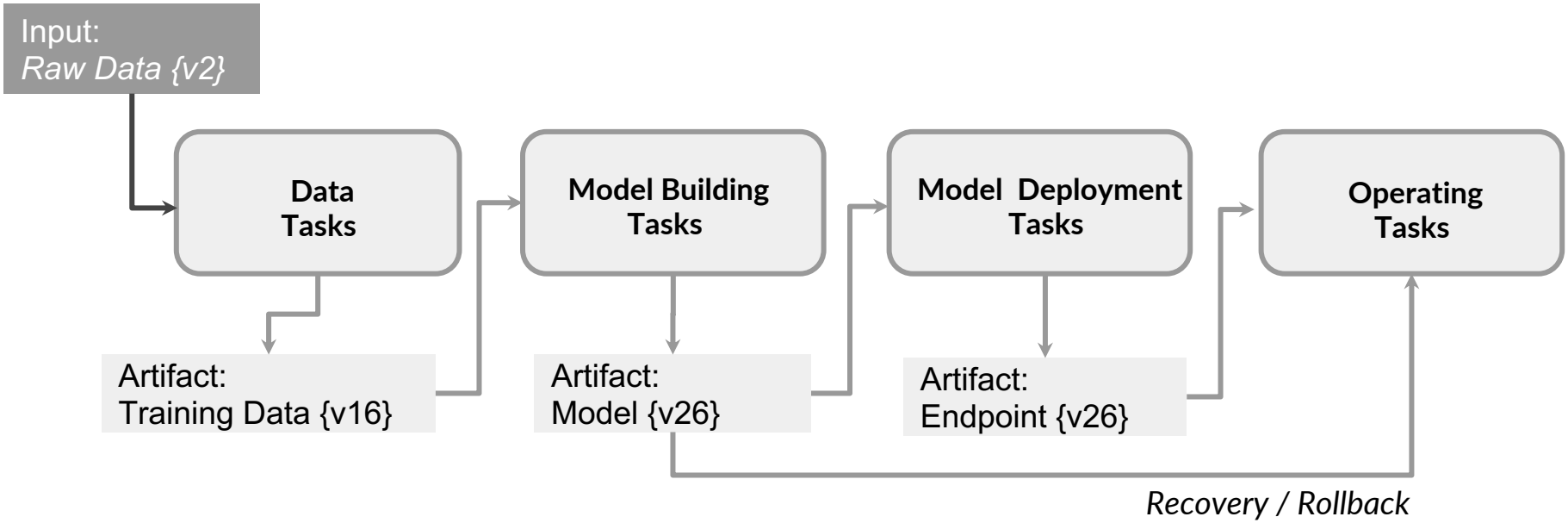


- An **Artifact** is the output of a step or task can be consumed by the next step in a pipeline or deployed directly for consumption

Artifact Tracking

Pipeline Manifest

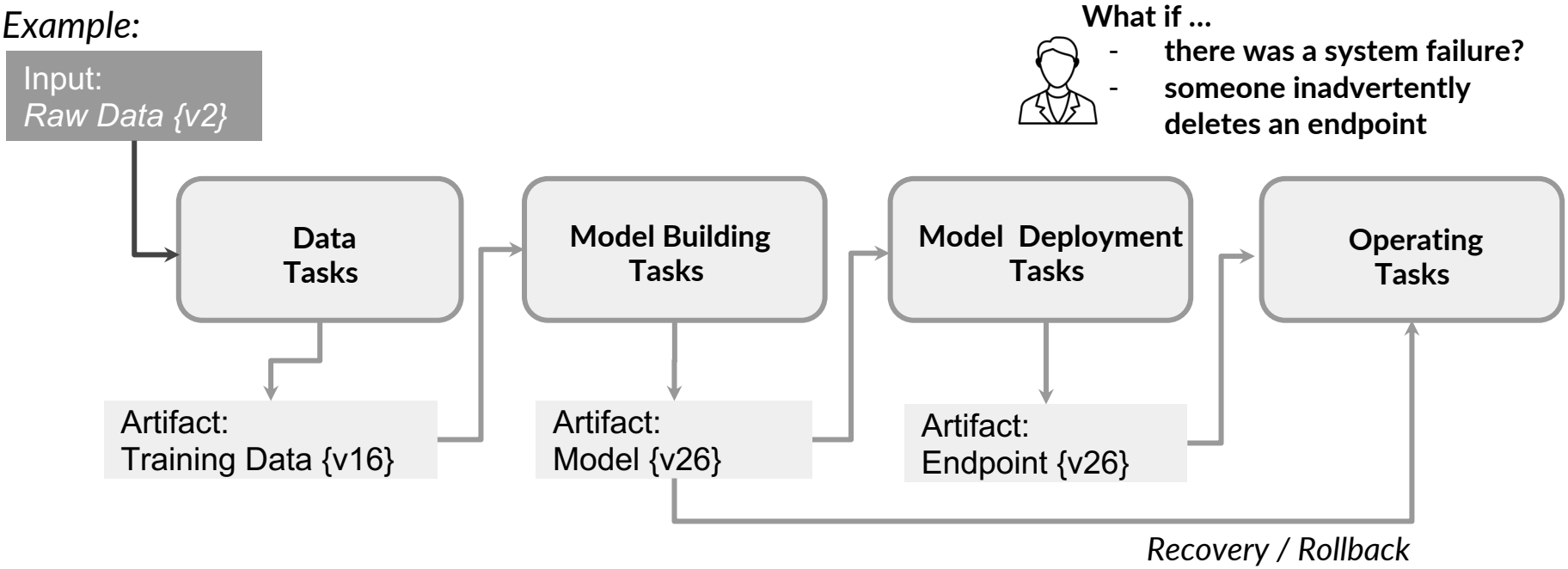
Example:



Artifact Tracking

Pipeline Manifest - Why it matters

Example:



What if ...

