Copyright Notice

Machine Learning Operations (MLOps)

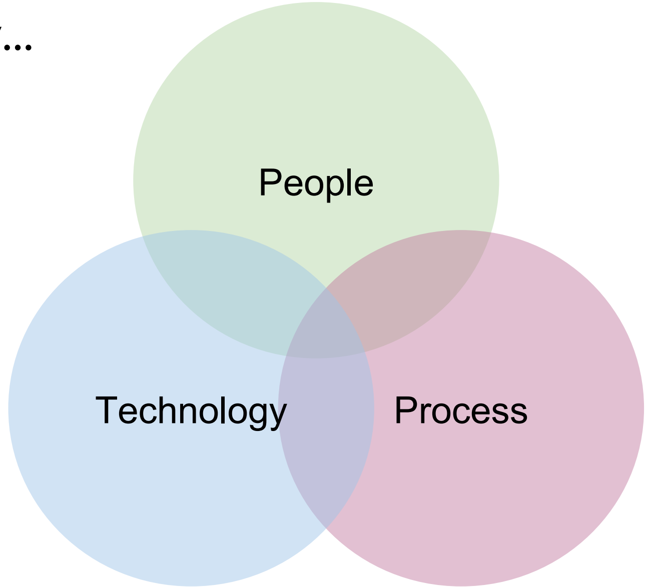
Overview

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)

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

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

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

Ingest & Analyze → **Prepare & Transform** → **Train & Tune** → **Deploy & Manage**

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

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

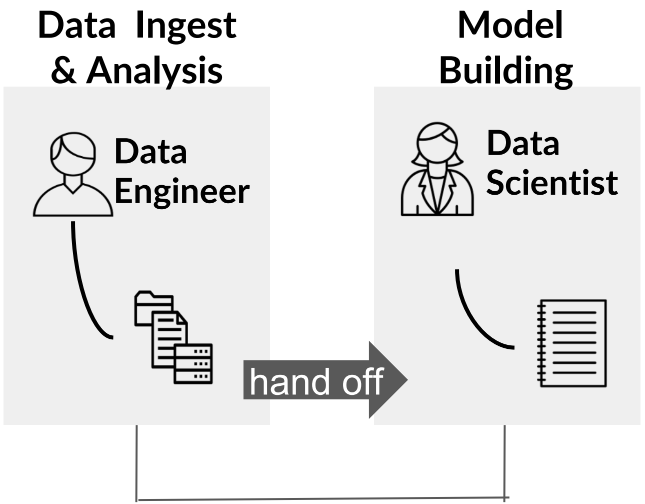
Build resilient, secure, performant, operationally efficient and cost optimized ML solutions

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

Operationalizing Machine Learning

Path to production

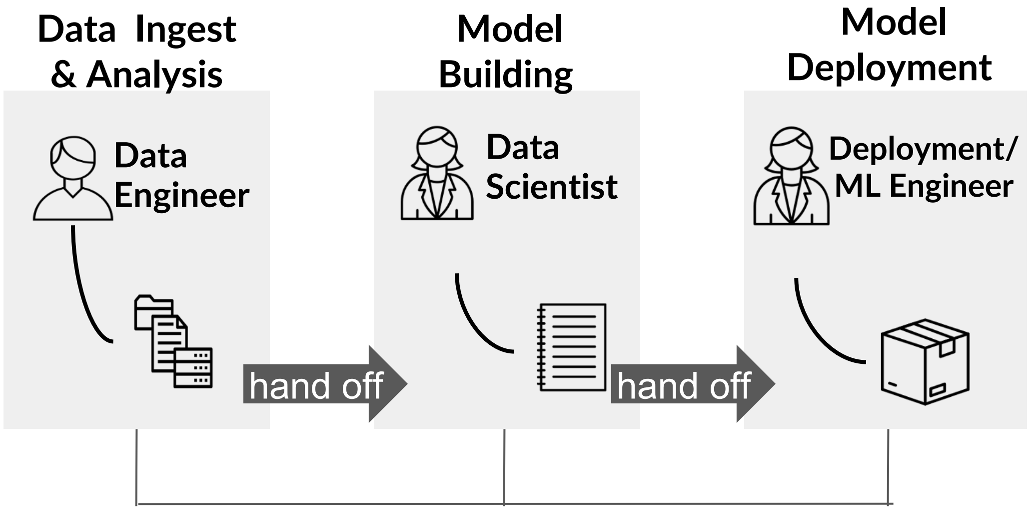
Example workflow with multiple hand offs:



Operationalizing Machine Learning

Path to production

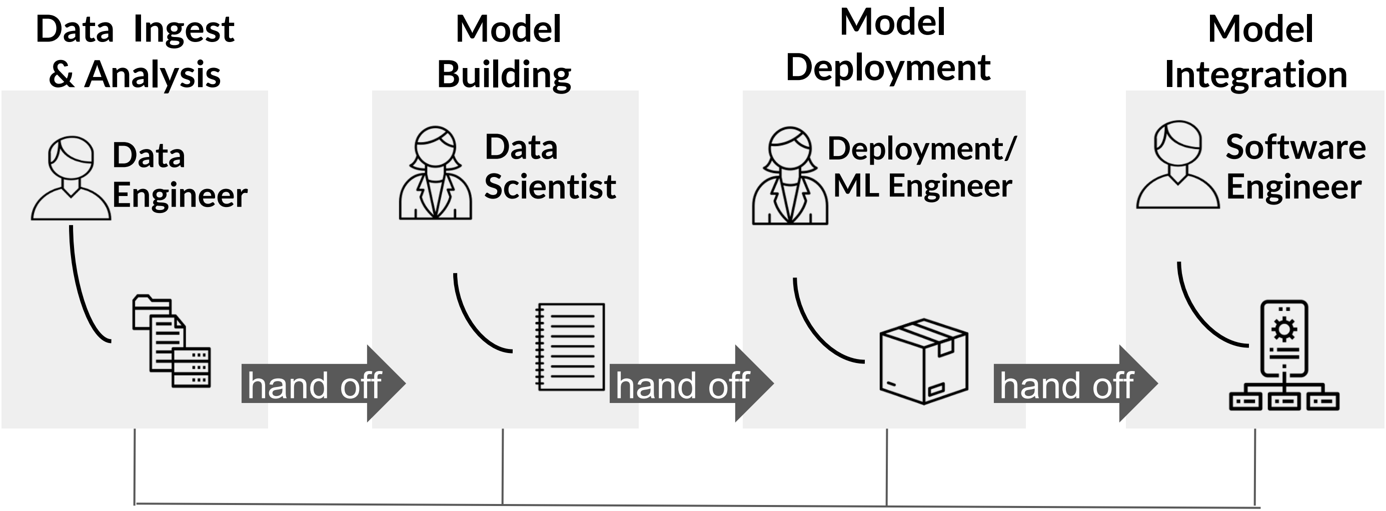
Example workflow with multiple hand offs:



Operationalizing Machine Learning

Path to production

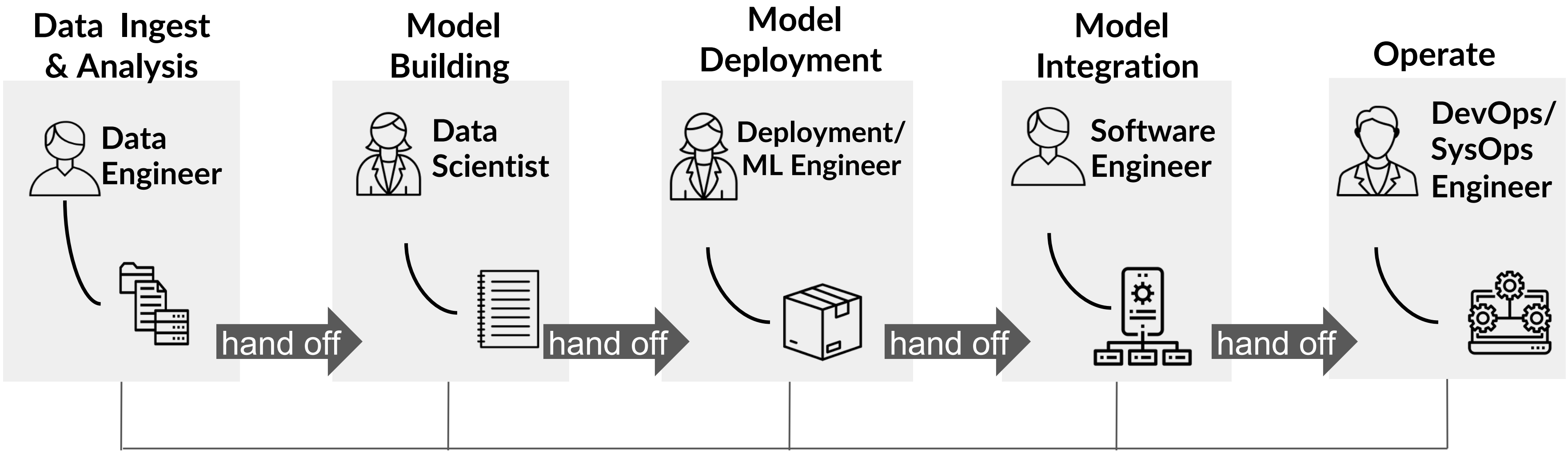
Example workflow with multiple hand offs:



Operationalizing Machine Learning

Path to production

Example workflow with multiple hand offs:

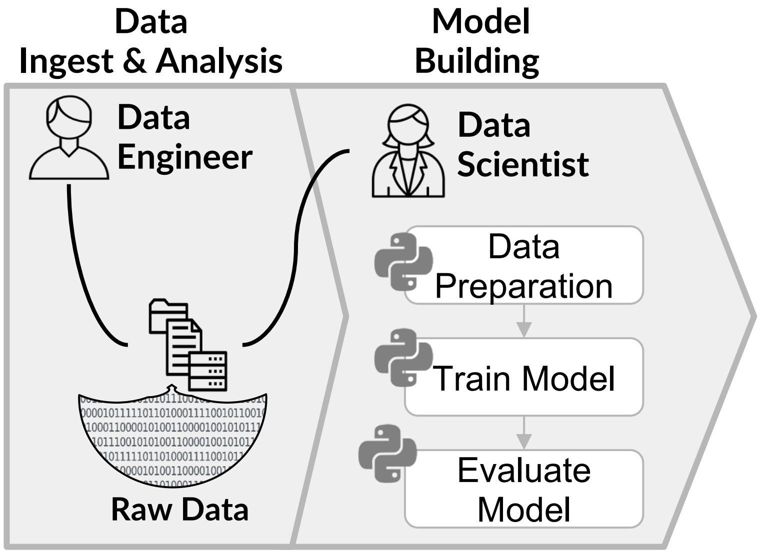


- Multiple handoffs
- Increased rework
- Limited Visibility & Transparency

Operationalizing Machine Learning

Accelerate the path to production

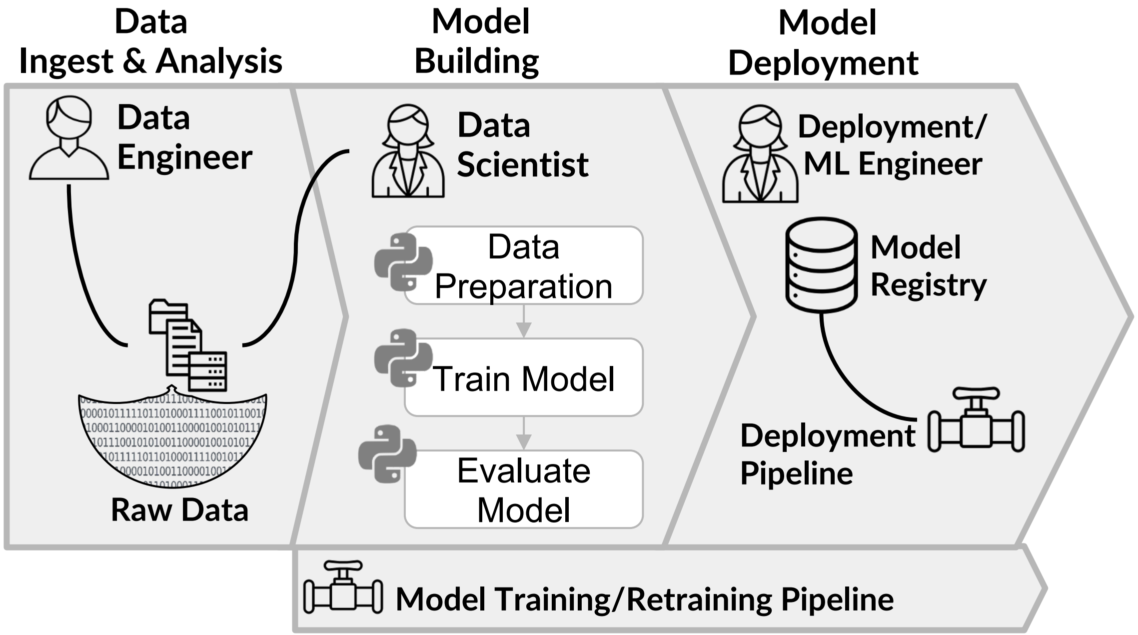
Automating the tasks within a workflow step



