ModelOps

A programming model for reusable, platform-independent, and composable AI workflows

Waldemar Hummer, Vinod Muthusamy

IBM Research AI

IBM Programming Languages Day
T.J. Watson Research Center
December 10, 2018
https://ibm.biz/plday2018

Contact: whummer@ibm.com

Team:
Yorktown: Waldemar Hummer, Anupama Murthi, Kaoutar El Maghraoui, Benjamin Herta, Darrell Reimer, Punleuk Oum, Gaodan Fang
Austin: Vinod Muthusamy
Cambridge: Scott Boag

A principled approach to operationalizing AI in business apps

AI models embedded in complex lifecycles

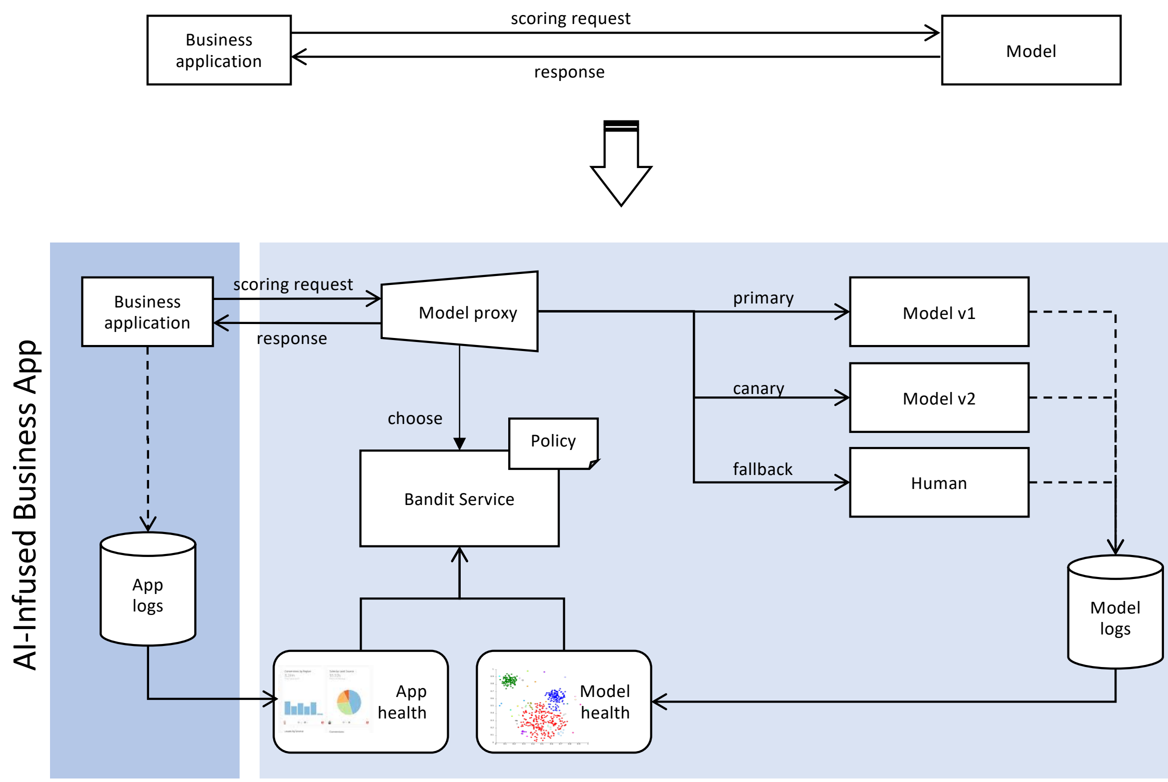
- Model versions matching business apps
- Monitoring and drift detection
- Active learning with human in the loop

Lots of moving parts

- Model and business KPIs
- Application and model logs
- Model proxies, evolving policies, …

Need a principled approach

- AI-aware staged deployments
- Reusable patterns
    - Simple out-of-the-box solutions
    - Customizable for concrete use cases

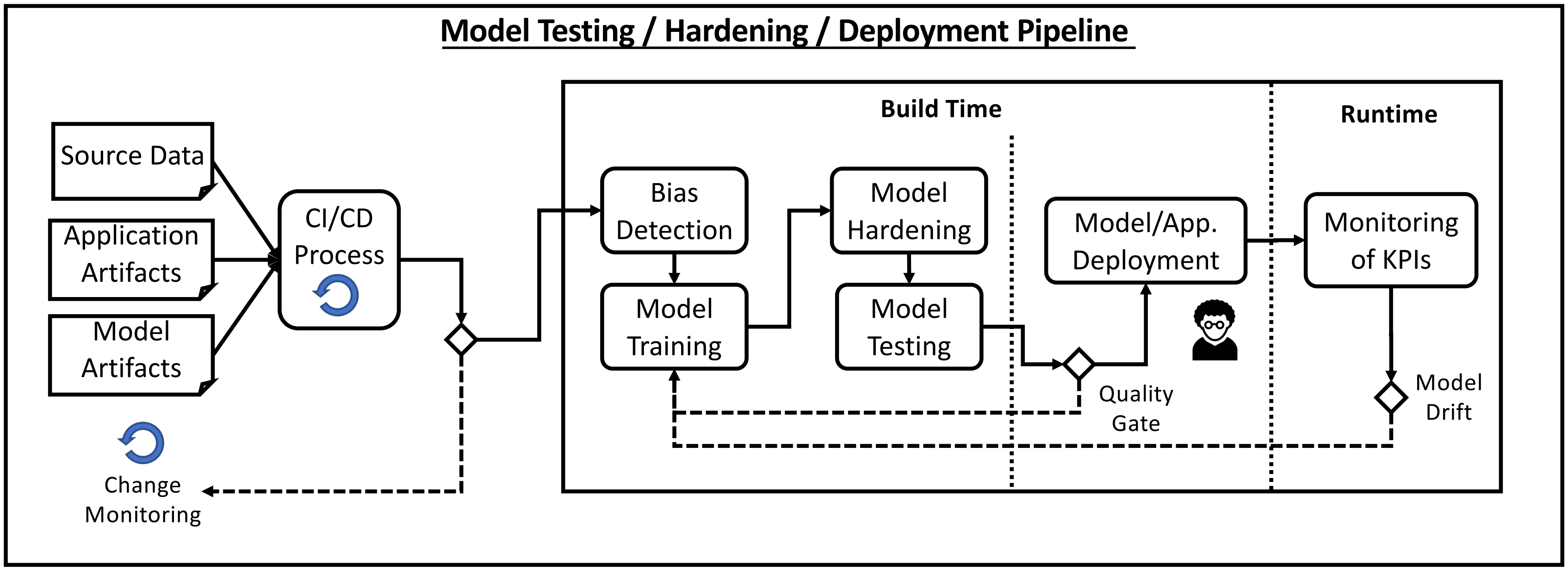


AI Models become engrained in the Software Lifecycle

- What's the difference to traditional DevOps?