- **there was a system failure?**
- **someone inadvertently deletes an endpoint**

Machine Learning Pipelines

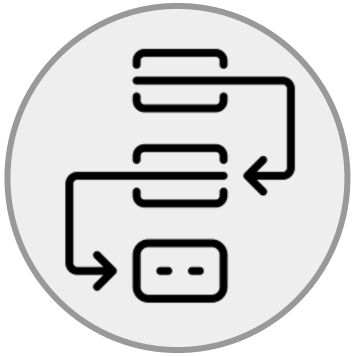
with
Amazon SageMaker Pipelines

Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines

Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

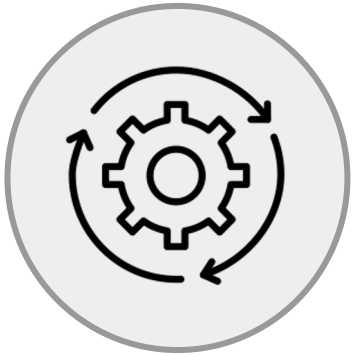
Amazon SageMaker Pipelines



Create & visualize automated workflows

Choose the best performing model to deploy

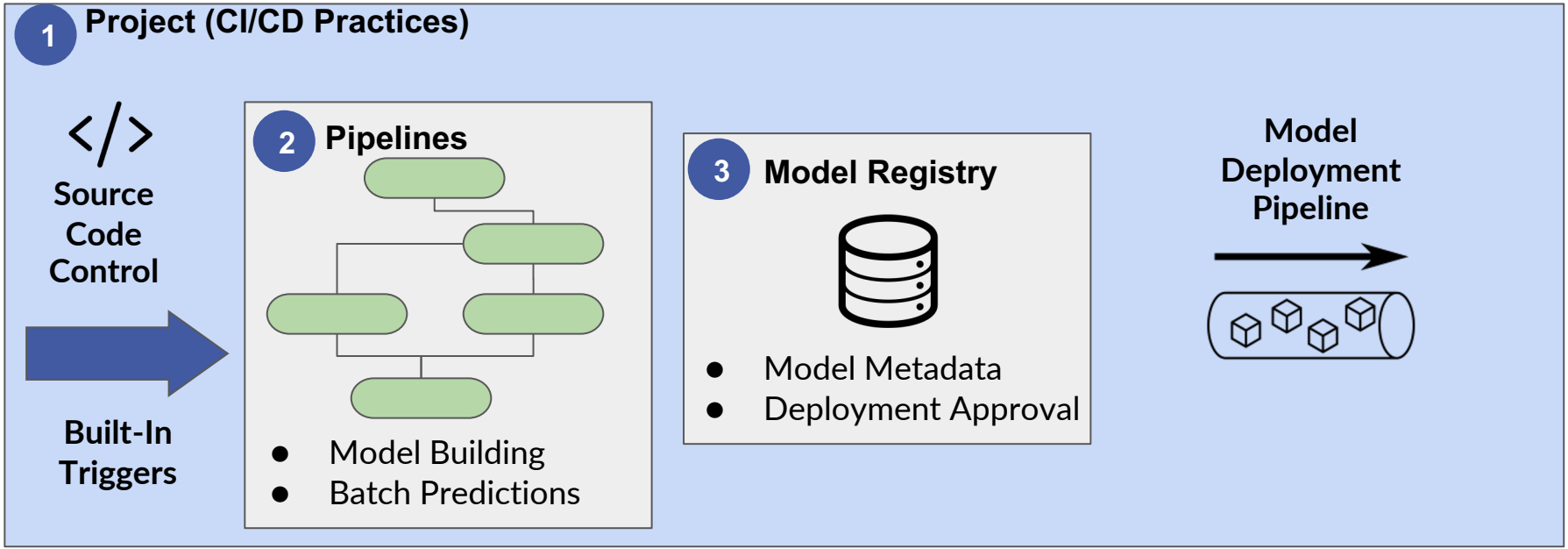
Automatic tracking of models



Bring CI/CD to Machine Learning

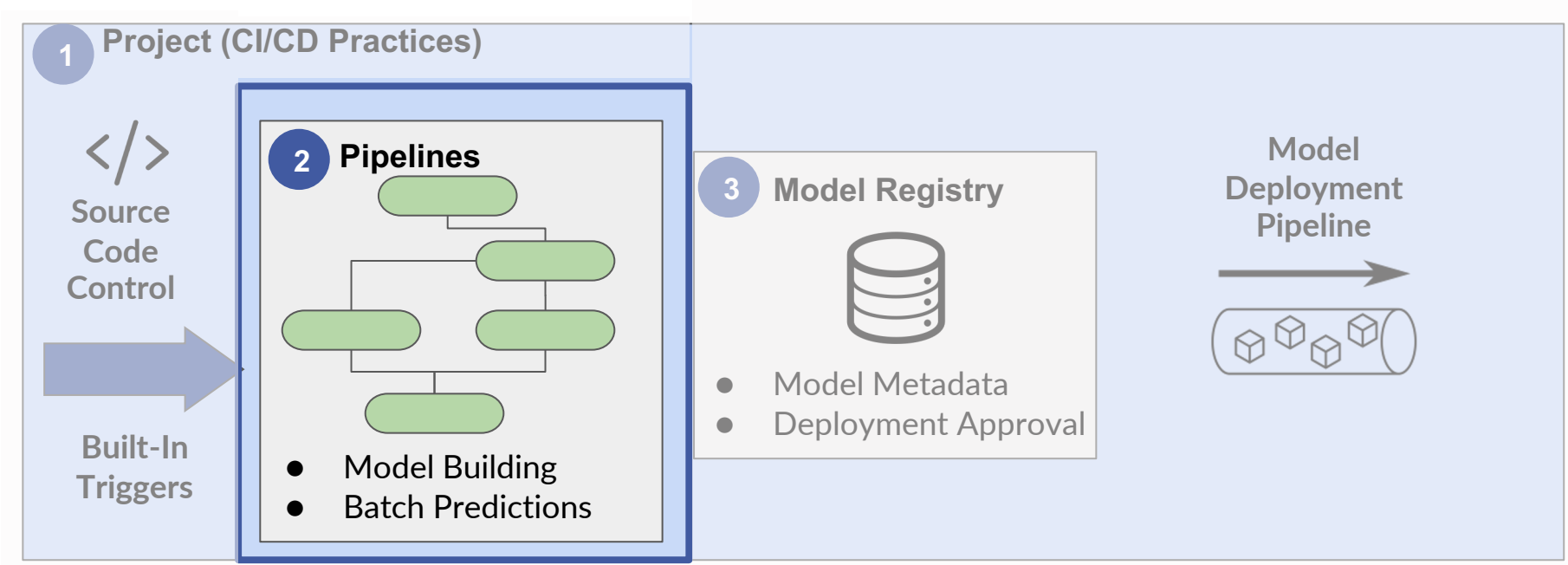
Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ....



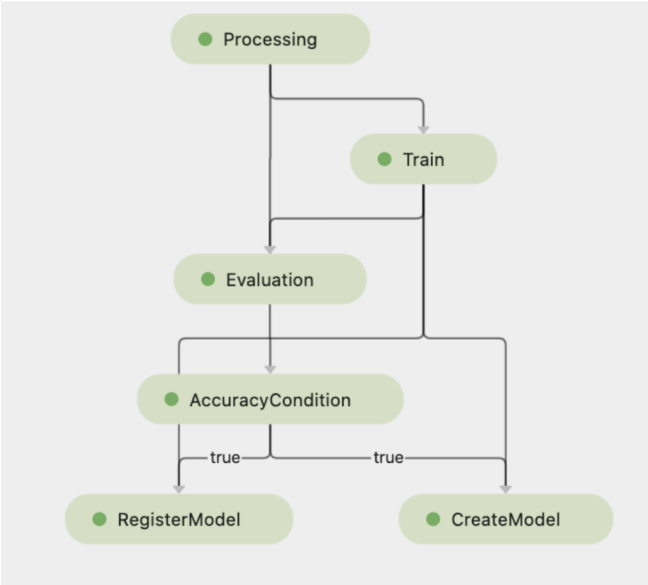
Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ....