Operationalizing Machine Learning

Accelerate the path to production

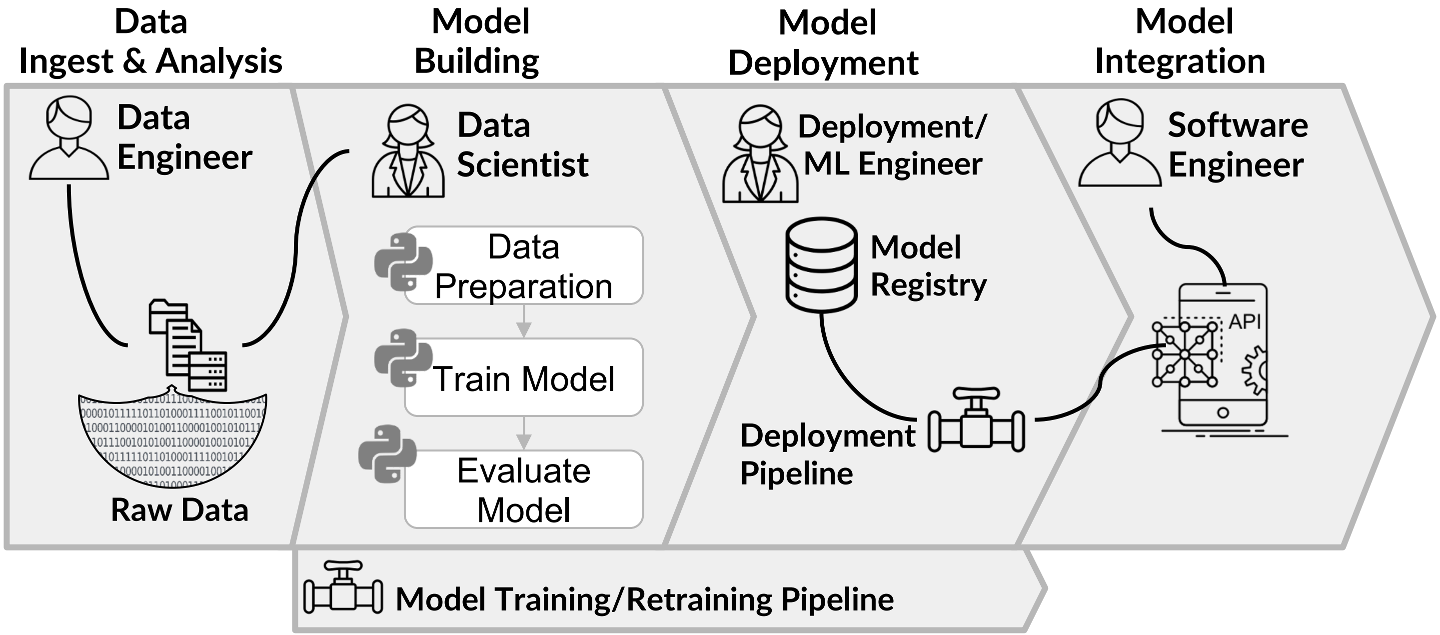
Automating the tasks within a workflow step



Operationalizing Machine Learning

Accelerate the path to production

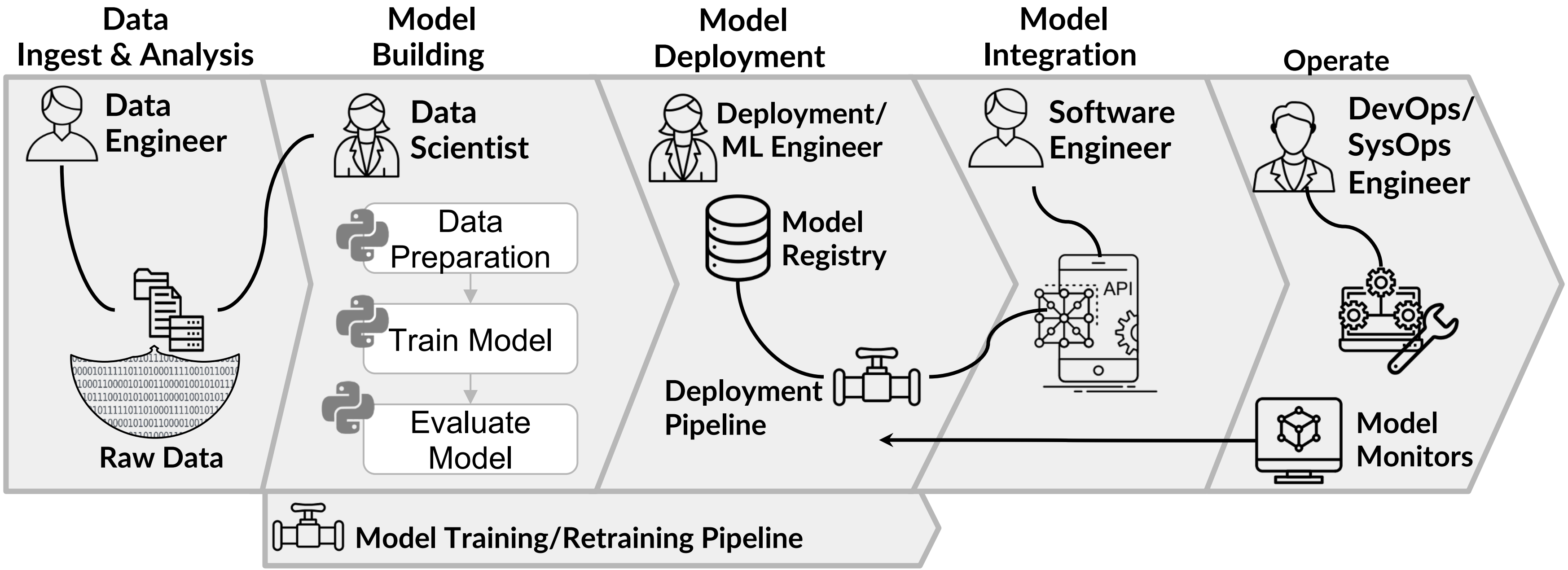
Automating the tasks within a workflow step



Operationalizing Machine Learning

Accelerate the path to production

Automating the tasks within a workflow step

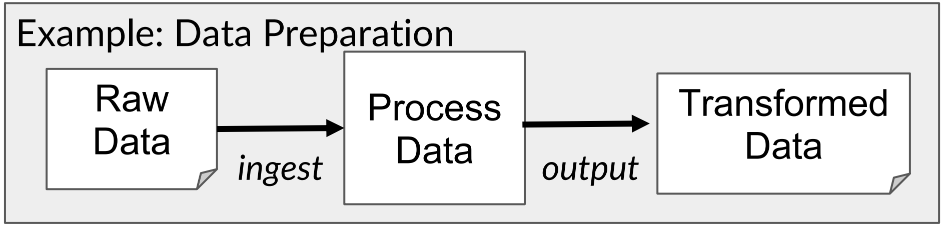


Operationalizing Machine Learning

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

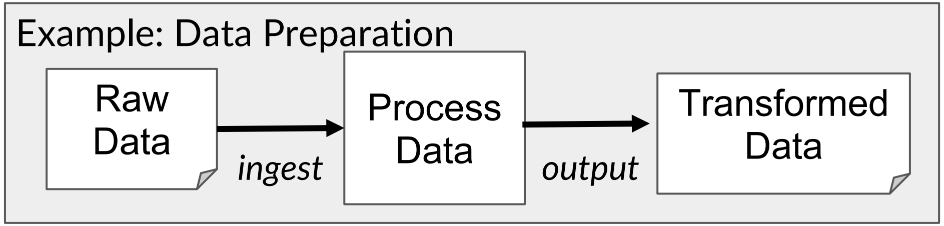


Operationalizing Machine Learning

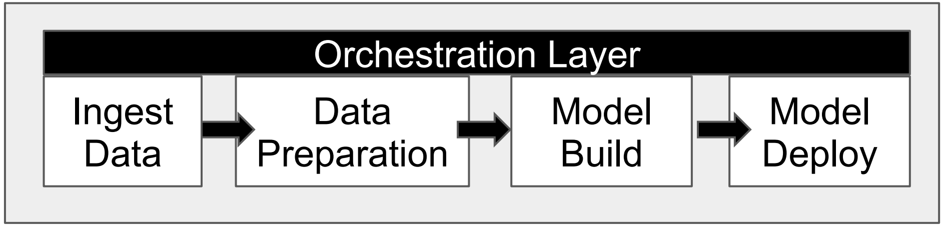
Accelerate the path to production

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



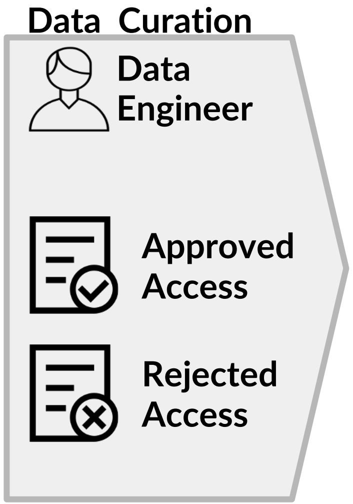
Orchestration: Orchestrate the steps of a workflow that contain a **collection of tasks**