Classical Application Lifecycle	AI Application Lifecycle
Requires dev / ops skills	Involves more diverse skill sets
Relatively short running	Long-running, resource intensive
Human speed (low change frequency)	Continuous (re-)training
Few versions of software artifacts	Huge number of models
Linear evolution of artifacts	Specialized models coexist
Configurations applied at runtime	Parameters tuned at training time
Codebase changes trigger new builds	Data/code changes trigger model re-training
Deterministic testing	Statistical/probabilistic testing
Monitoring of application performance & KPIs	Monitoring of model accuracy, drift, and KPIs

AI Operations – Pipeline Scenario



- **AI models** introduce **risk** to business applications
- Training and deployment pipelines can become quite **complex**
- Pipelines often **hand-crafted**
    - Hard to maintain
    - Hard to optimize or reason over

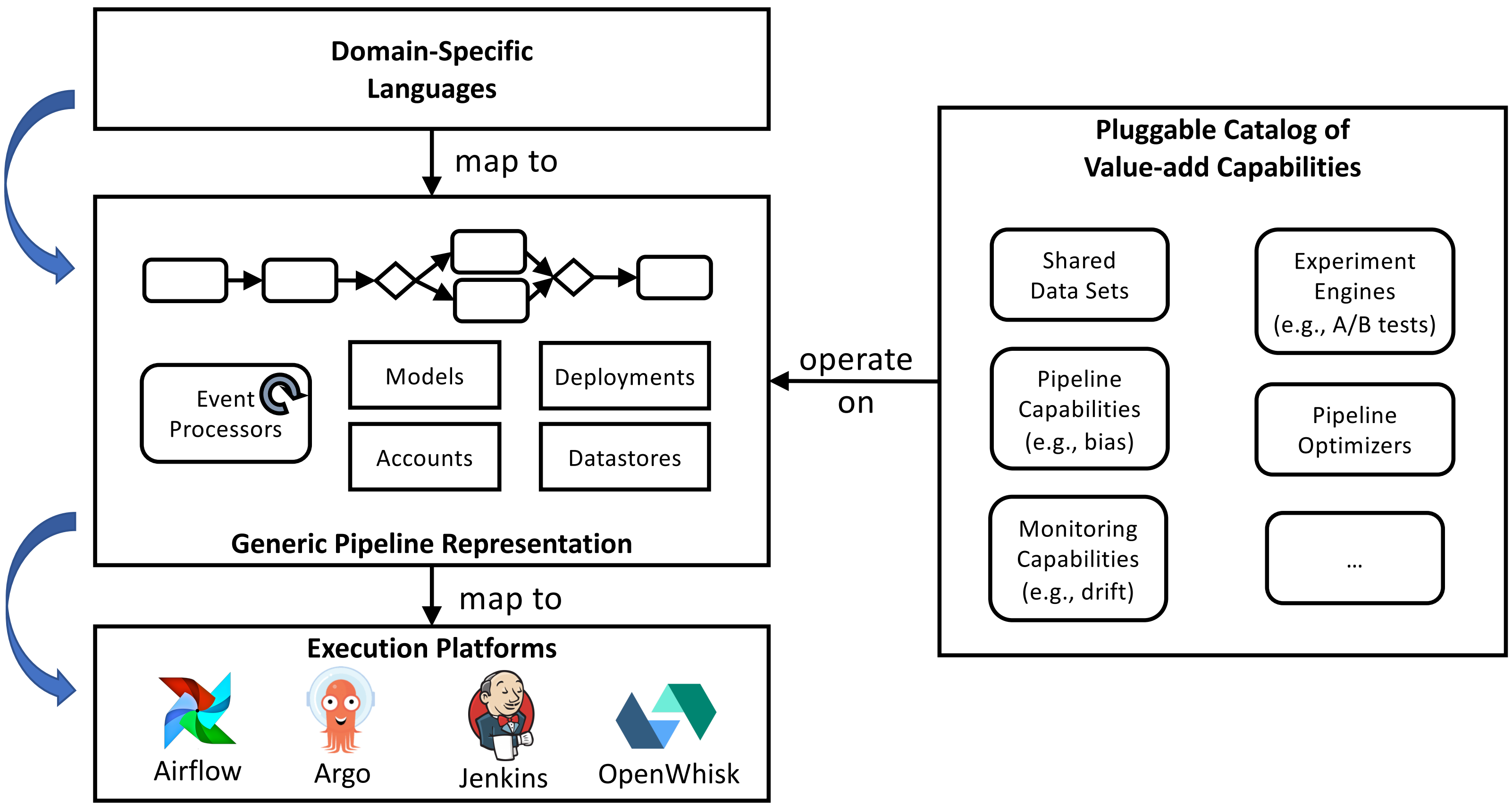
Core Problems Addressed

- **Reusability**
  - Pipelines are often hand-crafted (shell scripts, Makefiles, Python scripts, …)
  - Yet, there are common patterns: artifacts could be shared across users and teams
- **Composability**
  - Despite the common patterns: hardly a one-size-fits-all solution
  - Ability to compose pipelines and other features from building blocks is critical
- **Platform Independence**
  - Different Cloud vendors use different APIs – yet very similar concepts
    - E.g., "train model", "deploy model" → map to different API calls in WML / AWS / GCP
  - Parts of pipelines can run on various different environments
    - On local machines, on smart phones, in the cloud, on edge devices, HPC clusters, …

- **→ Ability to Optimize and Reason over AI Pipelines**
  - E.g., Task pruning, task co-location, task scheduling

Approach

ModelOps: Towards a programming model for Operationalizing AI



Generic AI Pipeline Representation

- Metamodel of apprx. 25 entities
  - Pipelines (DAG of tasks with dependencies)
  - Models (AI Classifier)
  - Event Triggers / Subscribers
  - Datastores
  - Accounts
  - Environments
  - Code Plugins
  - …
- Entites implemented as Python classes
  - Attributes defined in JSON Schema
  - Stored in markup files (e.g., YAML, JSON)
- Language features
  - Plugin Imports
  - Variable placeholders (lazy evaluation)
  - Control flow (pipelines DAG)

```yaml
accounts:
  - name: account_wml
    type: ibm/wml
    password: '{{credentials.wml.password}}'
    username: '{{credentials.wml.username}}'
    instance_id: '{{credentials.wml.instance_id}}'
datastores:
  - name: training_data
    type: cos
    bucket: trainint-data
    account: s3_account
models:
  - name: model1
    type: wml
    command: 'python train.py'
    framework_name: tensorflow
    source: training_data
    target: model_results
subscribers:
  - _type: plugins.templates.DriftDetector
    name: drift_detector_model1
pipelines:
  - name: train-deploy-model
    tasks:
      - _type: modelops.tasks.TrainModel
        name: TrainModel
        model: model1
      - _type: modelops.tasks.StoreModel
        name: StoreModel
        model: model1
      - _type: modelops.tasks.DeployModel
        name: DeployModel
        model: model1
```