SageMaker Pipelines

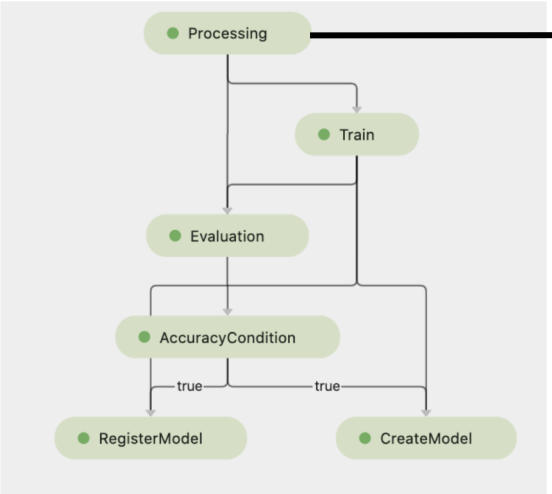
Pipelines



- ❑ Create Pipelines to build and evaluate models
- ❑ Python SDK for building workflows
- ❑ Pipeline visualization available through Amazon SageMaker Studio
- ❑ Fully managed pipelines - no servers to manage

SageMaker Pipelines

Processing Step



Use Amazon SageMaker Processing to process data set for training →

S3

1) *Product Reviews Dataset*
2) *Processing Script*

Inputs

Amazon SageMaker Processing Job

Sklearn Container Image

Script: prepare_data.py

Instance(s)

Processing Cluster

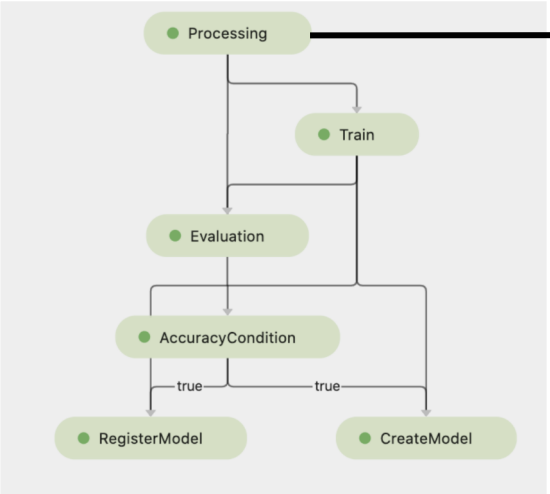
Outputs

S3

1) *Train/Validation/Test Datasets*

SageMaker Pipelines

Processing Step

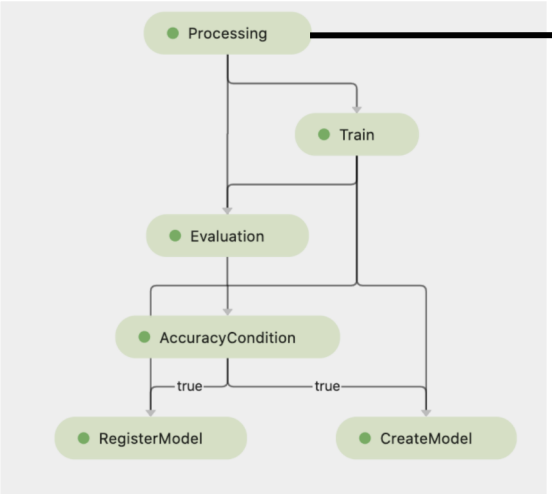


Define Step Inputs & Outputs →

```
processing_inputs = [
    ProcessingInput(
     input_name='customer-reviews-input-data',
     source='s3://...',
     destination='/opt/ml/processing/input/data/',
     s3_data_distribution_type='ShardedByS3Key'
    )
]

processing_outputs=[
         ProcessingOutput(...)
]
```

SageMaker Pipelines

Processing Step

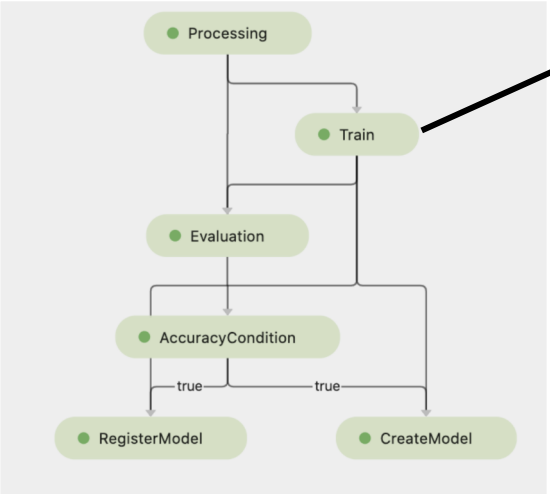


Configure the Processing Step →

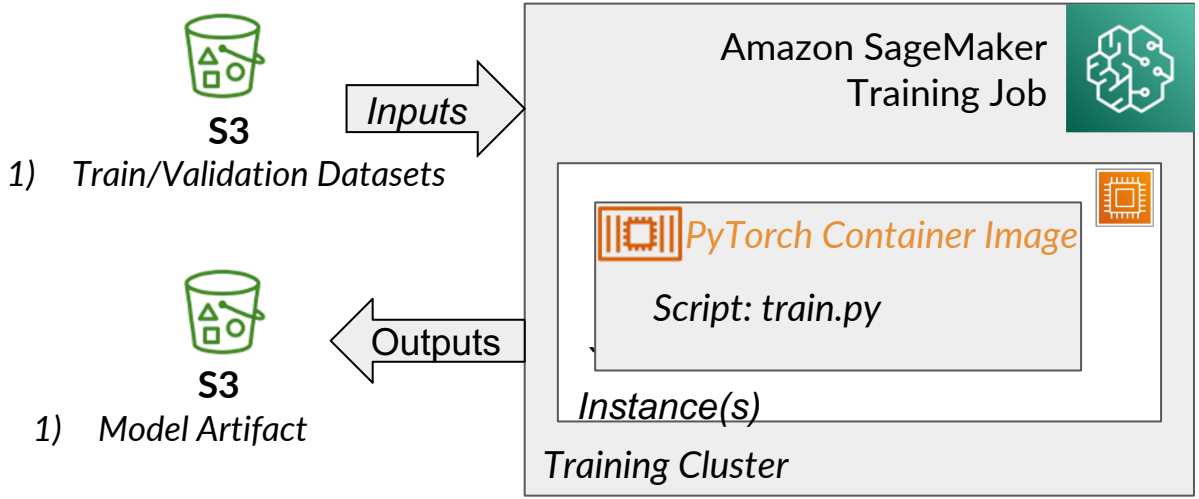
```
processing_step = ProcessingStep(
          name='Processing',
          code='src/prepare_data.py',
          processor=processor,
          inputs=processing_inputs,
          outputs=processing_outputs,
          job_arguments=[
          '--train-split-percentage',
          str(train_split_percentage.default_value,
          ...
      ]
)
```