Operationalizing Machine Learning

Improve the quality of deployed models

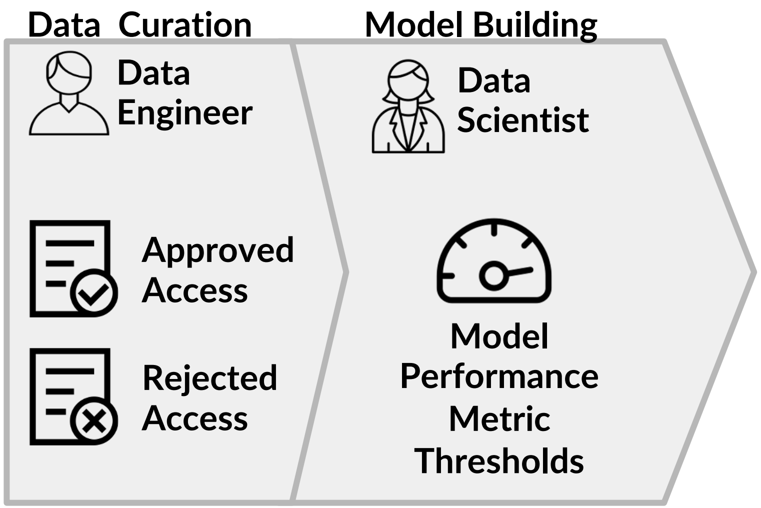
Examples: Automated quality gates



Operationalizing Machine Learning

Improve the quality of deployed models

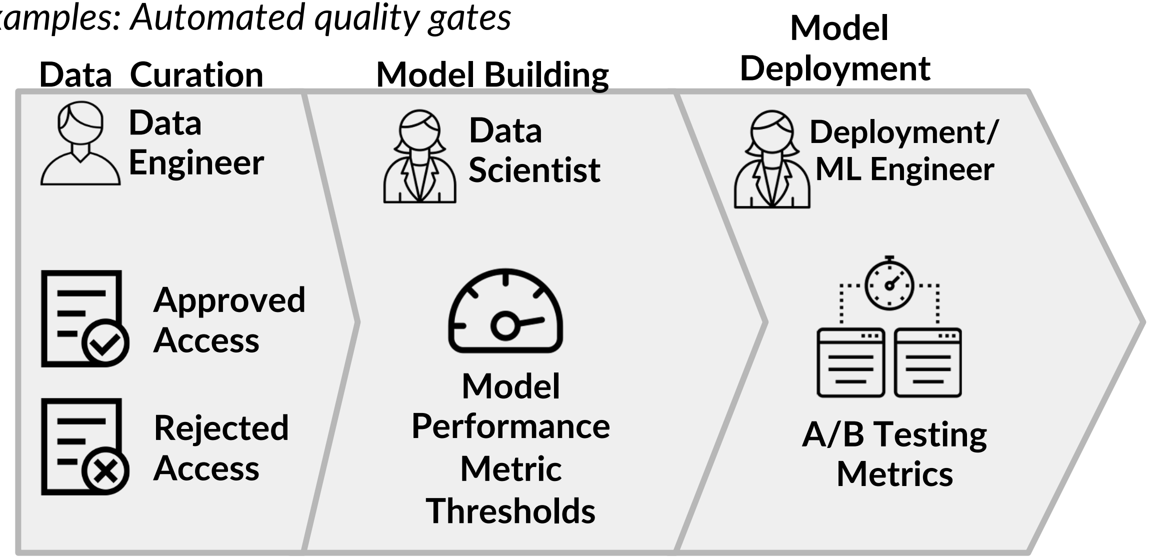
Examples: Automated quality gates



Operationalizing Machine Learning

Improve the quality of deployed models

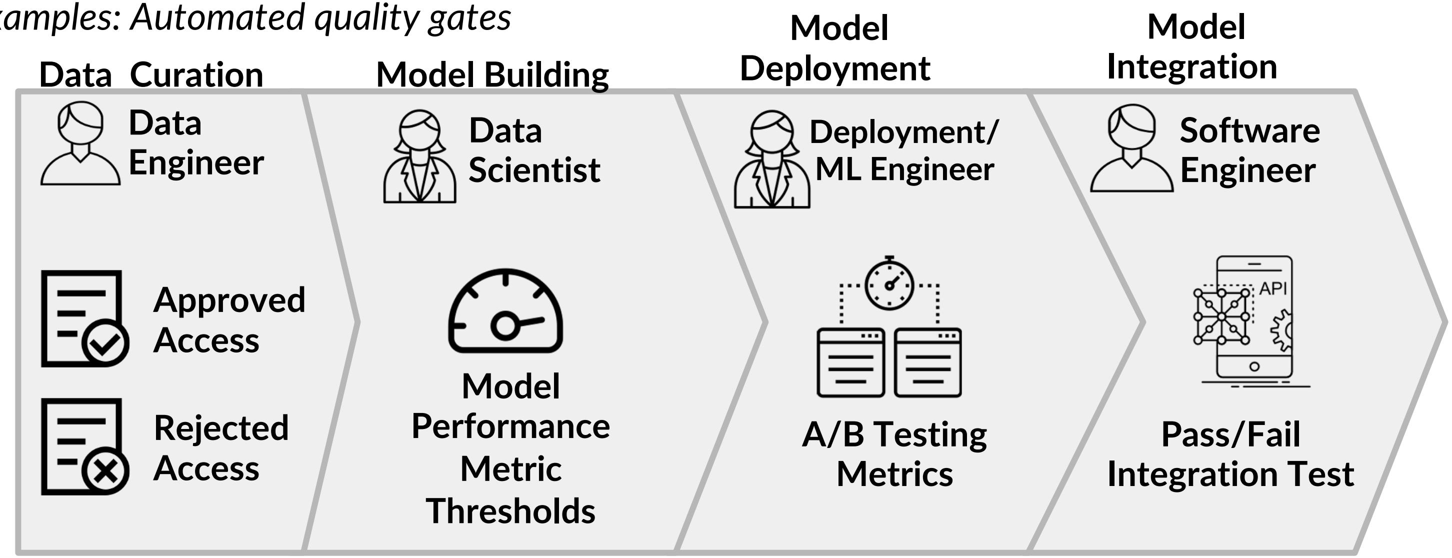
Examples: Automated quality gates



Operationalizing Machine Learning

Improve the quality of deployed models

Examples: Automated quality gates



Operationalizing Machine Learning

Improve the quality of deployed models

Examples: Automated quality gates

Data Curation

Data Engineer

- Approved Access
- Rejected Access

Model Building

Data Scientist

Model Performance Metric Thresholds

Model Deployment

Deployment/ ML Engineer

A/B Testing Metrics

Model Integration

Software Engineer

Pass/Fail Integration Test

Operate

DevOps/ SysOps Engineer

Model Monitor *Ex. Data Drift*

Operationalizing Machine Learning

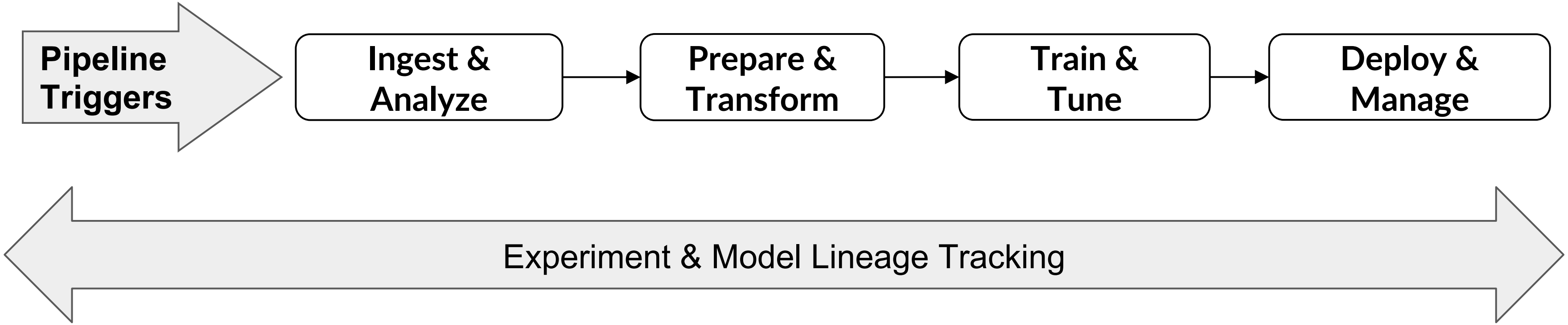
Key Considerations

- **Security**
- **Reliability**
- **Cost Optimization**
- **Performance Efficiency**
- **Operational Excellence**

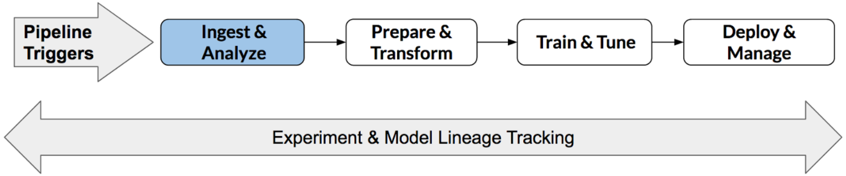
Creating Machine Learning Pipelines

Creating Machine Learning Pipelines

Building Effective Pipelines



Data Tasks



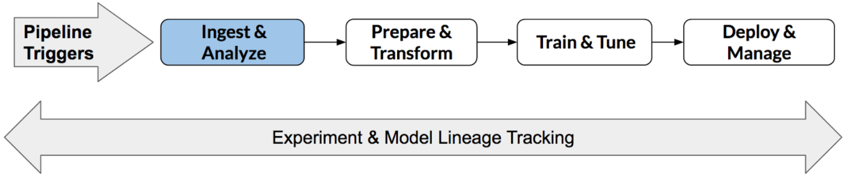
Data Ingestion for Model Development

FROM...

Data Scientist: Can I get an extract from our CRM System?

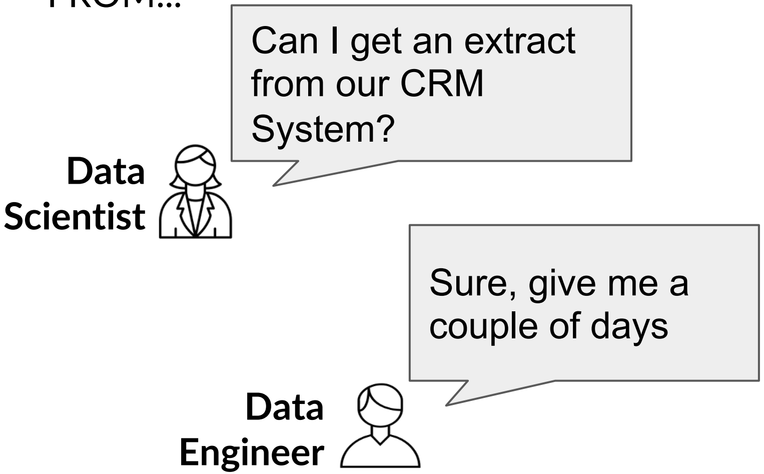
Data Engineer: Sure, give me a couple of days

Data Tasks

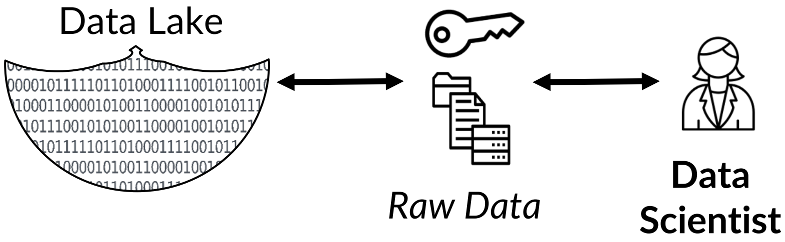


Data Ingestion for Model Development

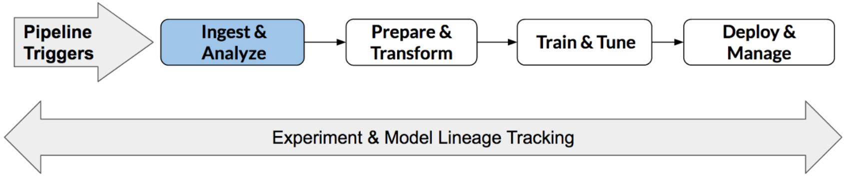
FROM...



TO...



Data Tasks



Data Ingestion for Model Retraining

FROM...

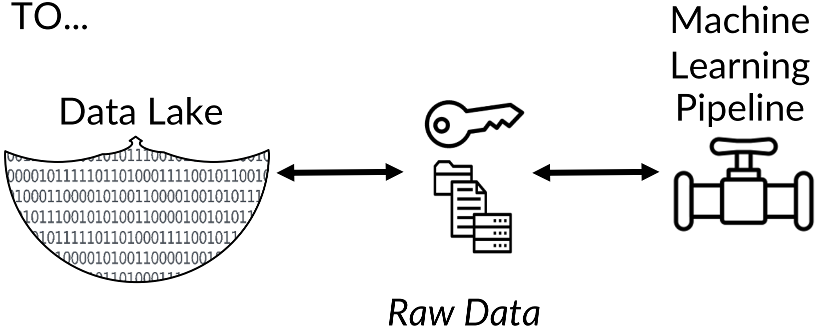
Can I get an extract from our CRM System?

Data Scientist

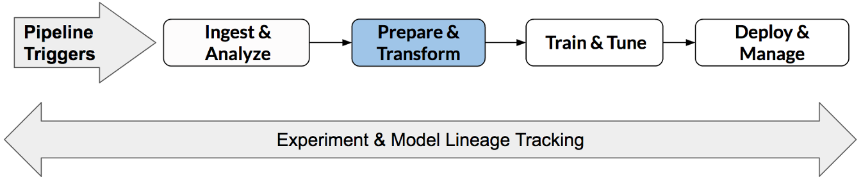
Sure, give me a couple of days

Data Engineer

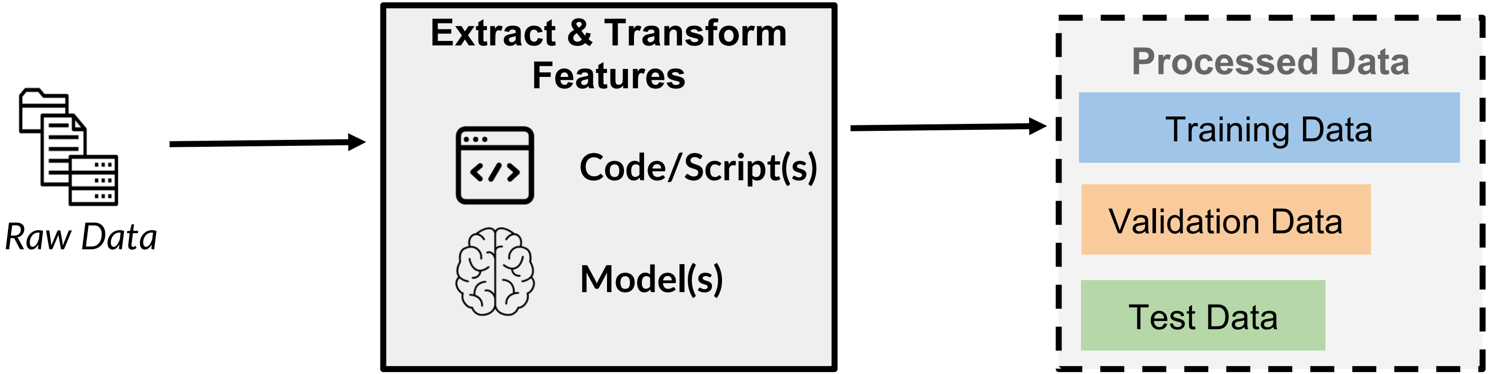
TO...