Extensibility via Plugins

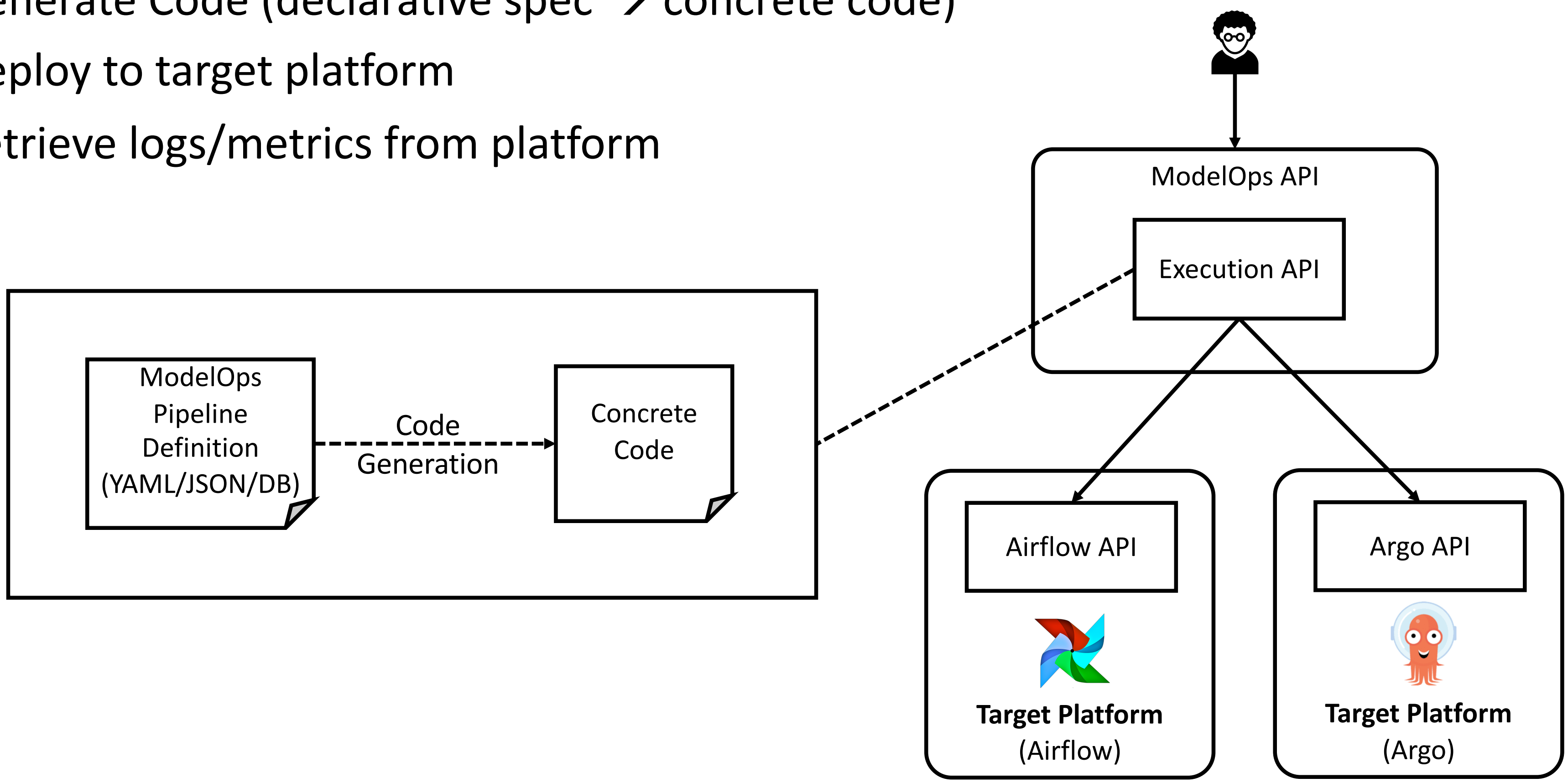
- Extensions as Plugins
  - Pipeline tasks
    - E.g. “Train Model”
  - Subscribers
    - E.g., “Detect Drift”
  - Transformers
    - (See following slides)

- Plugins can be referenced in the config
  - Automatically loaded into Python path at runtime

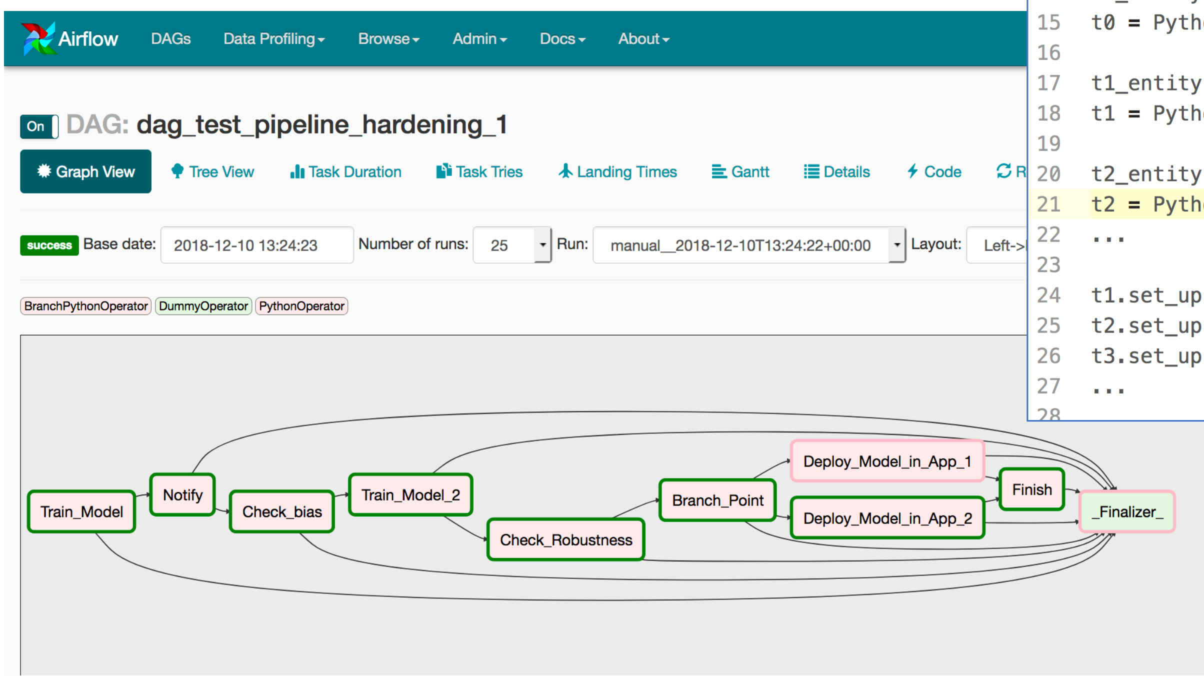
```python
class PipelineTask(EmbeddedEntity):
    """ Represents a task in a pipeline. """

    def execute(self, **params):
        """ Main method that starts the execution of this task. Implemented by subclasses. """
        raise NotImplementedError()

    def finished(self, execute_result, **params):
        """ Return the final result of this task, or False if this task has not finished yet.
            This method receives as parameter the return value of the "execute" method.
            By default, we assume that "execute" runs synchronously, and hence this method
            returns a truthy value by default. Subclasses need to overwrite this method with
            specific logic that polls for task completion. """
        return execute_result or True

    def poll_interval(self, **params):
        """ Interval in seconds to apply when polling for results. If this value is 0
            (default), then this is a synchronous task that is finished after "execute(..)"
            returns (i.e., no polling required). If this value is a negative number,
            then let the task executor pick a suitable polling interval. """
        return -1

    def get_timeout(self, **params):
        """ Return the timeout interval (in seconds) after which this task execution should
            be considered failed. Returns the value of the "timeout" property of this task
            instance (if present), or 30 minutes by default. Subclasses can overwrite this method.
            If a negative or non-integer value is provided, let the task executor choose a timeout. """
        return self.timeout if is_positive_number(self.timeout) else self.DEFAULT_TIMEOUT
```