SageMaker Pipelines

Training Step

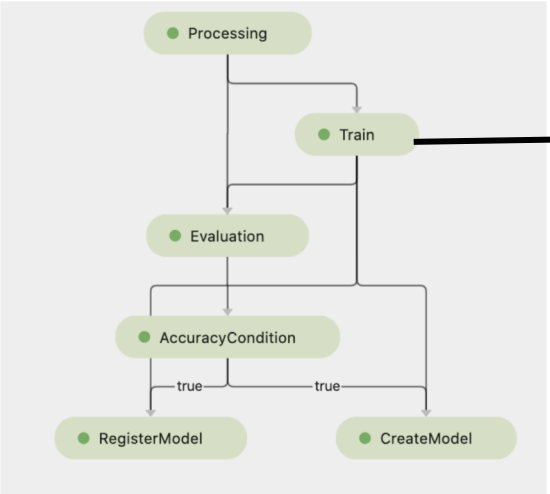


Use Amazon SageMaker Training Jobs to train the model using the outputs from the previous step as input →



SageMaker Pipelines

Training Step

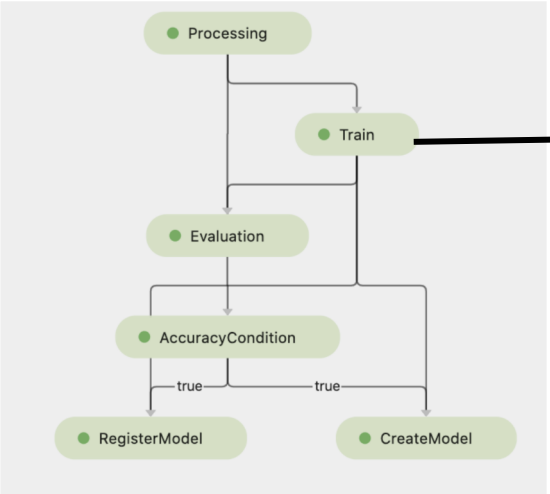


Configure Hyperparameters →

```
hyperparameters={
          'max_seq_length': max_seq_length,
          'epochs': epochs,
          'learning_rate': learning_rate,
      ...
}
```

SageMaker Pipelines

Training Step

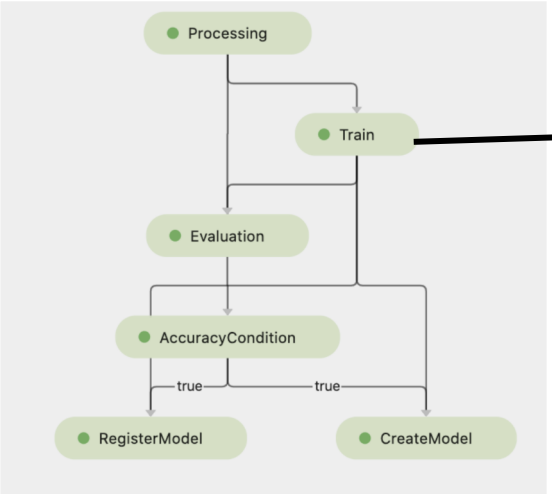


Configure Estimator →

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator
estimator = PyTorchEstimator(
            entry_point='train.py',
            source_dir='src',
            role=role,
            instance_count=train_instance_count,
            instance_type=train_instance_type,
            volume_size=train_volume_size,
            py_version='py3',
            framework_version='1.6.0',
            hyperparameters=hyperparameters,
            metric_definitions=metric_definitions,
            input_mode=input_mode
)
```

SageMaker Pipelines

Training Step



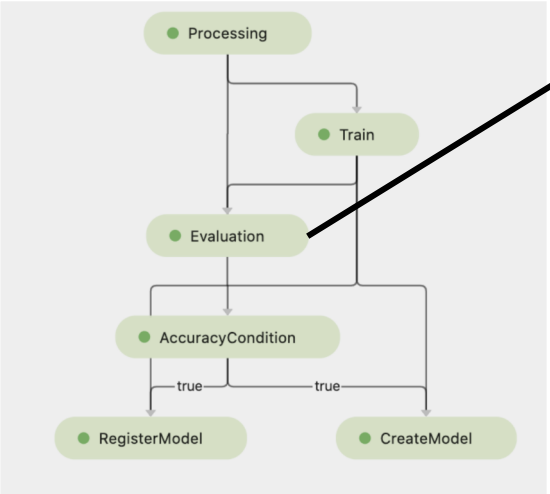
Configure the Training Step →

Output from Processing Step

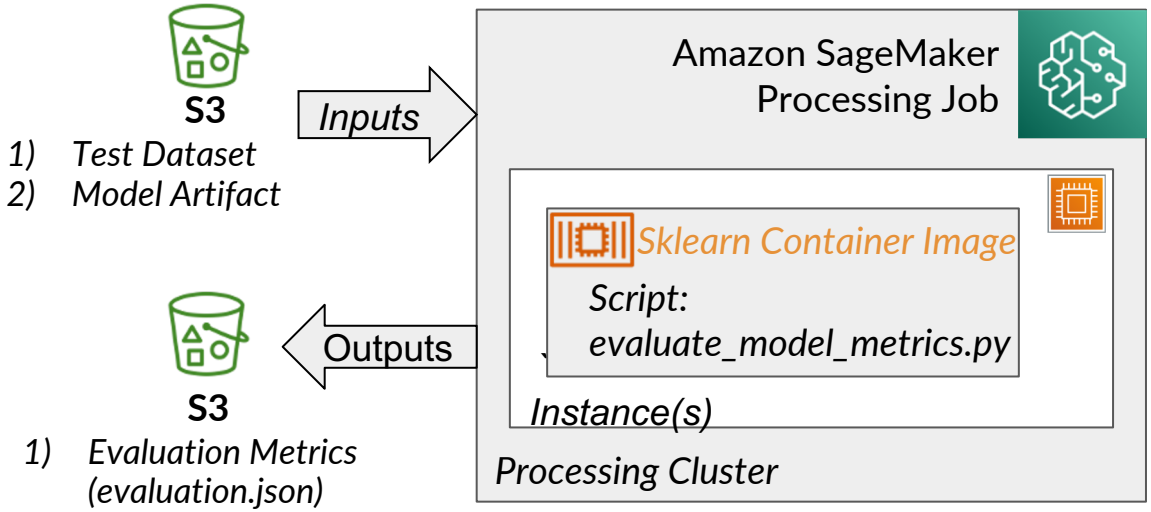
```
training_step = TrainingStep(
            name='Train',
            estimator=estimator,
            inputs={
            'train': TrainingInput(
            s3_data=processing_step.properties.ProcessingOu
tputConfig.Outputs[
                        'sentiment-train'
            ].S3Output.S3Uri,
            content_type='text/csv'
            ),
            'validation': TrainingInput(...
            )
    }
)
```

SageMaker Pipelines

Evaluation Step

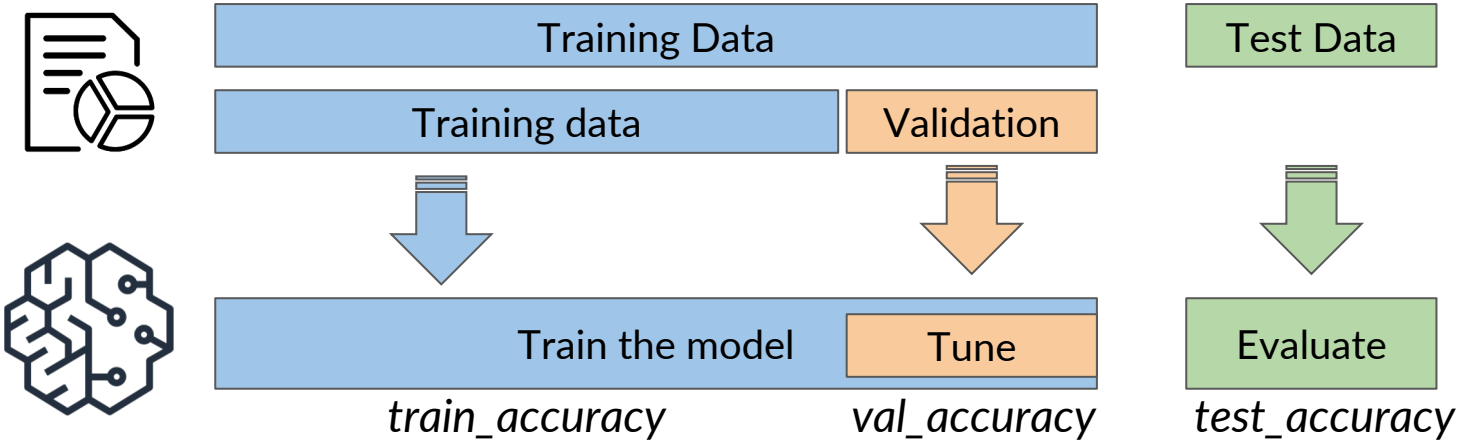


Use Amazon SageMaker Processing to evaluate trained model using test holdout dataset →



Model evaluation

- Evaluate the model with holdout test dataset



Code: *evaluate_model_metrics.py*