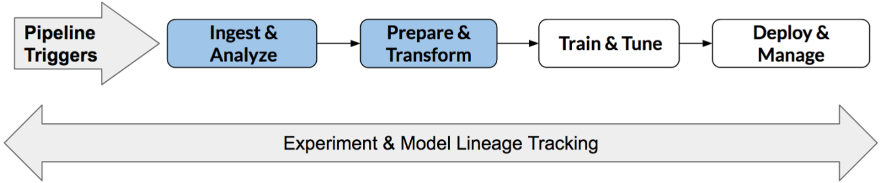
Data Tasks



Data Pre-Processing & Feature Engineering

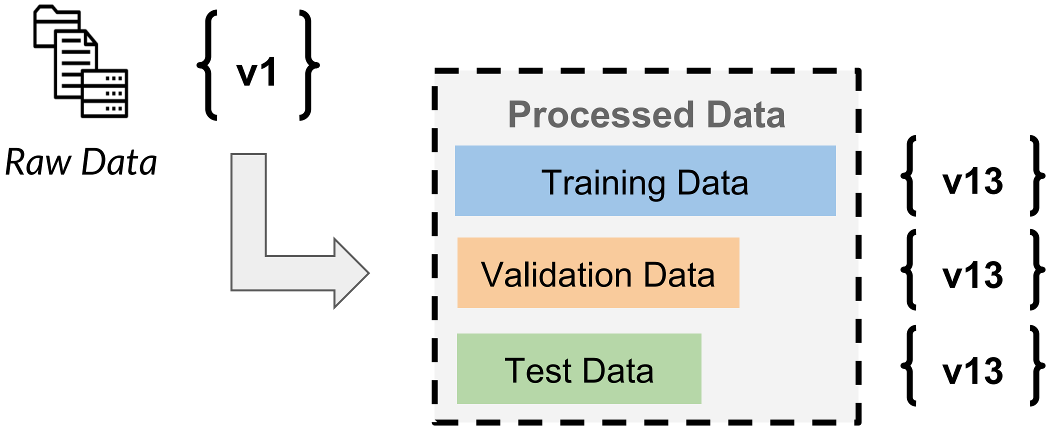


Data Tasks

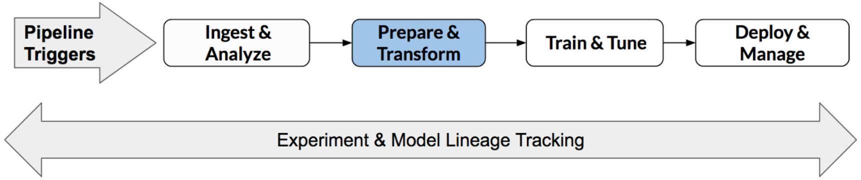


Data Versioning

Examples:



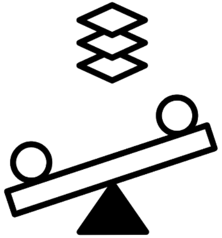
Data Tasks



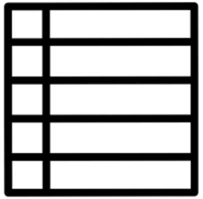
Data Validation

Examples:

Data Quality

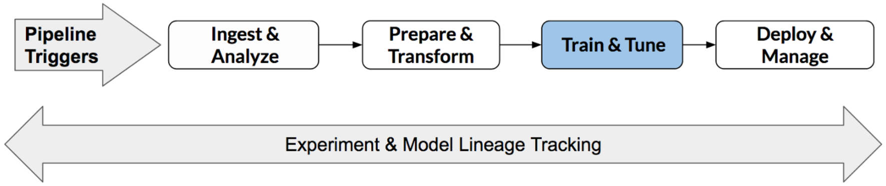


Statistical Bias

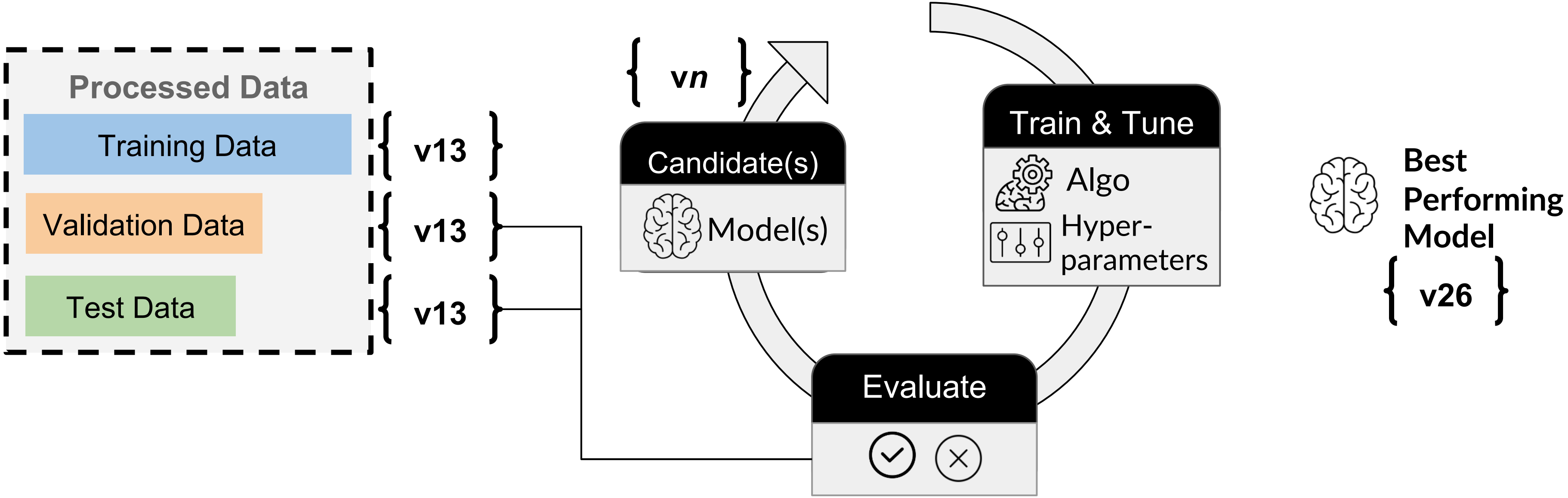


Data Schema

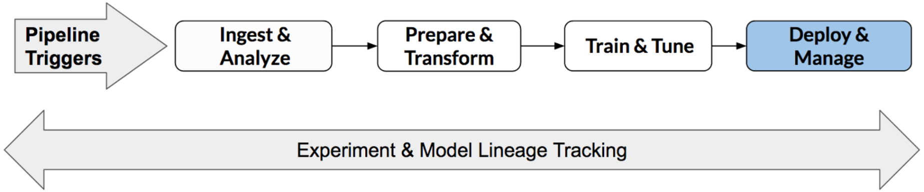
Model Building Tasks



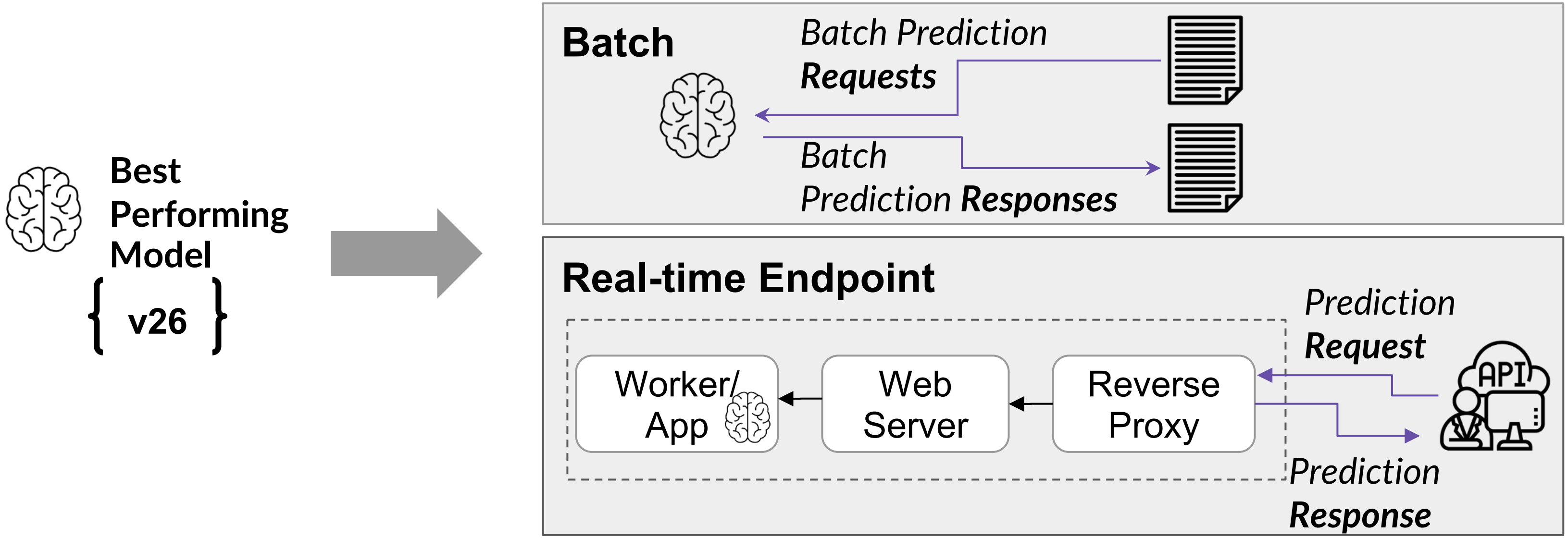
Model Training, Evaluation & Versioning



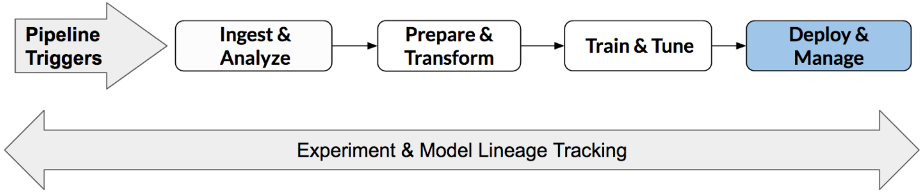
Model Deployment Tasks



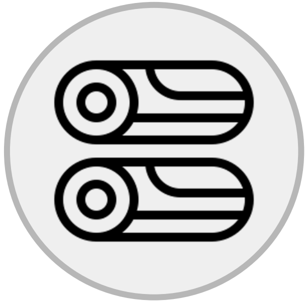
Model Deployment & Consumption



Operating Tasks



Logging & Monitoring



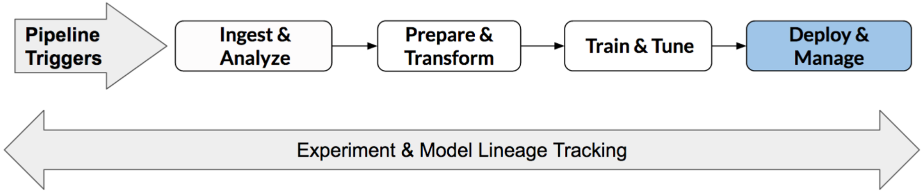
Logging →

- Model Data
- System Data

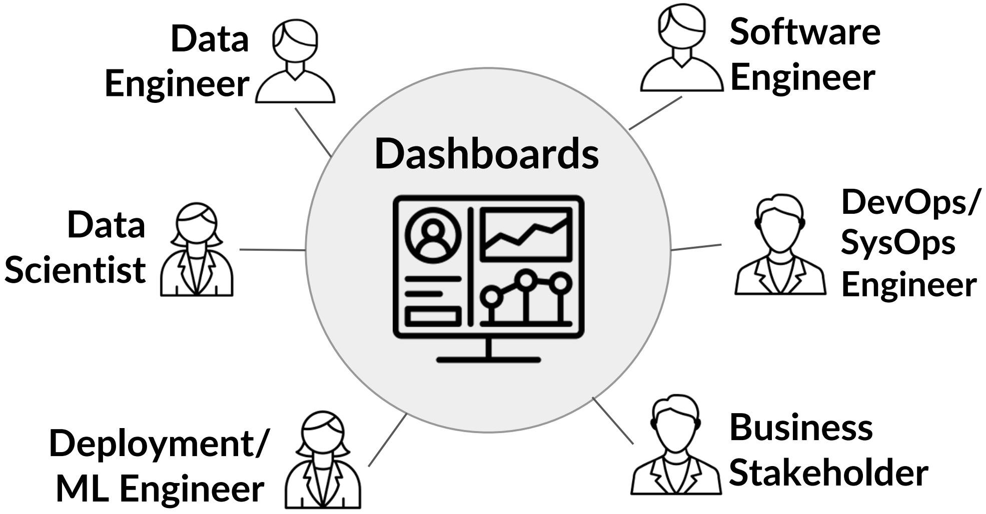
Monitoring →

- Collect Metrics
- Setup Alerts
- Trigger Automated Flows

Operating Tasks



Additional Feedback Mechanisms

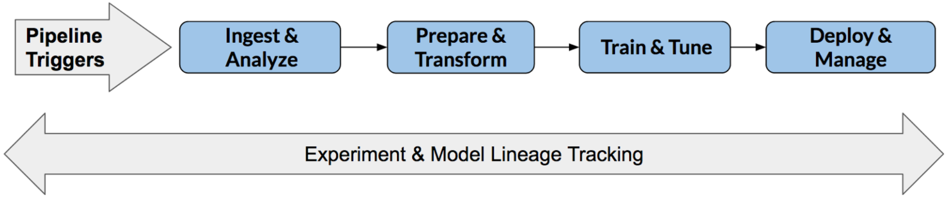


- Each persona can have different motivations and needs for monitors, logging and dashboards.
- Examples:
  - Pipeline Status
  - System Performance
  - Model Performance

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



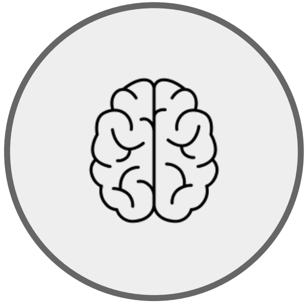
- ❑ **Orchestration** is required to coordinate the execution of tasks and steps within the tasks.

Model Lineage & Artifact Tracking

Model Lineage

What is Model Lineage?