Pipeline Executor

1. Generate Code (declarative spec → concrete code)
2. Deploy to target platform
3. Retrieve logs/metrics from platform



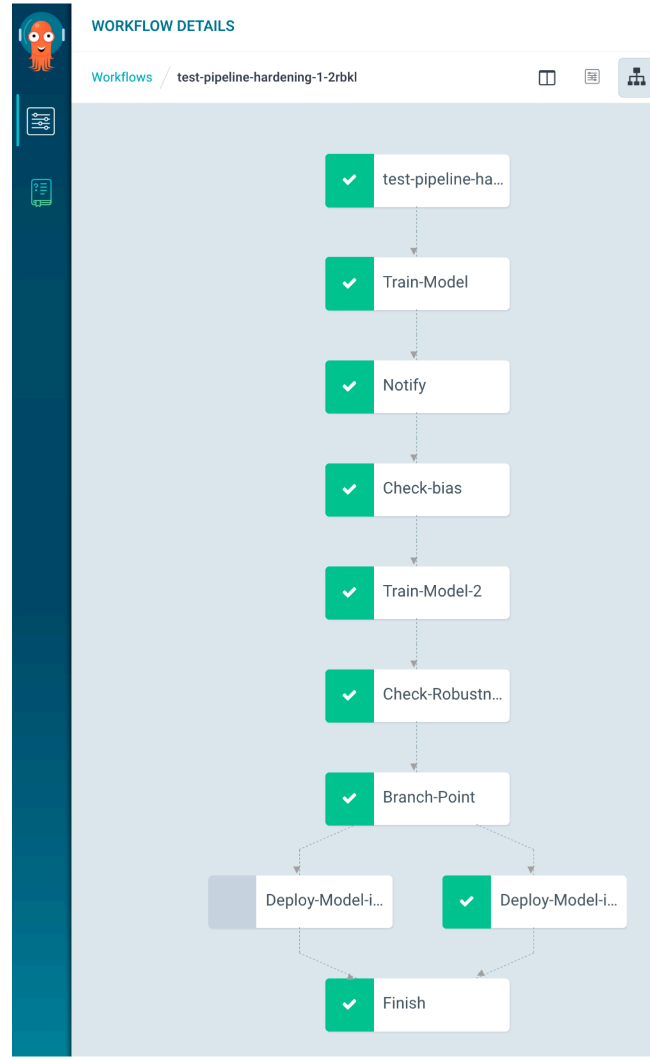
Generated Pipeline Code – Airflow




```python
import json
from airflow import DAG
...

def run_task(task_entity, **params):
    from modelops import main
    ...

args = {
    'start_date': datetime.datetime.fromtimestamp(1544448249.35)
}
dag = DAG(dag_id='dag_test_pipeline_hardening_1', default_args=args

t0_entity = json.loads('''{"_type": "modelops.tasks.TrainModel", "e
t0 = PythonOperator(task_id='Train_Model', python_callable=run_task

t1_entity = json.loads('''{"_type": "modelops.tasks.PostMessage", "
t1 = PythonOperator(task_id='Notify', python_callable=run_task, op_

t2_entity = json.loads('''{"_type": "modelops.tasks.DataBiasCheck",
t2 = PythonOperator(task_id='Check_bias', python_callable=run_task,
...

t1.set_upstream(t0)
t2.set_upstream(t1)
t3.set_upstream(t2)
...
```

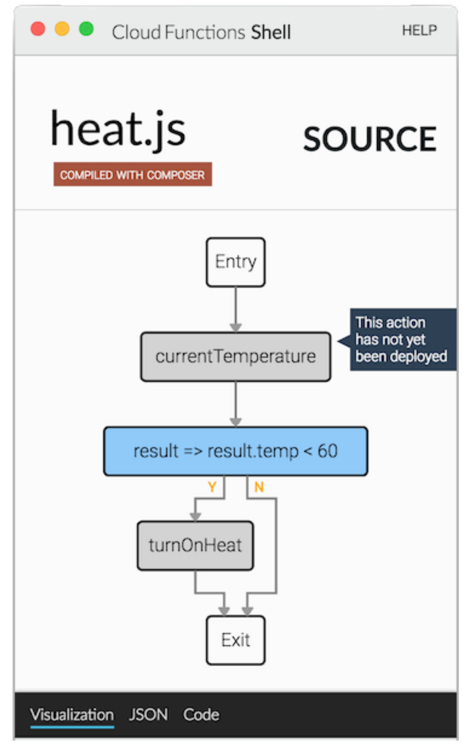
Generated Pipeline Code – Argo

```
name: test-pipeline-hardening-1
dag:
  tasks:
    - arguments: {parameters: [{name: task, value: eyJfdHwcy5wbH
      name: Train-Model
      template: run-task
    - arguments: {parameters: [{name: task, value: eyJfdHyNrLlBv
      dependencies: [Train-Model]
      name: Notify
      template: run-task
    - arguments: {parameters: [{name: task, value: eyJfcy5wJpYXM
      dependencies: [Notify]
      name: Check-bias
      template: run-task
    - arguments: {parameters: [{name: task, value: eyJfcy5wbLmFw
      dependencies: [Check-bias]
      name: Train-Model-2
      template: run-task
    - arguments: {parameters: [{name: task, value: eyJfdHlwZSI6I
      dependencies: [Train-Model-2]
      name: Check-Robustness
      template: run-task
    - arguments: {parameters: [{name: task, value: eyJfluZV9icmF
      dependencies: [Check-Robustness]
      name: Branch-Point
      template: run-task
```



Generated Pipeline Code – OpenWhisk Composer

```
const composer = require('@ibm-functions/composer/composer')
const conductor = require('@ibm-functions/composer/conductor')
const pipeline_name = 'test_pipeline_hardening_1'
const wsk = conductor({ignore_certs: true})

const composition = composer.sequence(
    composer.sequence("test_pipeline_hardening_1_Train_Model",
      composer.dowhile_nosave((tmp) => (tmp),
        composer.sequence("test_pipeline_hardening_1_Train_Model_poll", (r)
    ),
    composer.sequence("test_pipeline_hardening_1_Notify",
      composer.dowhile_nosave((tmp) => (tmp),
        composer.sequence("test_pipeline_hardening_1_Notify_poll", (r) => (
    ),
    composer.sequence("test_pipeline_hardening_1_Check_bias",
      composer.dowhile_nosave((tmp) => (tmp),
        composer.sequence("test_pipeline_hardening_1_Check_bias_poll", (r)
    ),
    composer.sequence("test_pipeline_hardening_1_Train_Model_2",
      composer.dowhile_nosave((tmp) => (tmp),
        composer.sequence("test_pipeline_hardening_1_Train_Model_2_poll", (
    ),
    composer.sequence("test_pipeline_hardening_1_Check_Robustness",
      composer.dowhile_nosave((tmp) => (tmp),
        composer.sequence("test_pipeline_hardening_1_Check_Robustness_poll"
    ),
    composer.if("test_pipeline_hardening_1_Branch_Point",
```



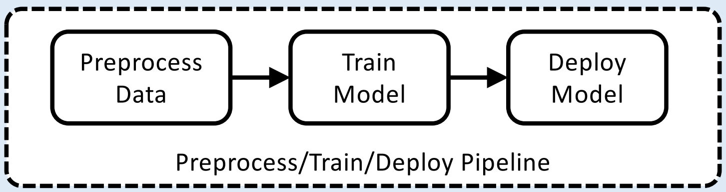
Example Image, taken from:
https://www.ibm.com/blogs/bluemix/2017/10/serverless-composition-ibm-cloud-functions

Composable Pipeline Templates

Pipeline Templates

→ Parameterizable templates of common pipeline patterns

- **Example:** Preprocess/Train/Deploy Pipeline
  - Easily bootstrap configurations with default values
  - Users can customize and fine-tune the configuration



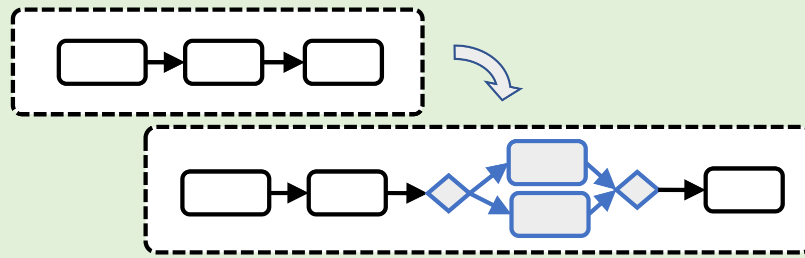
Extensible Catalog of Common Patterns

- Composability becomes critical
- Leverage techniques from BPM, service composition, and configuration management