```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
...

def predict_fn(input_data, model):
    model.eval()
    ...
    return predicted_classes_jsonlines

...
y_test = df_test_reviews['review_body'].map(predict)
y_actual = df_test_reviews['sentiment'].astype('int64')

print(classification_report(y_true=y_test, y_pred=y_actual))

accuracy = accuracy_score(y_true=y_test, y_pred=y_actual)
print('Test accuracy: ', accuracy)
```

Define model predict function

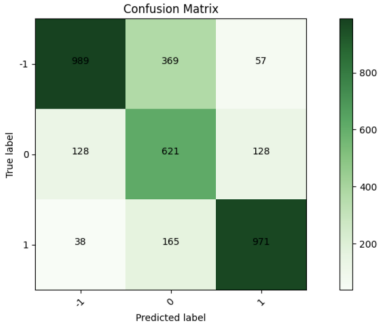
Use "test" holdout data

Calculate test accuracy

Analyze results

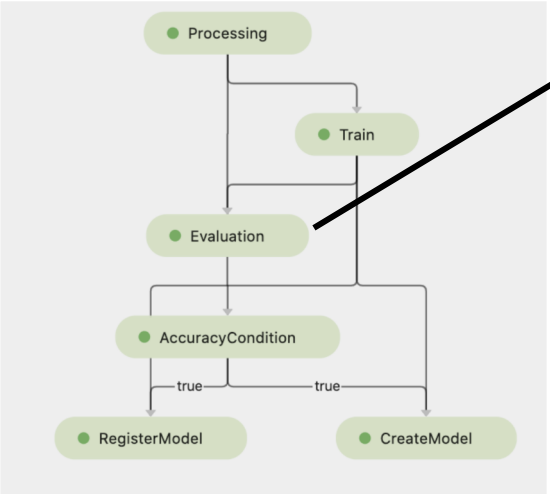
```
from pprint import pprint

evaluation_json = sagemaker.s3.S3Downloader.read_file(
    "{}/evaluation.json".format(evaluation_metrics_s3_uri))

pprint(json.loads(evaluation_json))
```

```
>> {'metrics': {'accuracy': {'value': 0.7458165031736872}}}
```



SageMaker Pipelines

Evaluation Step

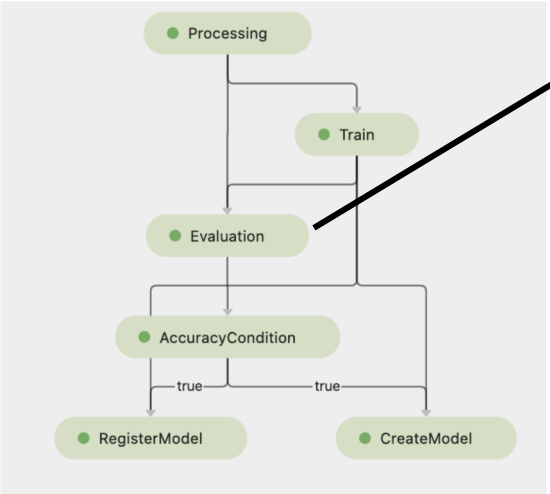


Define Output →

```
from sagemaker.workflow.properties import
PropertyFile

evaluation_report = PropertyFile(
        name='EvaluationReport',
        output_name='metrics',
        path='evaluation.json'
)
```

SageMaker Pipelines

Evaluation Step

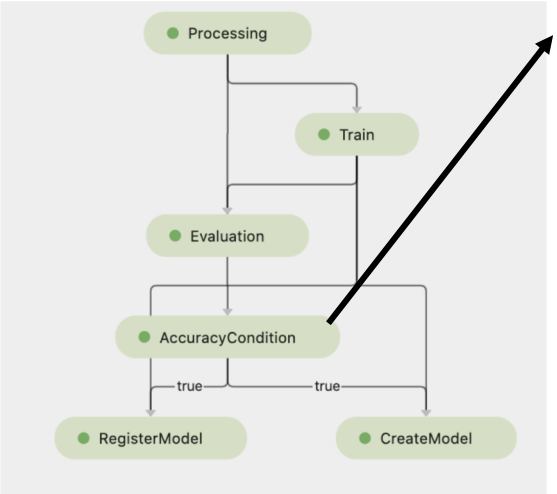


Configure Processing Step →