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 Inputs

Training Data

Code
- python
- Dockerfile
- Hyper-parameters

Packages & Libraries
- pandas
- numpy
- PyTorch
- Bert
- ...
- ...

Algorithm
- Docker Container Image
- Algorithm

EACH Input has a version

Model Artifact

{ v26 }

Model Metadata

(1) *Metric(s):*
- *Accuracy: 82.32%*

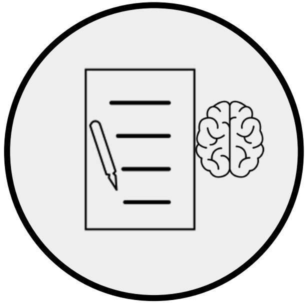
(1) Package Repository:
- *<location of model artifact>*

(1) Model Inputs

Model Lineage

Model Registry

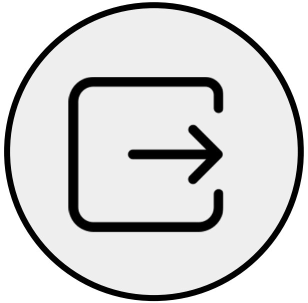
What is a Model Registry?



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

Artifact Tracking

What is Artifact Tracking?

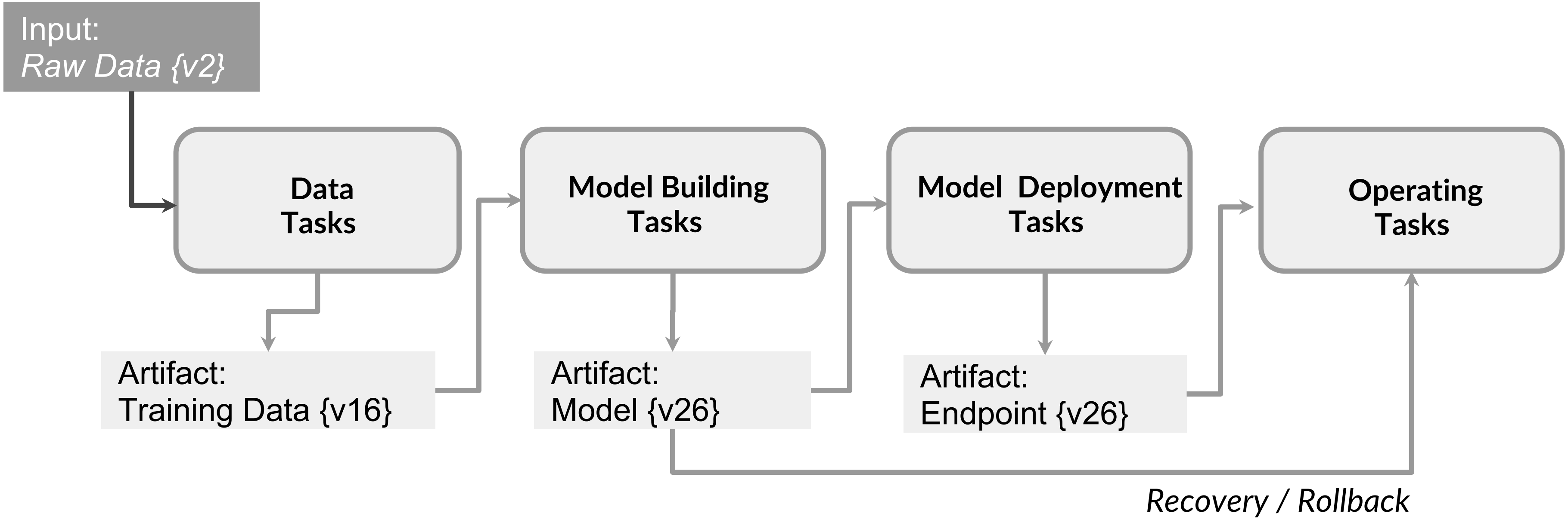


- An **Artifact** is the output of a step or task can be consumed by the next step in a pipeline or deployed directly for consumption

Artifact Tracking

Pipeline Manifest

Example:

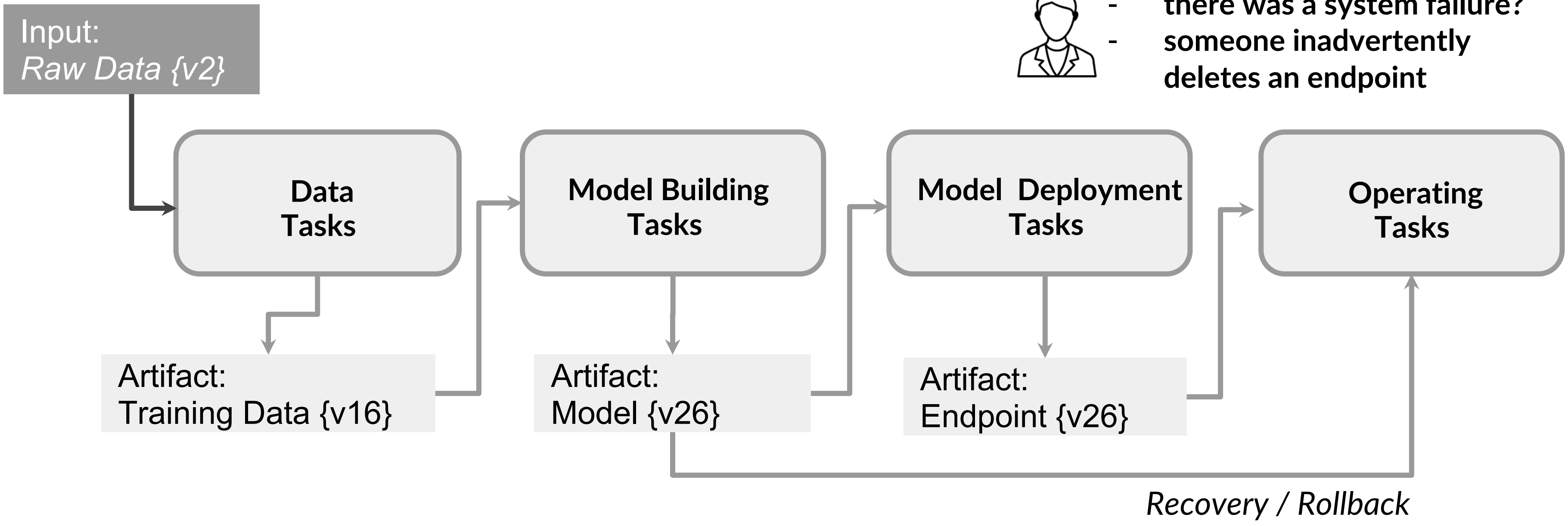


Artifact Tracking

Pipeline Manifest - Why it matters

Example:

What if ...
- there was a system failure?
- someone inadvertently deletes an endpoint



Machine Learning Pipelines

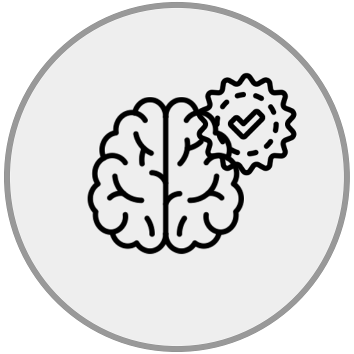
with
Amazon SageMaker Pipelines

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

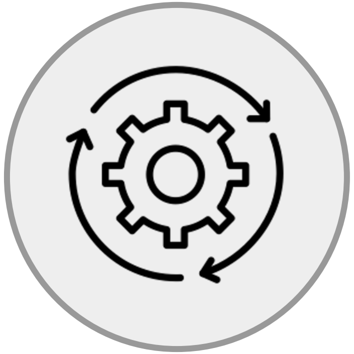
Amazon SageMaker Pipelines

Create & visualize automated workflows



Choose the best performing model to deploy

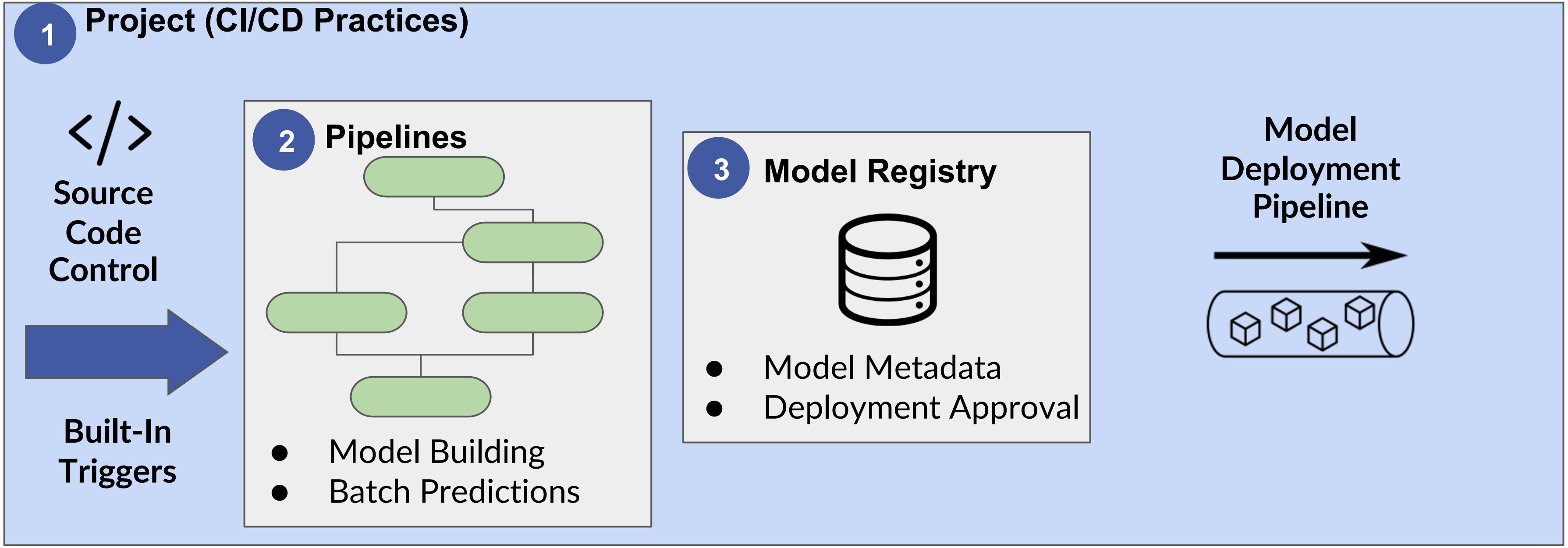
Automatic tracking of models



Bring CI/CD to Machine Learning

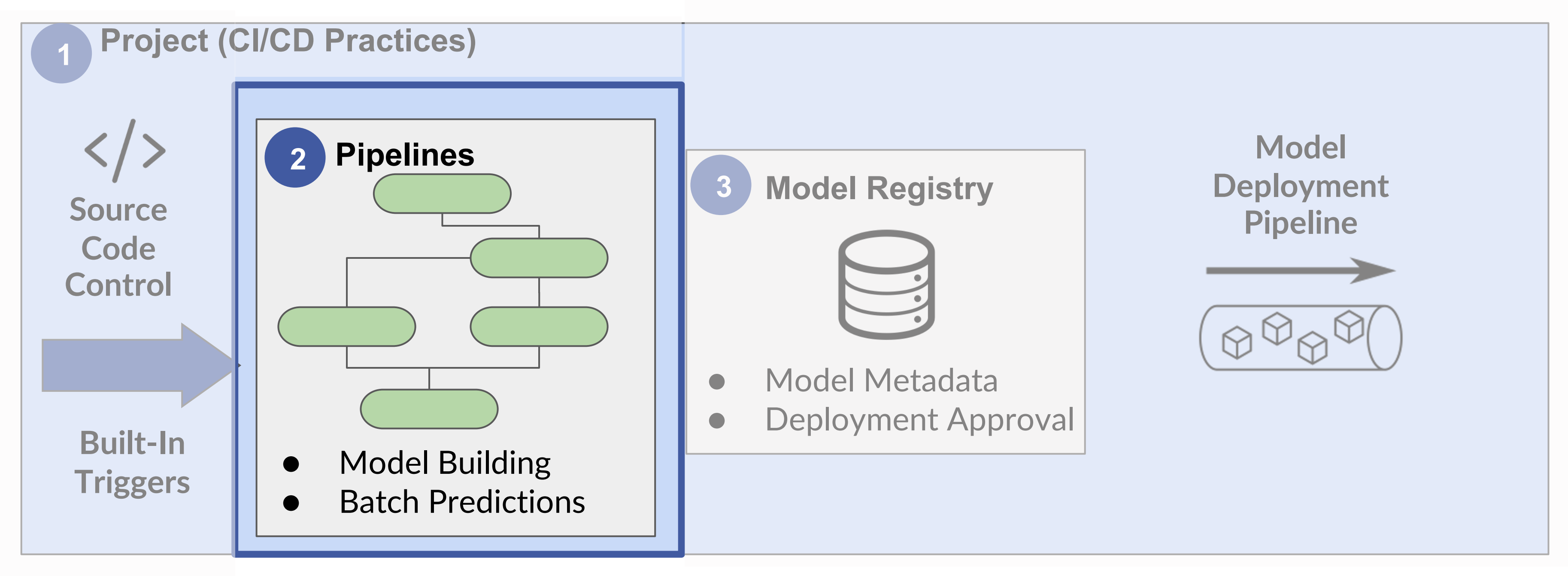
Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ….