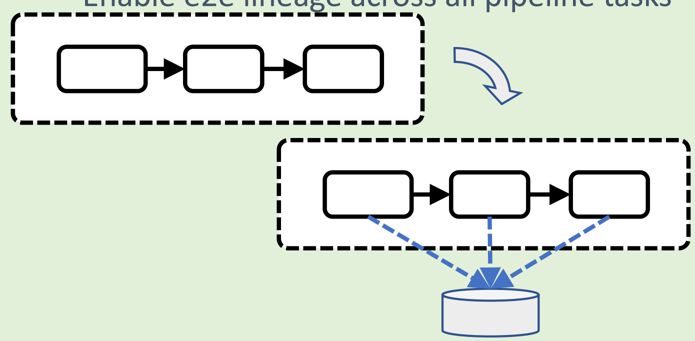
Pipeline Transformers

→ Enrich existing configurations with additional features

- **Example:** Add safeguards for deployment
  - A/B testing with feedback and human in the loop



- **Example:** Plug in cross-cutting features for trusted AI
  - Enable e2e lineage across all pipeline tasks

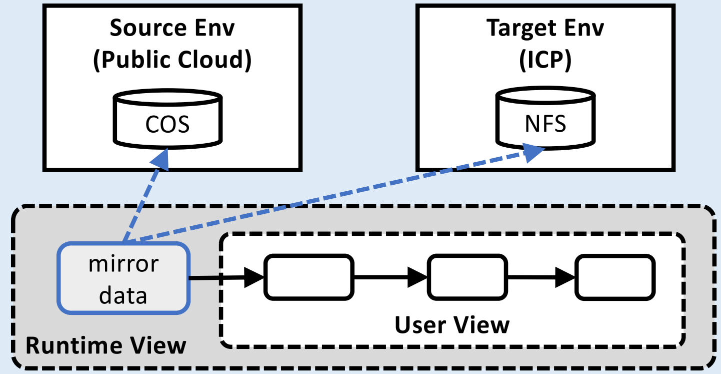


Simple Demo – Pipeline Templates

Reusable domain abstractions

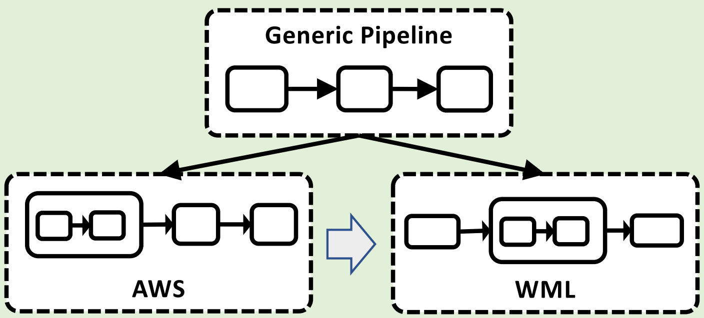
Use domain knowledge to build smarts into the pipeline

Example:
- Seamlessly move between Public Cloud and ICP
- Automatically enrich and adjust the pipeline at runtime
- Managing all artifacts required to run the pipeline



Platform-independent pipeline tasks

Example:
- Generic pipelines allow easy on-ramp into our platform



Rollout to mobile and distributed deployments

Use cases:
- VR models on mobile devices with automated retraining loop
- Patient-specific models for predicting hypoglocemia based on real-time data

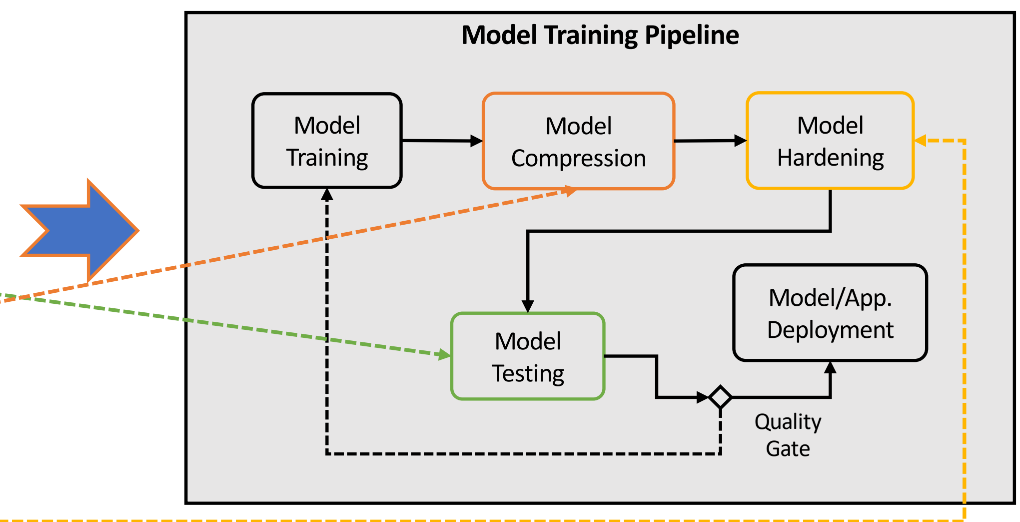
Testing and Fine-Tuning Classifiers

– **Context:** Model training entails multiple **specialized stages** (e.g., data bias checks[1], model robustness checks [1], model compression)

– **Problem:** Each stage requires custom configuration, input-output mapping, etc, which can be tedious

– **Solution:** Annotate the model entity with desired features – pipeline tasks will get inserted automatically

```yaml
models:
  - name: model1
    type: wml
    command: 'python train.py'
    framework_name: tensorflow
    source: training_data
    test_command: 'python test.py'
    compression:
      quantization_bits: 8
    robustness:
      defenses: {"threshold": 0.3}
```



[1] https://github.com/IBM/AIF360
[2] https://github.com/IBM/adversarial-robustness-toolbox

Staged Deployments

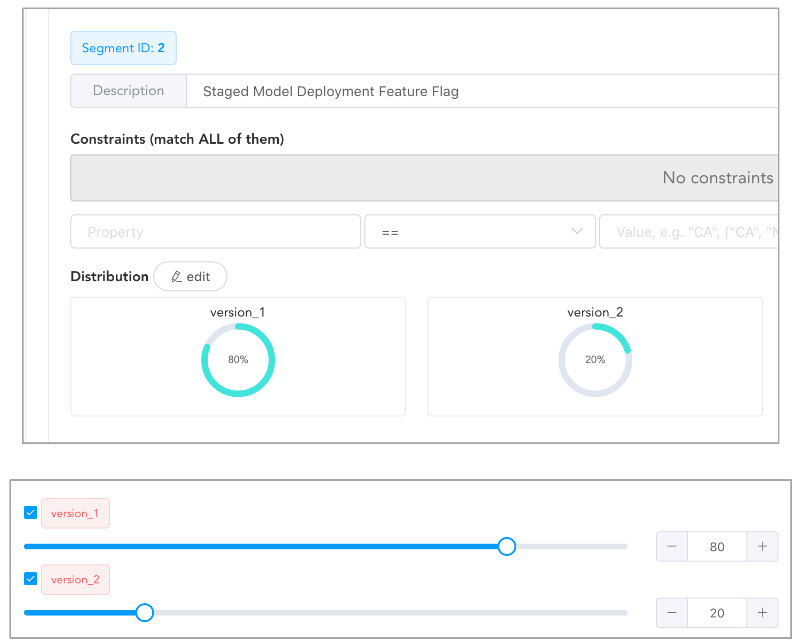
– **Context:** Models are deployed into business-critical applications