```
evaluation_step = ProcessingStep(
            name='EvaluateModel',
            processor=evaluation_processor,
            code='src/evaluate_model_metrics.py',
            inputs=[
            ProcessingInput(...),...
            ],
            outputs=[
            ProcessingOutput(...),
            ],
            job_arguments=[...],
            property_files=[evaluation_report],
)
```

SageMaker Pipelines

Condition Step



Use Amazon SageMaker Pipelines *Condition Step* to conditionally execute step(s) →

Evaluation Step

Accuracy Metric

Conditional Step

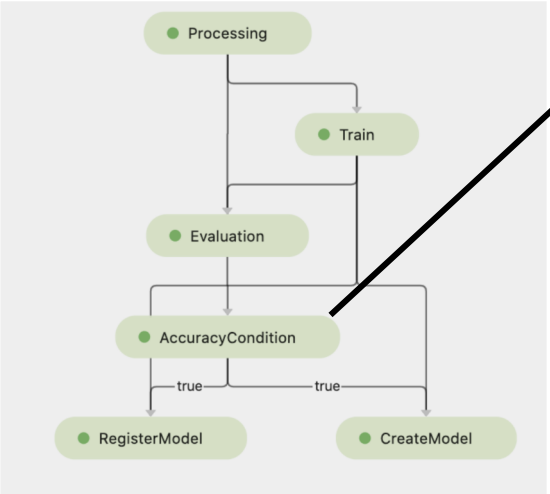
Accuracy Value? {.99}

If steps

Else steps

SageMaker Pipelines

Condition Step



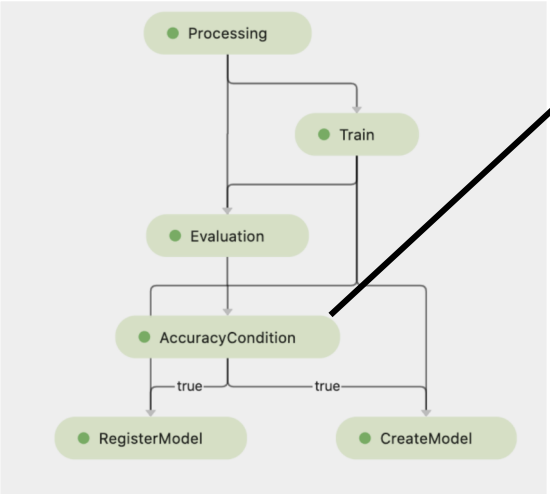
Define a Condition & Import Conditional Workflow Step →

```
min_accuracy_value = ParameterFloat(
            name="MinAccuracyValue",
            default_value=0.01
)
```

```
from sagemaker.workflow.conditions import
ConditionGreaterThanOrEqualTo

from sagemaker.workflow.condition_step import (
            ConditionStep,
            JsonGet,
)
```

SageMaker Pipelines

Condition Step

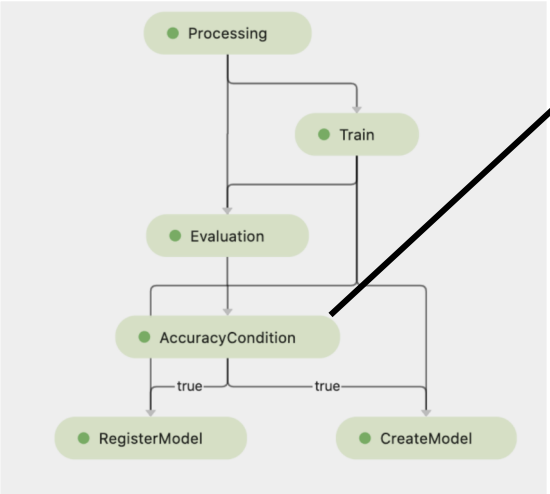


Configure the Condition →

```
minimum_accuracy_condition =
ConditionGreaterThanOrEqualTo(
          left=JsonGet(
          step=evaluation_step,
          property_file=evaluation_report,
          json_path="metrics.accuracy.value",
          ),
          right=min_accuracy_value # accuracy
)
```

SageMaker Pipelines

Condition Step

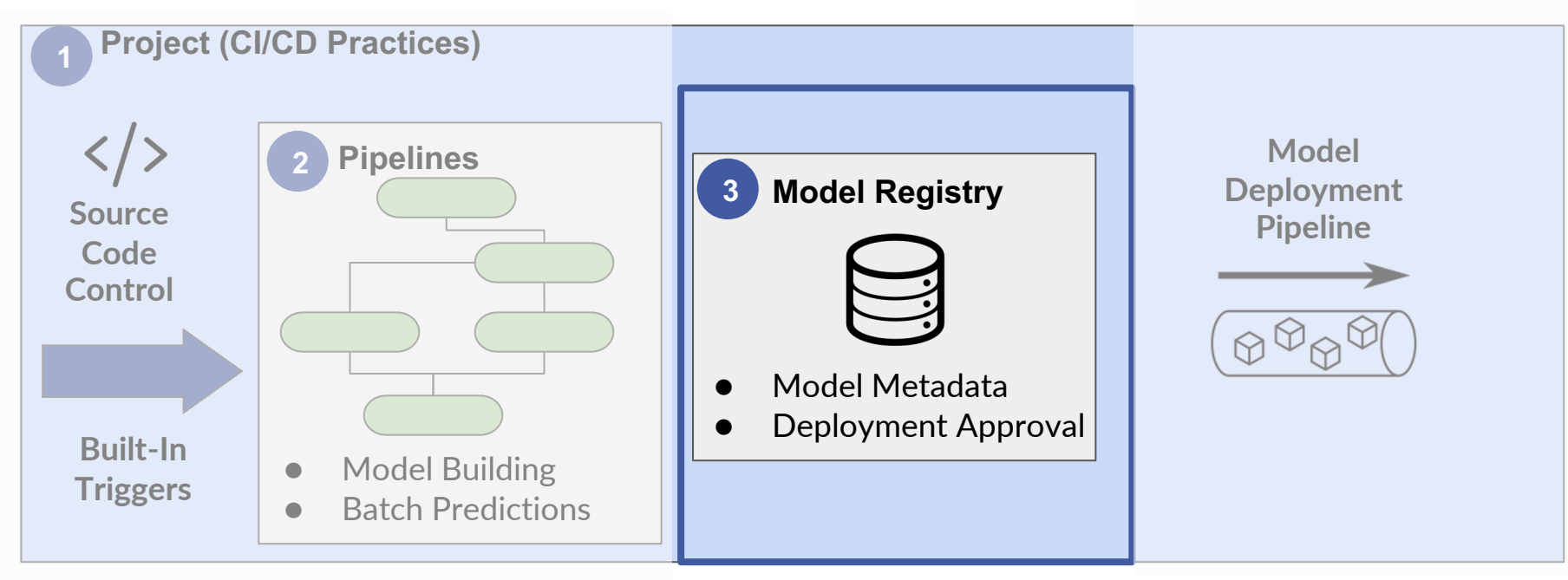


Configure the Condition Step →