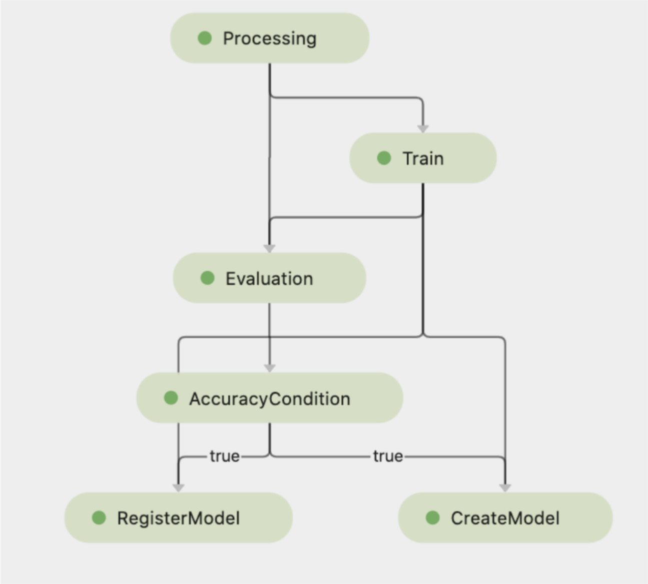
Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ….



SageMaker Pipelines

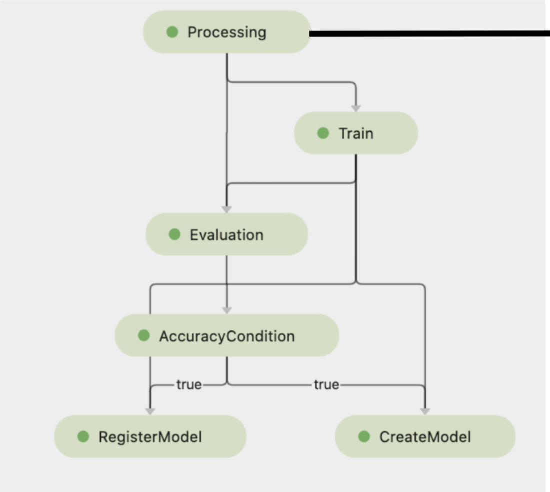
Pipelines



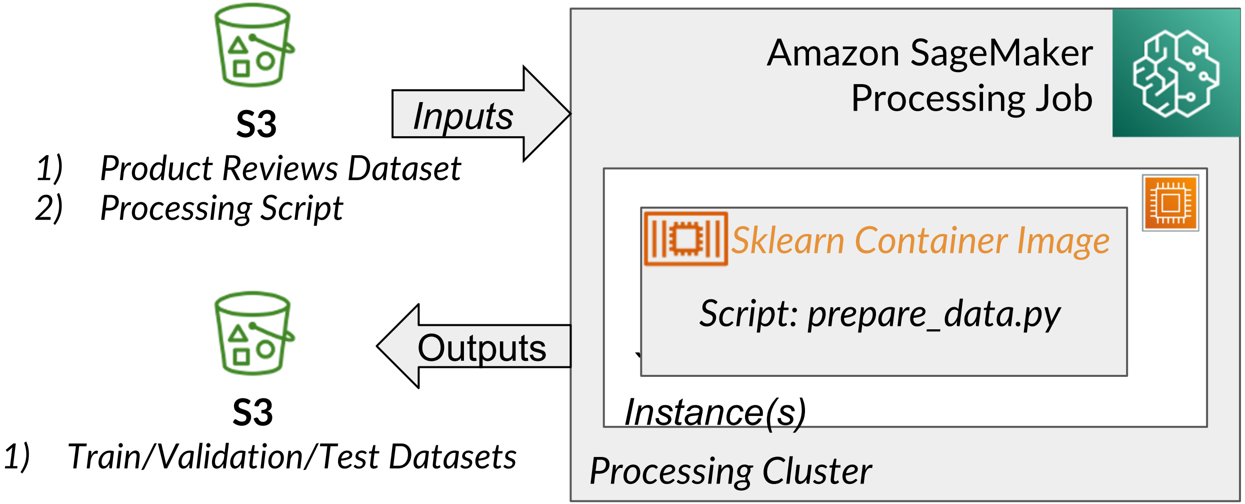
- ❏ Create Pipelines to build and evaluate models
- ❏ Python SDK for building workflows
- ❏ Pipeline visualization available through Amazon SageMaker Studio
- ❏ Fully managed pipelines - no servers to manage

SageMaker Pipelines

Processing Step

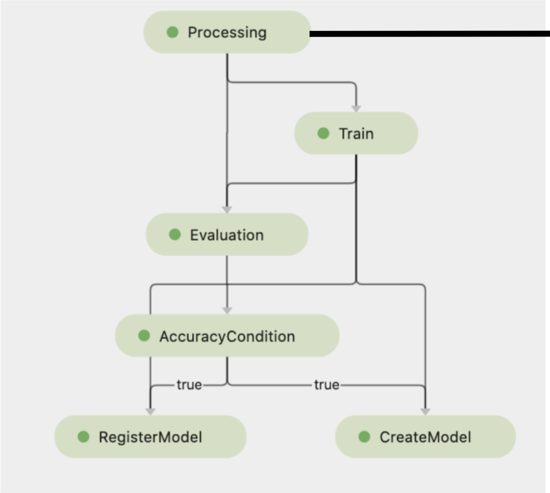


Use Amazon SageMaker Processing to process data set for training →



SageMaker Pipelines

Processing Step

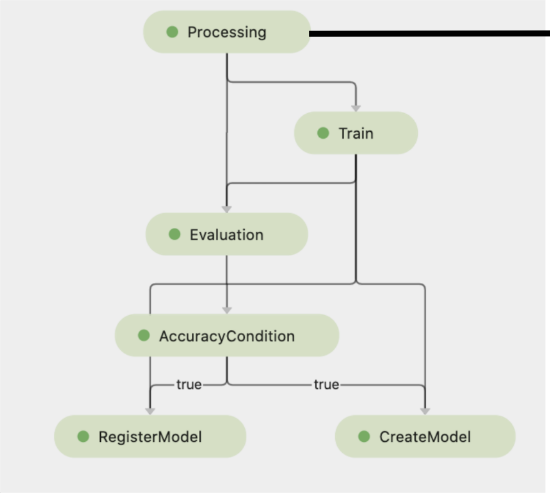


Define Step Inputs & Outputs →

```
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

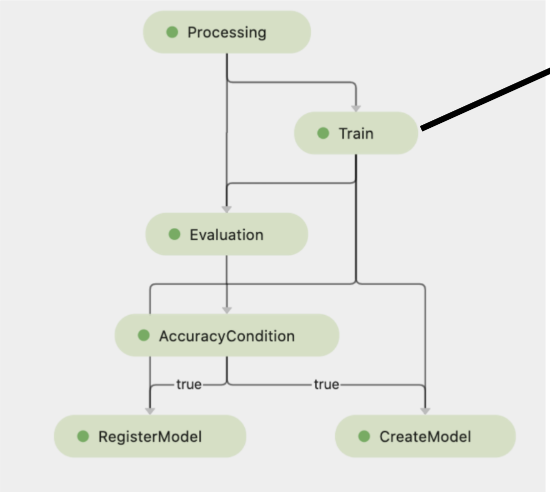


Configure the Processing Step →

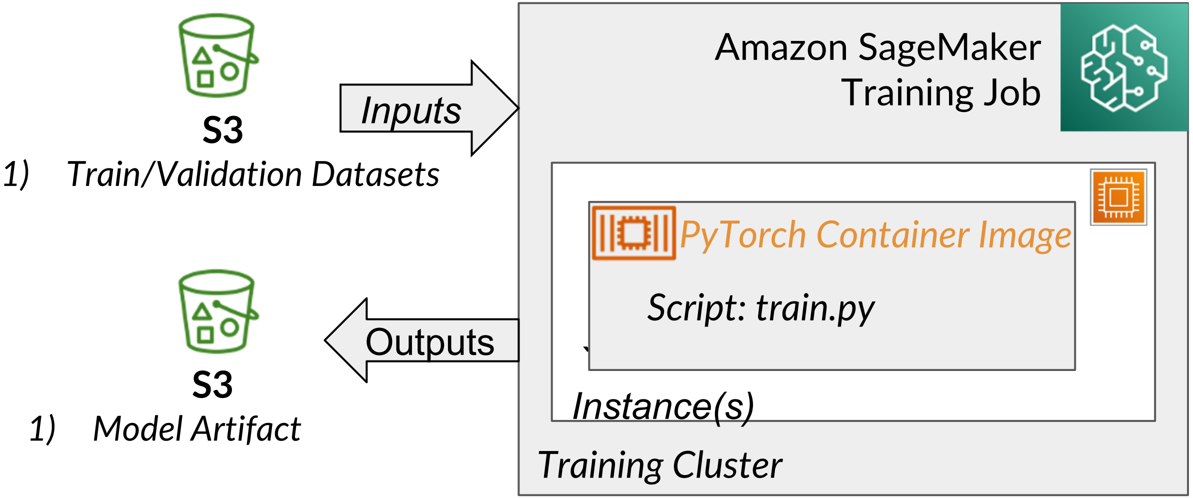
```
processing_step = ProcessingStep(
            name='Processing',
            code='src/prepare_data.py',
            processor=processor,
            inputs=processing_inputs,
            outputs=processing_outputs,
            job_arguments=[
            '--train-split-percentage',
            str(train_split_percentage.default_value,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

SageMaker Pipelines

Training Step

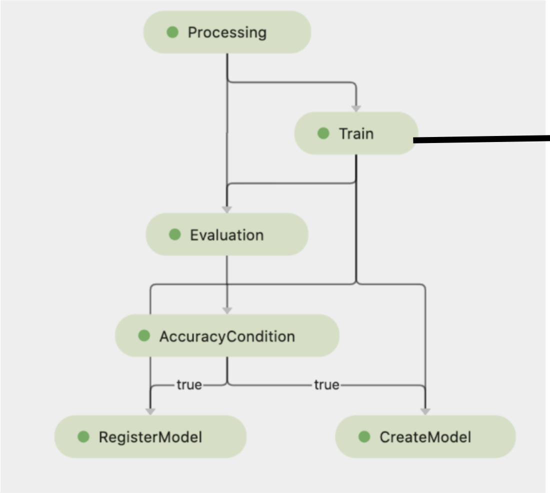


Use Amazon SageMaker Training Jobs to train the model using the outputs from the previous step as input →