– **Problem:** Need a controlled way to deploy, compare, and update model versions

– **Solution:** Hide new model deployments behind feature flags, gradually roll out the change

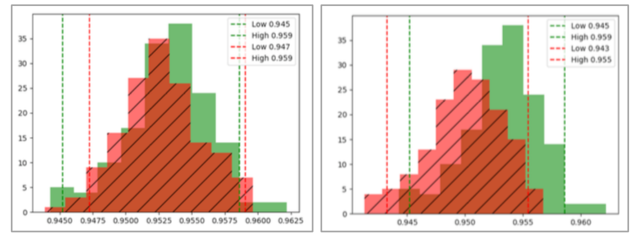
```yaml
experiments:
  - name: a_b_test
    settings:
      _type: modelops.routing.ABTest
      model: model1
      routing:
        type: fixed
        new_model: 0.2
        old_model: 0.8
      deployment:
        _type: modelops.routing.DeployToFlagr
        endpoint: {{flagr_url}}
```

A/B Testing experiment configuration



Deployed feature flag for different model versions

Model Drift Detection



– **Context:** Models operate in **dynamic environments**

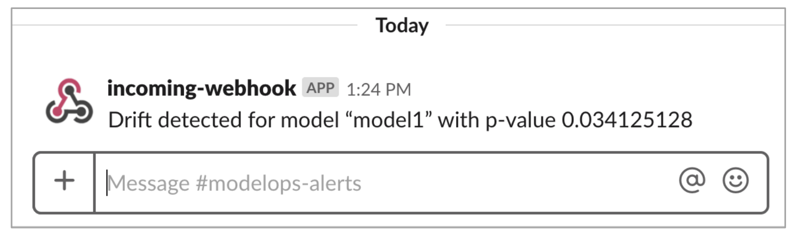
→ Data and conditions may change over time

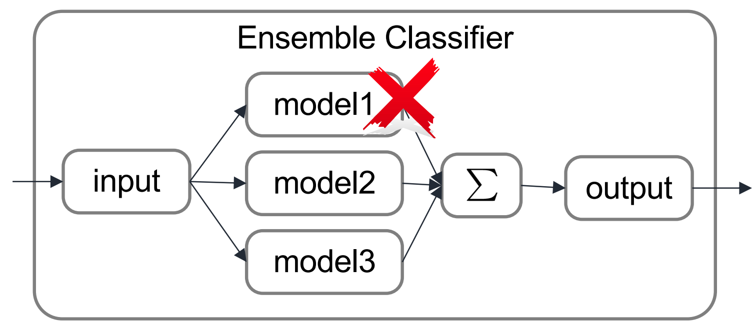
– **Problem:** Models may result in incorrect or **inaccurate predictions** (denoted model or data “drift”)

– **Solution:** Monitor the runtime traffic and raise an alert if drift is detected

```
subscribers:
  - name: drift-detector
    _type: modelops.drift.DriftDetector
    model: model1
    window_size: 500
    p_value_threshold: 0.05
    actions:
      - notify_slack
      - update_ensemble_classifier
```

Drift detector configuration



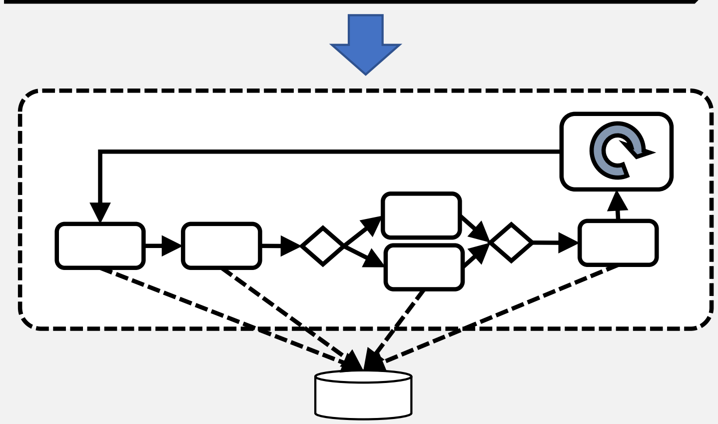


Future Work: Dynamic pipeline generation and large-scale optimization

User View

Goal Specification

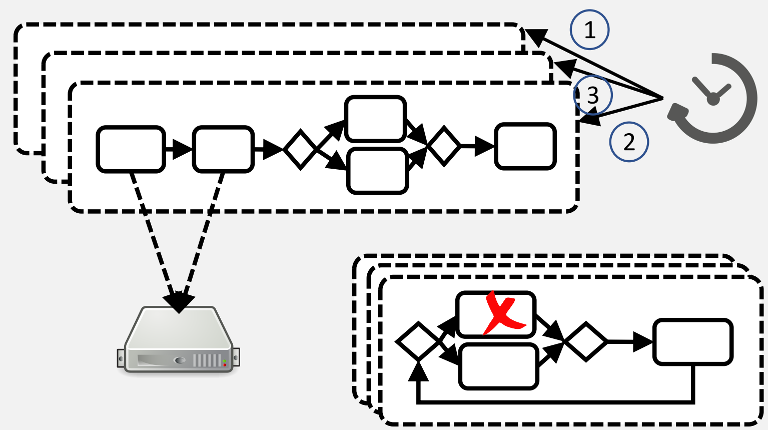
Example: *"We need to deploy our model with **bias** checks and **lineage** enabled. If the runtime data starts **drifting**, we need to **retrain** and deploy a new version."*



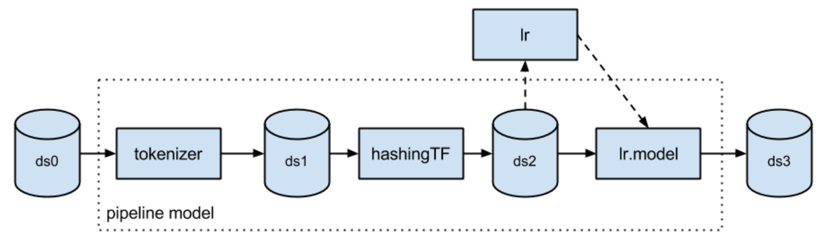
- Two Flavors:
  - Dynamic generation, static execution
  - Ad-hoc generation, dynamic execution

Provider View

- Analyze execution traces to learn usage patterns
- Perform platform-level optimizations
  - Task scheduling
  - Task pruning
  - Task colocation



Related Work



Pipelines in different Machine Learning environments

- ML Pipelines in Spark using MLlib [Meng'16]

- ML Pipelines in scikit-learn [Pedregosa'11]
  - Transformers and estimators

Automated Machine Learning

- Efficient and robust automated machine learning [Feurer'15]
  - AutoML as a Combined Algorithm Selection and Hyperparameteroptimization (CASH) problem

Production-grade machine learning platforms

- TFX: A TensorFlow-Based Production-Scale Machine Learning Platform [Baylor'17]
- Michelangelo: Uber's Machine Learning Platform [Hermann'17]

ModelOps – Future Directions

- **Domain Abstractions**
  - Cross-platform model train&deploy (hybrid/multi-cloud)
  - Model versions, staged deployments, A/B testing (e.g., large-scale transfer learning pipelines)
  - Multi-level pipelines in edge scenarios