```
minimum_accuracy_condition_step = ConditionStep(
        name="AccuracyCondition",
        conditions=[minimum_accuracy_condition],
    # success, continue with model registration
        if_steps=[register_step, create_step],
        else_steps=[], # fail, end the pipeline
)
```

Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ....



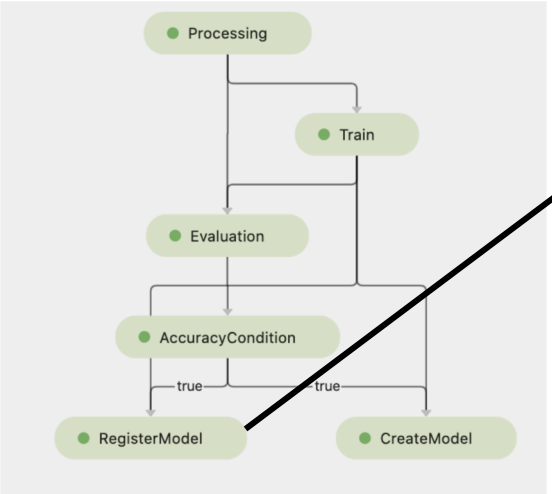
SageMaker Pipelines

Model Registry

- ❑ Catalog models for production
- ❑ Manage model versions & metadata
- ❑ Manage the approval status of a model
- ❑ Trigger model deployment pipeline

SageMaker Pipelines

Register Model Step

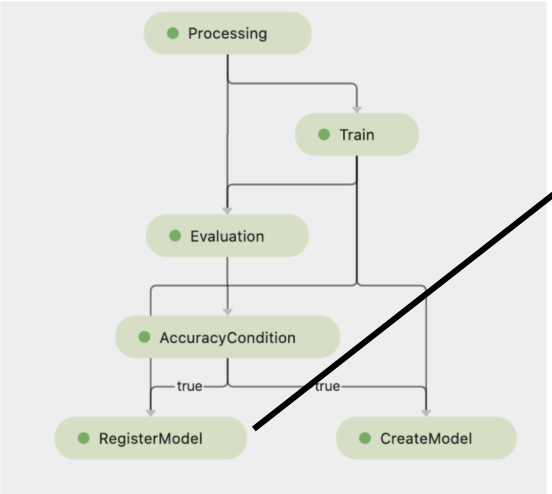


Define deployment image for inference →

```
inference_image_uri = sagemaker.image_uris.retrieve(
          framework="pytorch",
          region=region,
          version="1.6.0",
          py_version="py36",
          instance_type=deploy_instance_type,
          image_scope="inference"
)
```

SageMaker Pipelines

Register Model Step

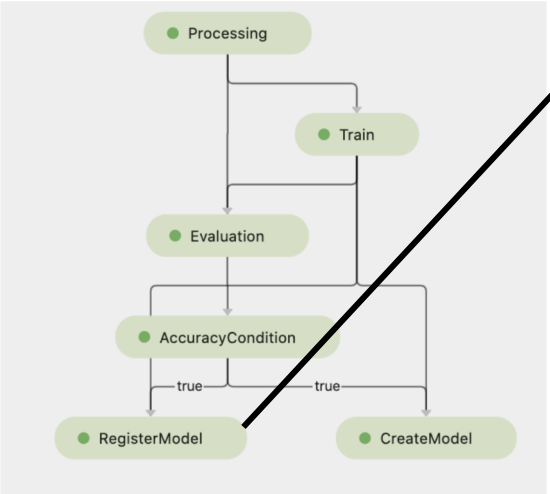


Define model metrics to be stored as metadata →

```
from sagemaker.model_metrics import MetricsSource,
ModelMetrics

model_metrics = ModelMetrics(
        model_statistics=MetricsSource(
        s3_uri="s3://..."),
        content_type="application/json"
        )
)
```

SageMaker Pipelines

Register Model Step

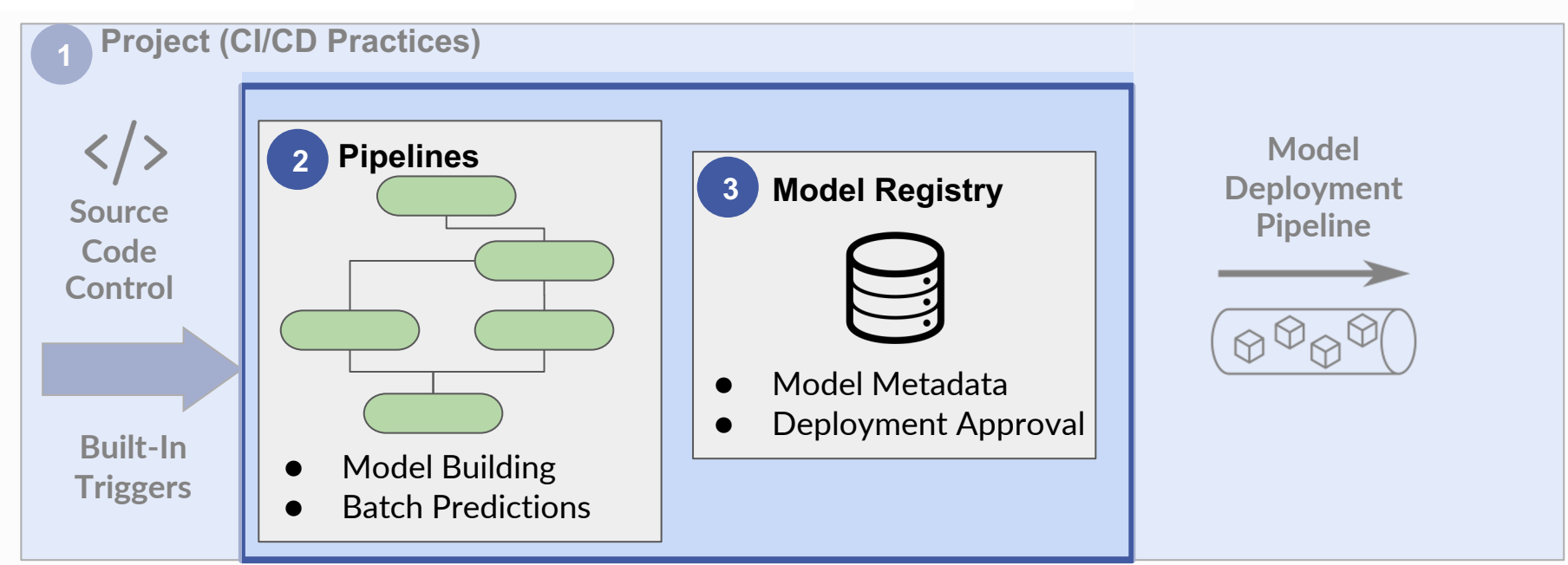


Configure the Register Model Step →