SageMaker Pipelines

Training Step

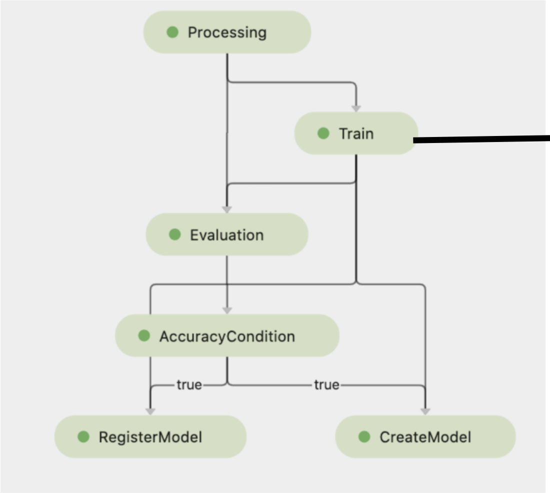


Configure Hyperparameters →

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

SageMaker Pipelines

Training Step

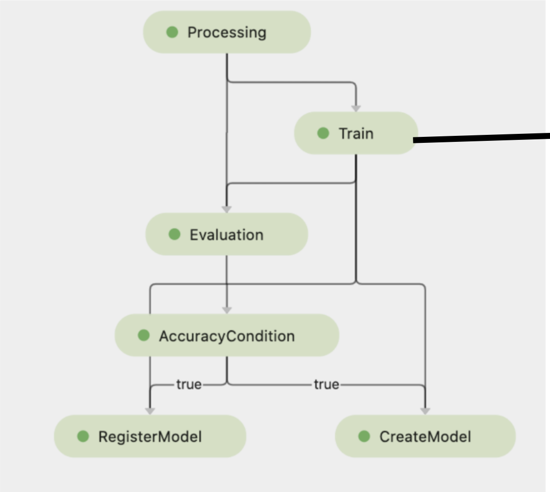


Configure Estimator →

```
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



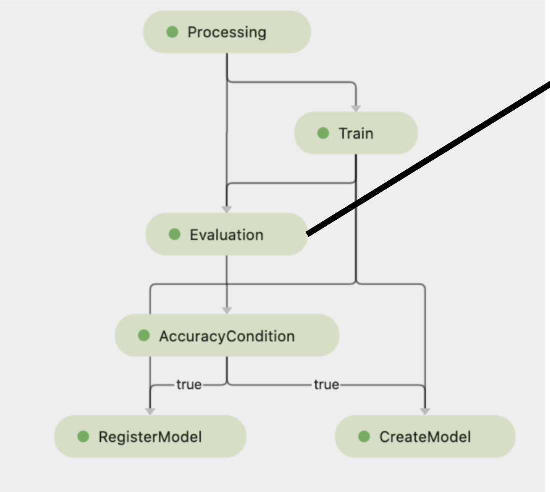
Configure the Training Step →

Output from
Processing Step

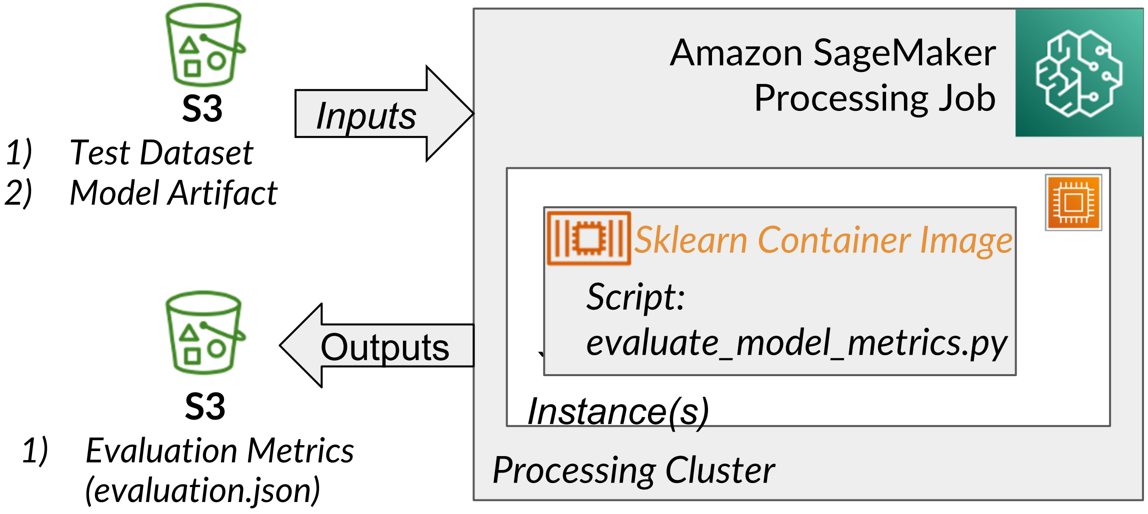
```
training_step = TrainingStep(
          name='Train',
          estimator=estimator,
          inputs={
          'train': TrainingInput(
          s3_data=processing_step.properties.ProcessingOu
tputConfig.Outputs[
                    'sentiment-train'
          ].S3Output.S3Uri,
          content_type='text/csv'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

SageMaker Pipelines

Evaluation Step

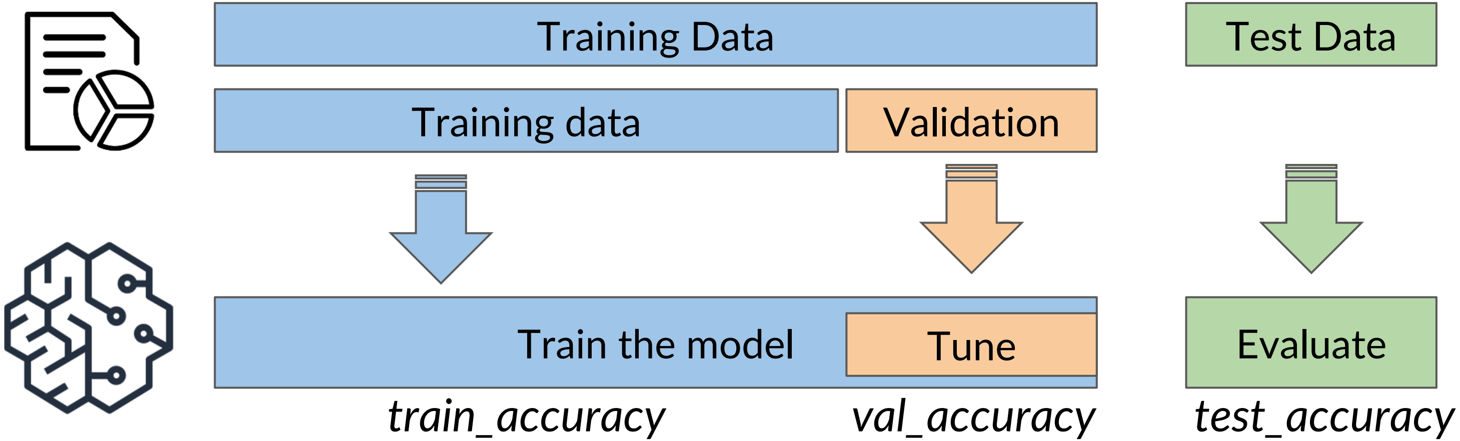


Use Amazon SageMaker Processing to evaluate trained model using test holdout dataset →



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

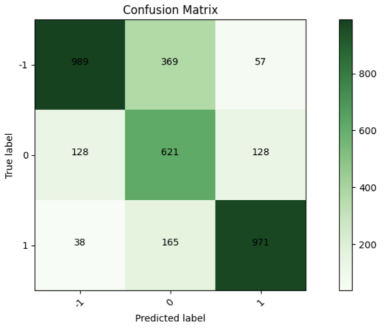
Use "test" holdout data

Calculate test accuracy

Analyze results

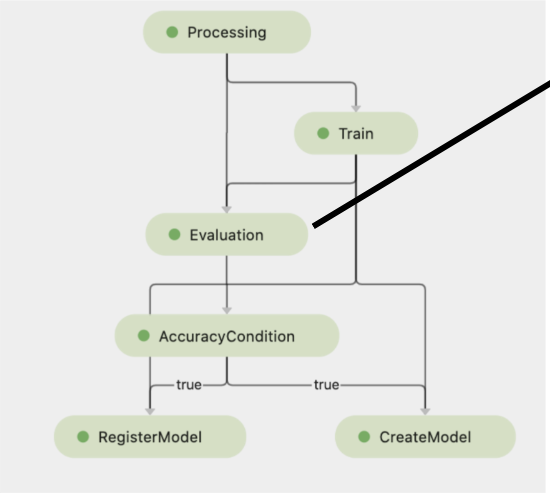
```python
from pprint import pprint

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

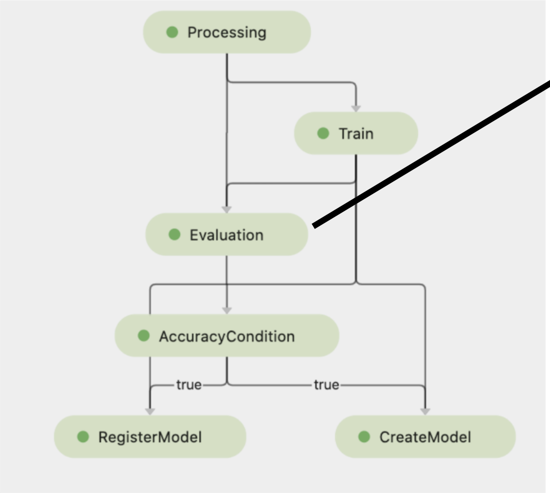


Define Output →

```
from sagemaker.workflow.properties import
PropertyFile

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

SageMaker Pipelines

Evaluation Step

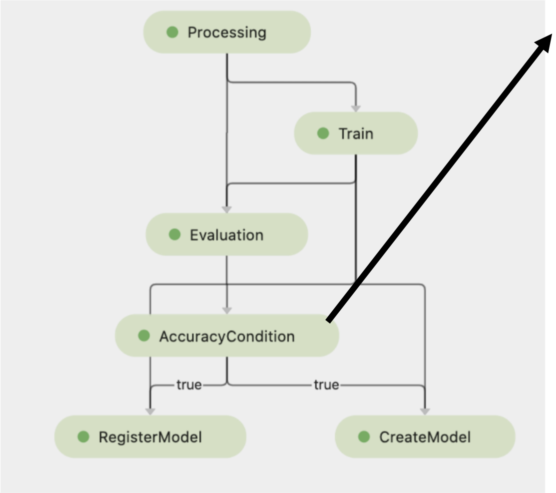


Configure Processing Step →

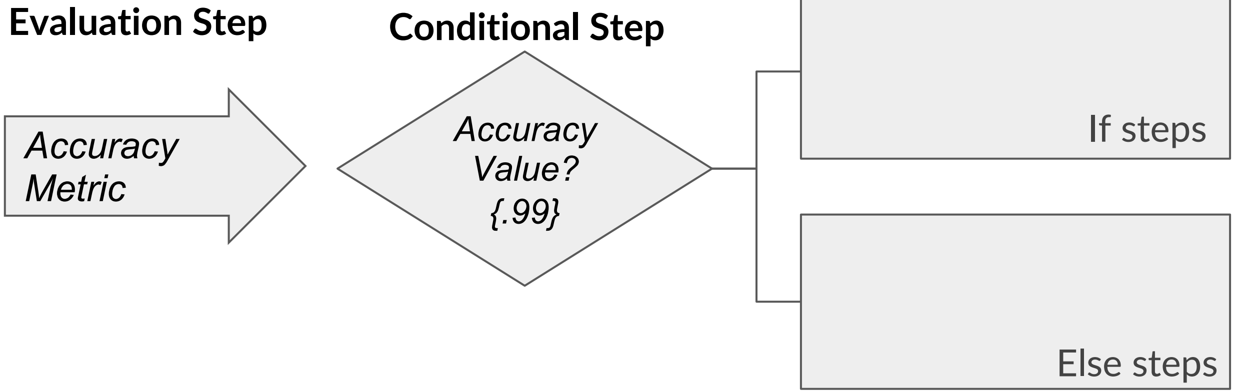
```
evaluation_step = ProcessingStep(
                name='EvaluateModel',
                processor=evaluation_processor,
                code='src/evaluate_model_metrics.py',
                inputs=[
                ProcessingInput(...),...
                ],
                outputs=[
           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

SageMaker Pipelines

Condition Step

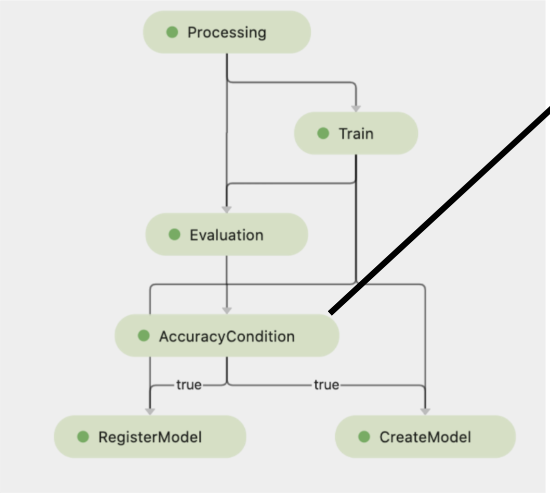


Use Amazon SageMaker Pipelines *Condition Step* to conditionally execute step(s) →



SageMaker Pipelines

Condition Step



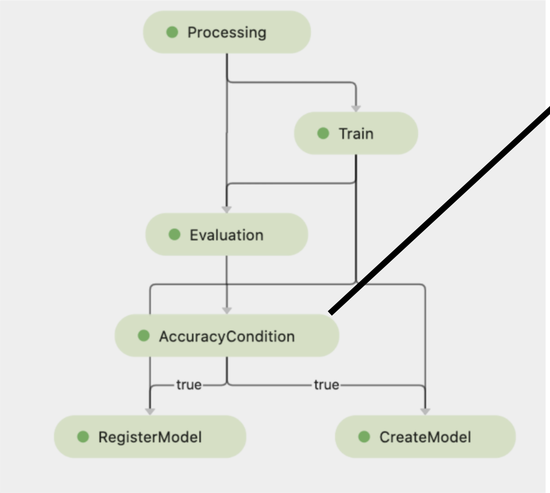
Define a Condition & Import Conditional Workflow Step →