- **Usability / UX**
  - User study on UX and configuration formats
  - Configuration CLI – (“`modelops init`”)

- **Large-Scale Pipeline Scheduling & Optimization**
  - Looking at ModelOps pipelines from **provider’s point of view**
  - Ad-hoc pipeline creation and scheduling under resource constraints
  - Simulation, experimentation, and analytics environment to evaluate different strategies

- **Extended Pipelines for Online and Reinforcement Learning**
  - Explore the computational model for stateful checkers, bandits, RL agents

ModelOps

A programming model for reusable, platform-independent, and composable AI workflows

Waldemar Hummer, Vinod Muthusamy

IBM Research AI

IBM Programming Languages Day
T.J. Watson Research Center
December 10, 2018
https://ibm.biz/plday2018

Contact: whummer@ibm.com

Team:
Yorktown: Waldemar Hummer, Anupama Murthi, Kaoutar El Maghraoui, Benjamin Herta, Darrell Reimer, Punleuk Oum, Gaodan Fang
Austin: Vinod Muthusamy
Cambridge: Scott Boag

Discussion

References

- Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Xin, D. (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, *17*(1), 1235-1241.

- Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems* (pp. 2962-2970).

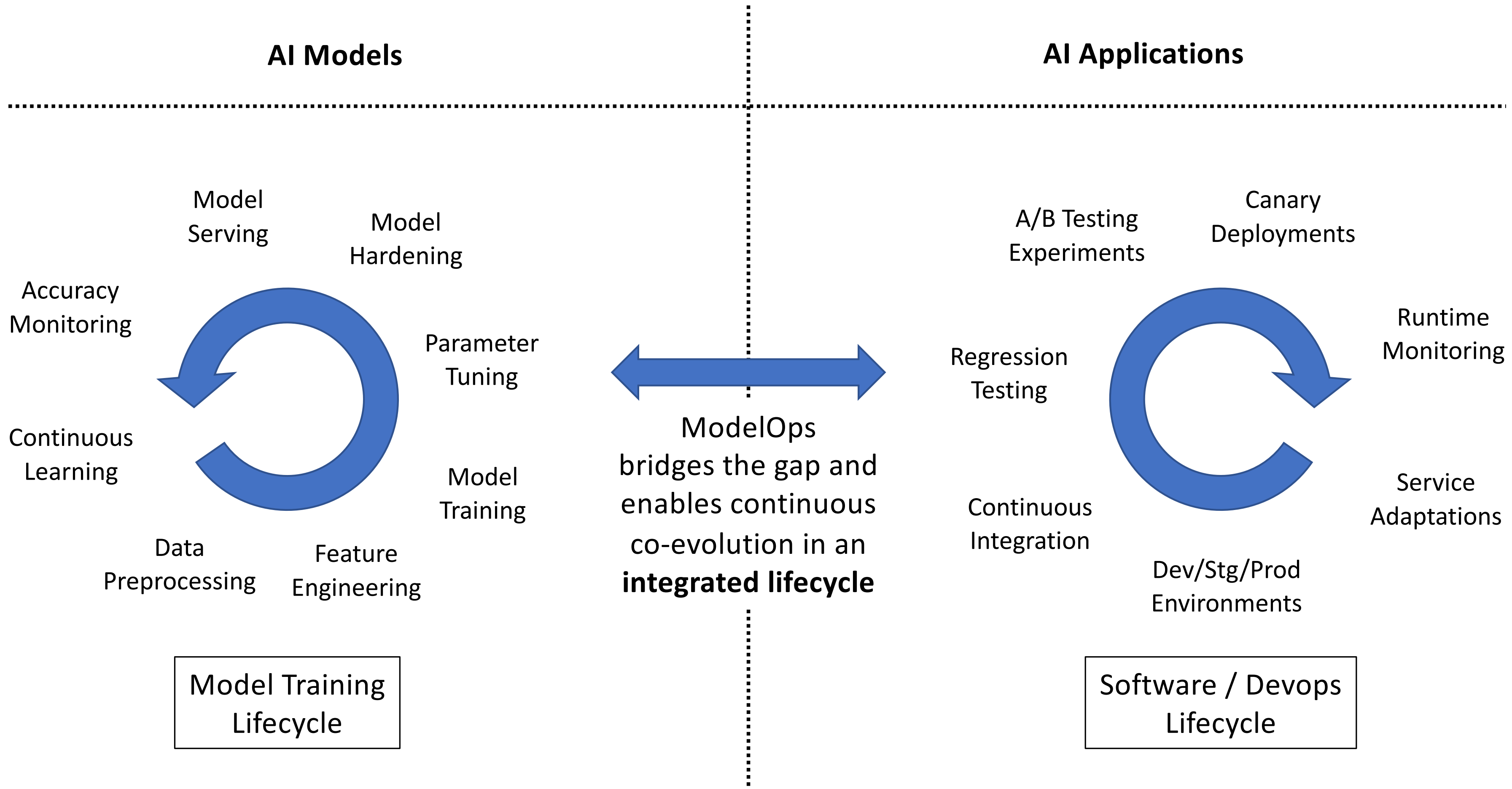
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, *12*(Oct), 2825-2830

- Baylor, D., Breck, E., Cheng, H. T., Fiedel, N., Foo, C. Y., Haque, Z., ... & Koo, C. Y. (2017, August). Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1387-1395). ACM.

- Hermann, J., & Del Balso, M. (2017). Meet Michelangelo: Uber's machine learning platform. *URL* *https://eng.uber.com/michelangelo*