```
register_step = RegisterModel(
            name="RegisterModel",
            estimator=estimator,
            image_uri=...,
      model_data=
            training_step.properties.ModelArtifacts.S3ModelArtifacts,
      content_types=["application/jsonlines"],
      response_types=["application/jsonlines"],
      inference_instances=[deploy_instance_type],
      transform_instances=['ml.m5.xlarge'], # batch transform
      model_package_group_name=model_package_group_name,
      approval_status=model_approval_status,
      model_metrics=model_metrics)
```

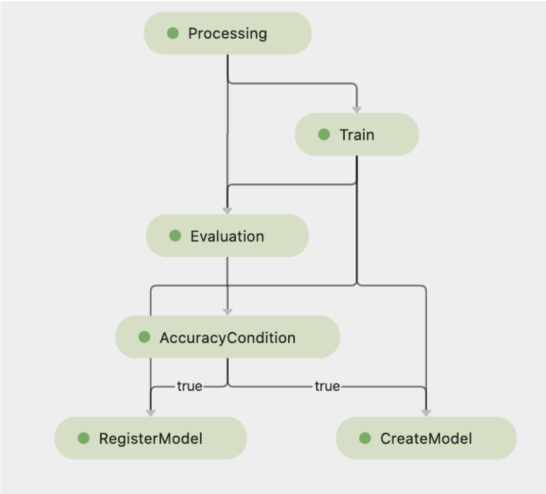
Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ....



SageMaker Pipelines

Bringing It All Together

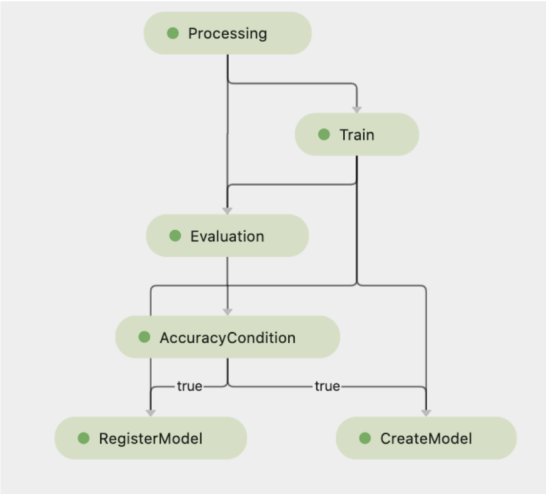


Configure the Pipeline →

```
from sagemaker.workflow.pipeline import Pipeline

pipeline = Pipeline(
      name=pipeline_name,
            parameters=[
            input_data,
            processing_instance_count,
            ...
      ],
      steps=[processing_step, training_step, evaluation_step,
      minimum_accuracy_condition_step], sagemaker_session=sess,
)
```

SageMaker Pipelines

Bringing It All Together



Create & Execute the Pipeline →

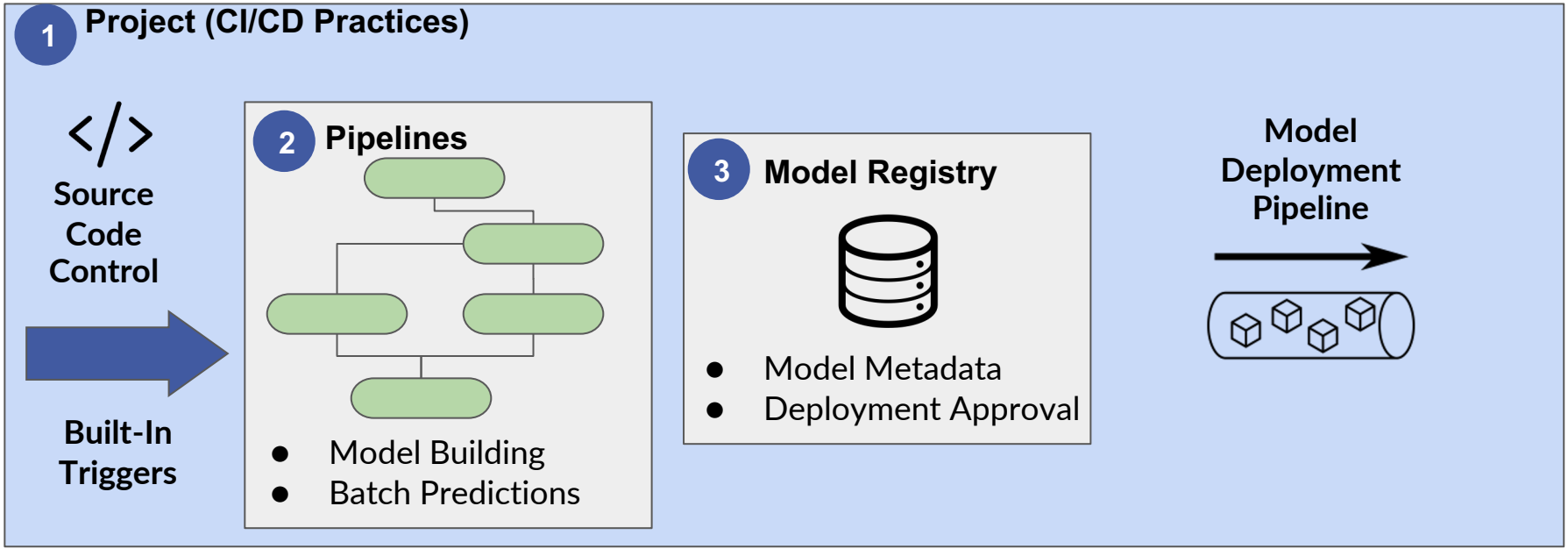
```
response = pipeline.create(role_arn=role)

pipeline_arn = response["PipelineArn"]

execution = pipeline.start(
            parameters=dict(
            InputData=raw_input_data_s3_uri,
            ProcessingInstanceCount=1,
      ...
            )
)
```

Machine Learning Pipelines

with
Amazon SageMaker Projects

Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ....



SageMaker Projects

- Create end-to-end ML solutions with CI/CD practices
- Incorporates source/version control
- 3 Built-In MLOps Project Templates covering:
  - Build, Train, Deploy
  - Build, Train
  - Deploy
- Ability to bring custom Project templates