```
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

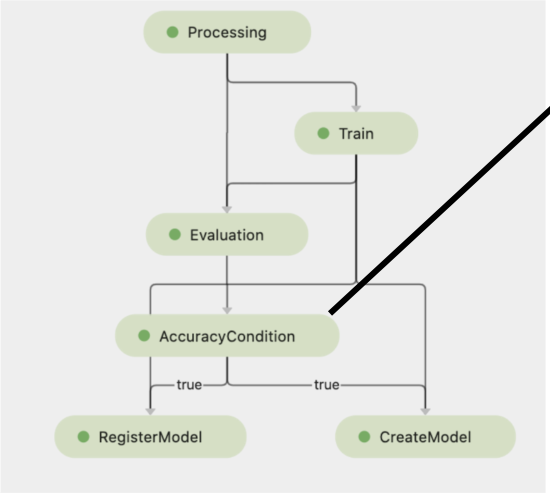


Configure the Condition →

```
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

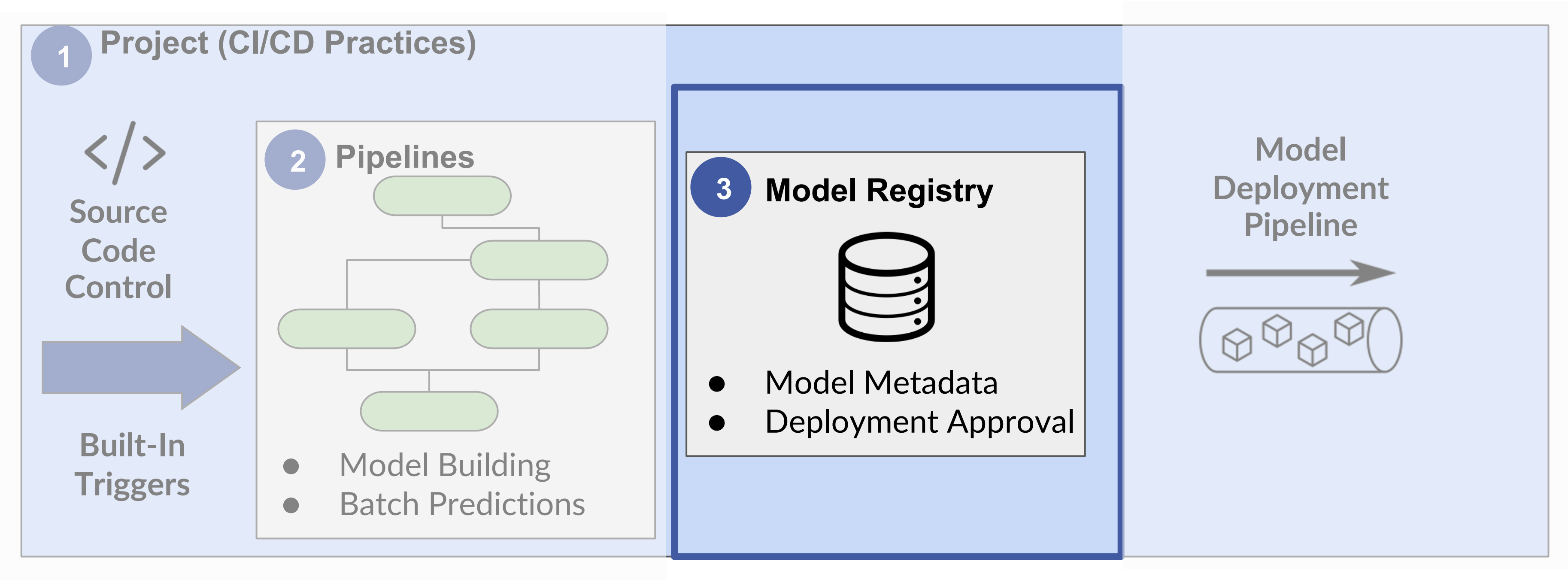


Configure the Condition Step →

```
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 ....



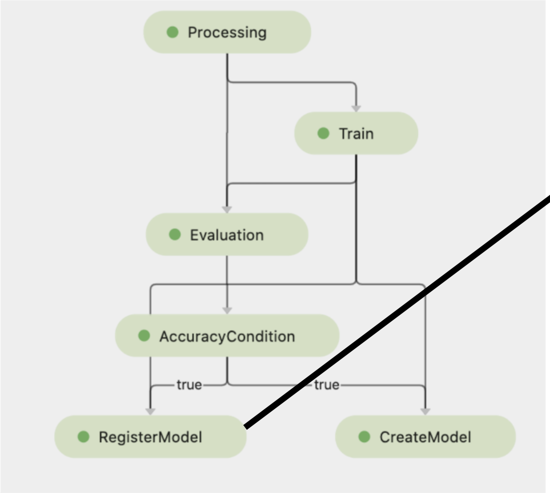
SageMaker Pipelines

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

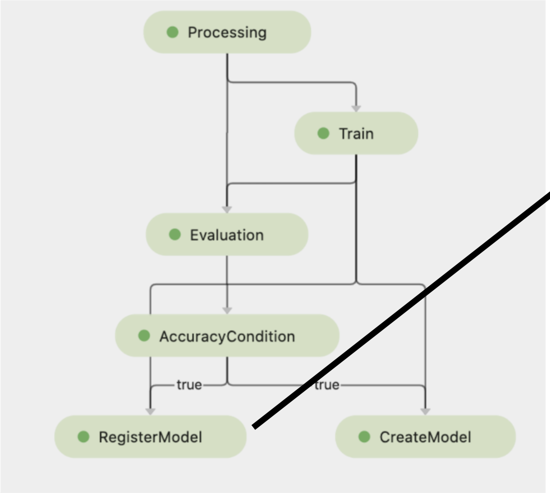


Define deployment image for inference →

```
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

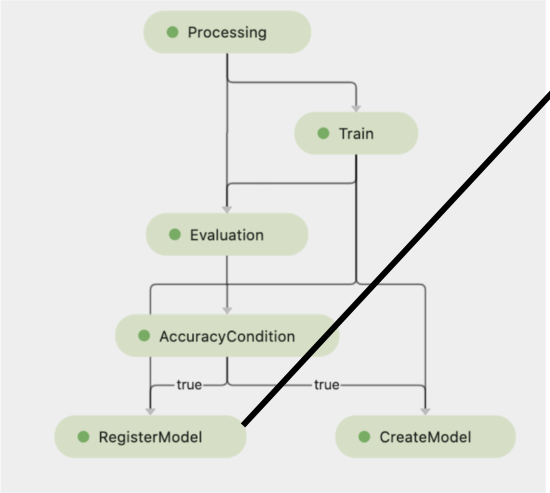


Define model metrics to be stored as metadata →

```
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

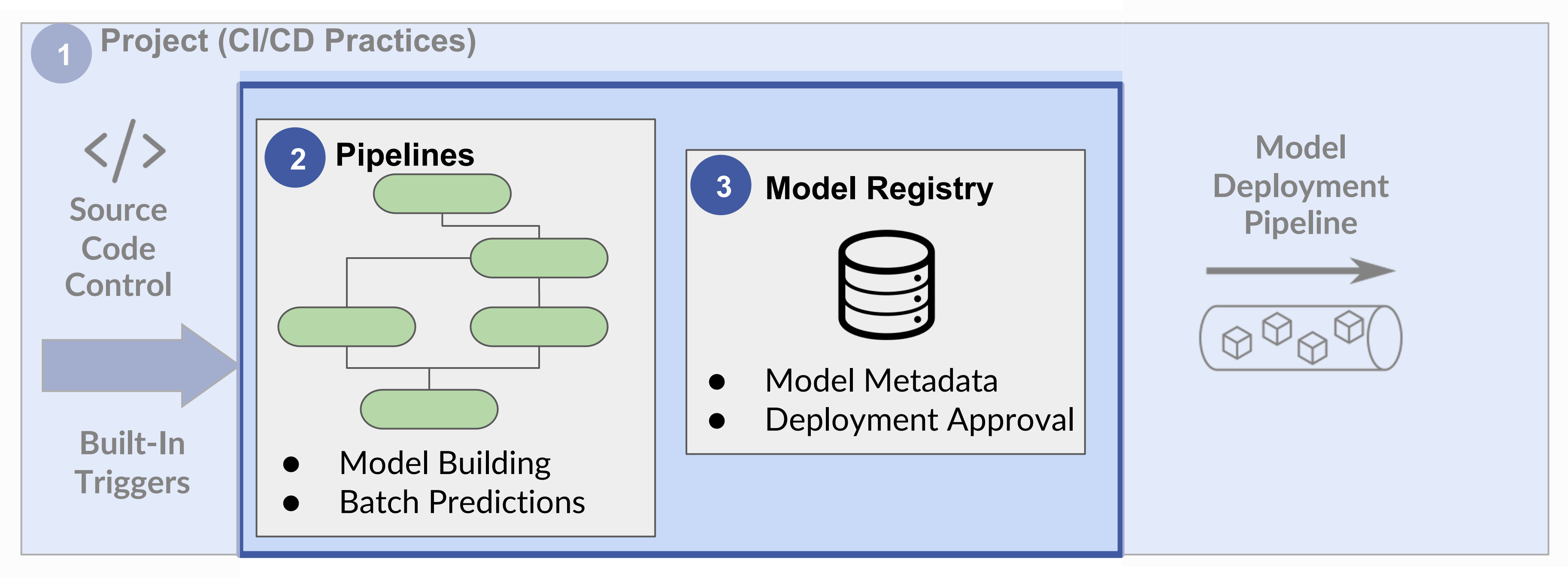


Configure the Register Model Step →

```
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)
```

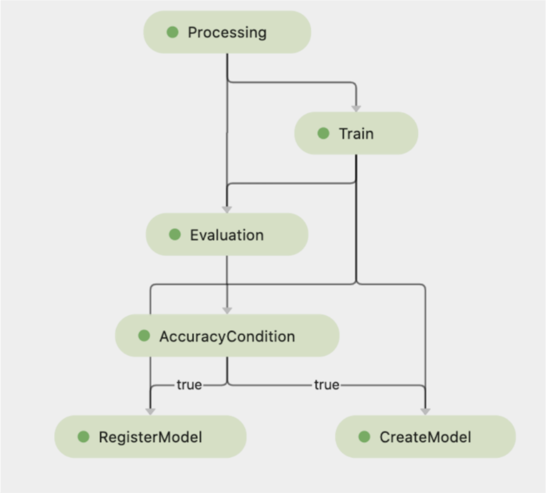
Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ….



SageMaker Pipelines

Bringing It All Together

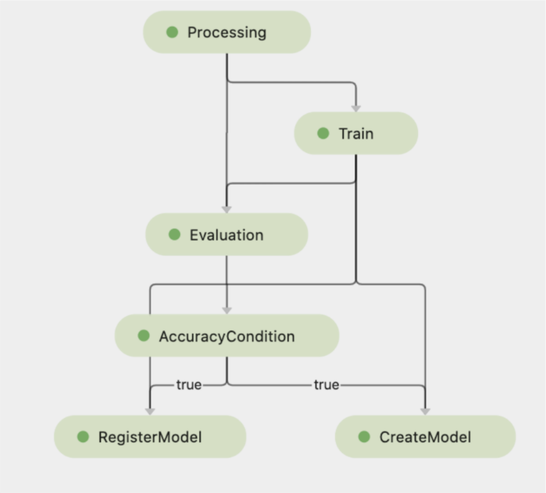


Configure the Pipeline →

```
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 →

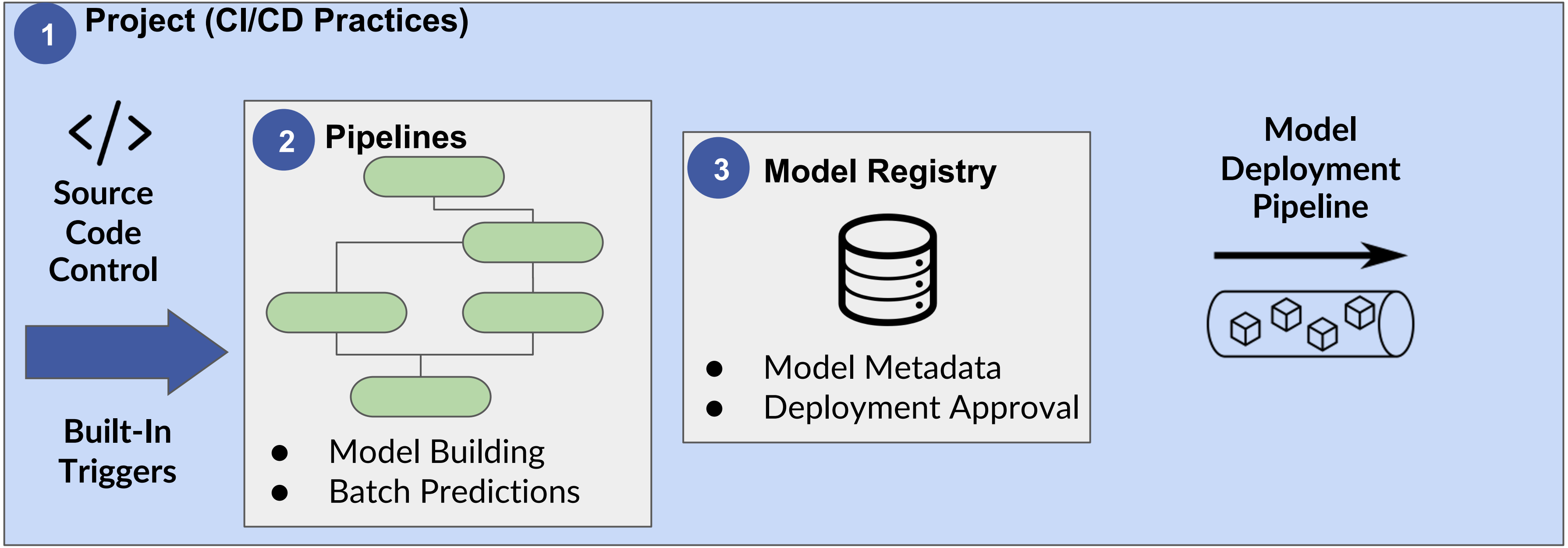
```
response = pipeline.create(role_arn=role)

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