Backup

ModelOps – Mission Statement

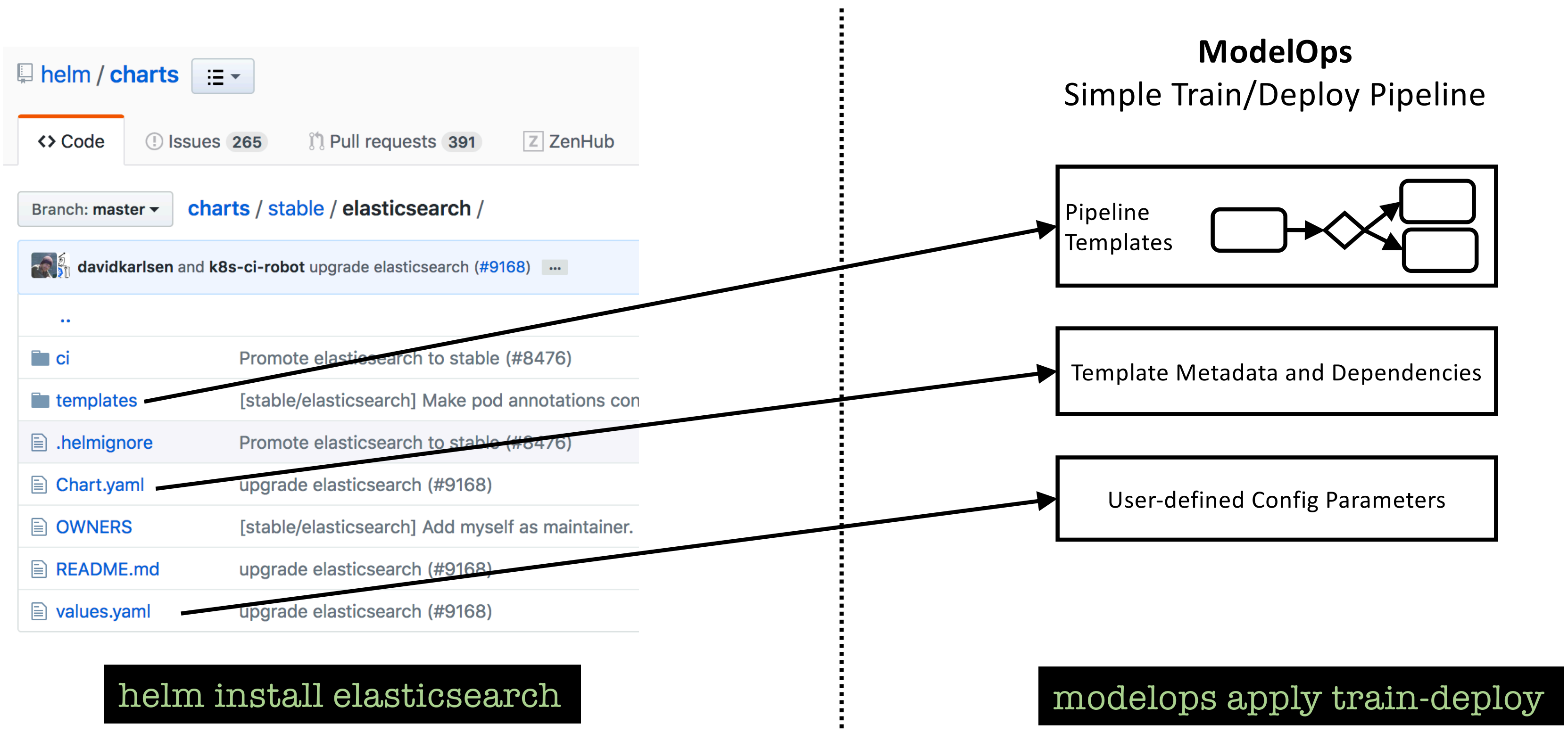


Generic AI Pipeline Representation - Python

```python
from modelops.plugins.slack import PostMessage as SlackNotification

# add single datastore to list
datastores + DataStore('account_wml', type='ibm/wml', ...)

# define pipelines
pipeline = Pipeline('dsl-demo-flow')

# define tasks
hardening_task = Harden('harden_model', ...)
notification_task = SlackNotification('notify_training', ...)
deploy_task = Deploy(...)
train_task = Train(...)

# add task
pipeline + train_task

# define a pipeline branch/fork
pipeline | (hardening_task, notification_task)

# define a pipeline join
pipeline & ('harden_model', 'notify_training', deploy_task)

...
```

Pipeline plugins from the community

- Lifecycle capabilities can be **crowdsourced**, based on the pluggable framework
- Analogy to Helm Charts



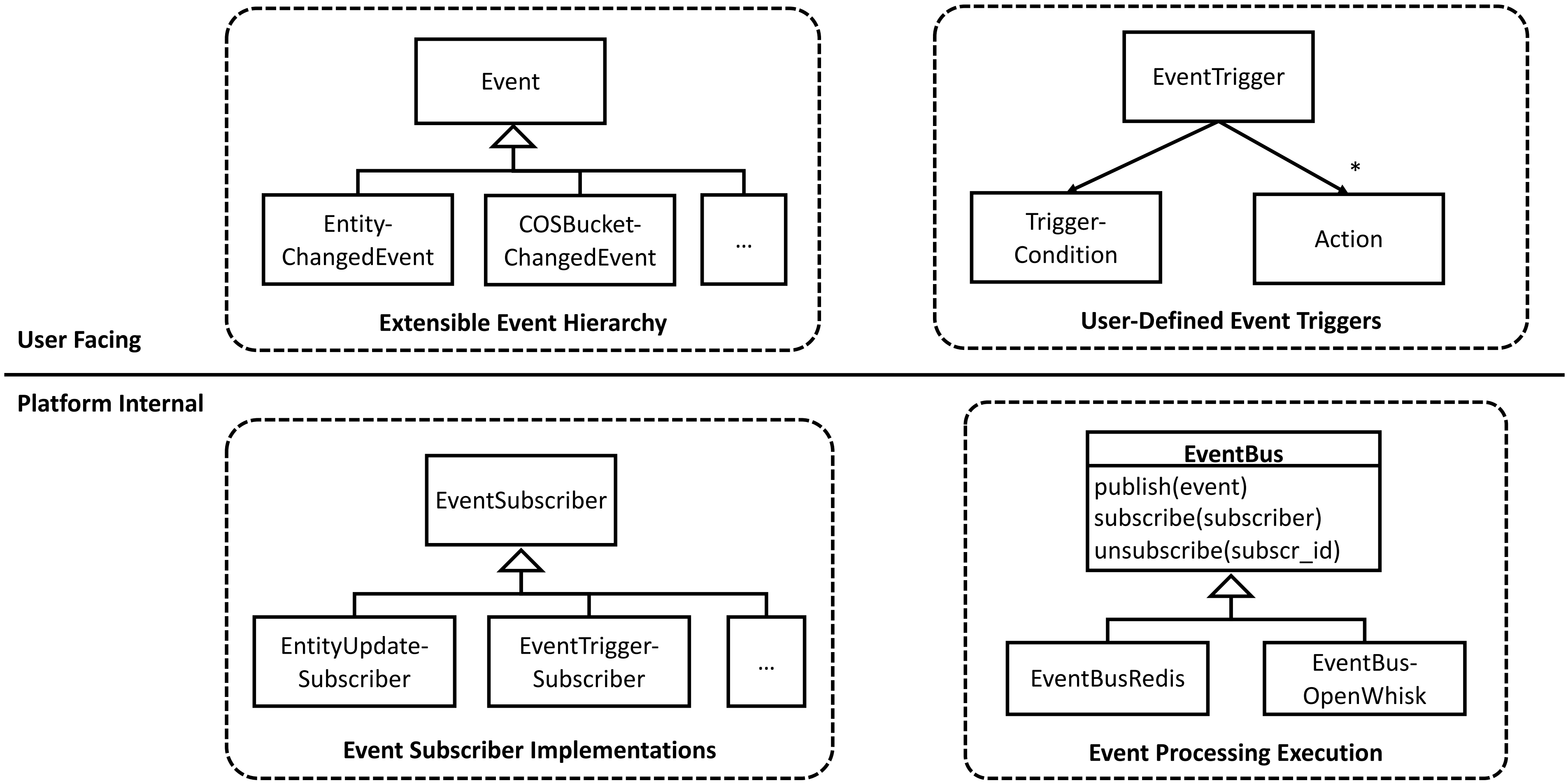
Environment Switches

```yaml
name: test_manifest
environments:
  - name: dev
    imports:
      - type: account
        name: wml_account_dev
        localname: wml_account
  - name: stg
    imports:
      - type: account
        name: wml_account_stg
        localname: wml_account
accounts:
  - name: wml_account_dev
    username: user_dev
  - name: wml_account_stg
    username: user_stg
```

- Import entities into an environment namespace
  - Create a local name alias
    - wml_account_dev → wml_account
    - wml_account_stg → wml_account
  - Local name “wml_account” can then be used in other parts of the configuration

- Activate different environments:
  - **ENV=dev** modelops run
  - **ENV=stg** modelops run

ModelOps – Lifecycle Event Processing



ModelOps – Event Trigger Example

```yaml
triggers:
  - name: wml-e2e-demo-trigger
    condition:
      _type: modelops.eventing.triggers.BucketChangedCondition
      datastore: wml-e2e-demo-store
      file_pattern: train.csv
    actions:
      - type: execute-pipeline
        pipeline: wml-e2e-demo-train
```

https://github.ibm.com/ModelOps/modelops-demo-titanic/blob/master/modelops.yml