

Breaking the Wall between AI and DevOps with MLOps

Introduction

What is DevOps?

DevOps is the standard way to manage application lifecycles through a pipeline of version control, test, build, and deployment tools (CI/CD cycle).

In the best case, it means a fully automated cycle: from a software engineer submitting the code into central version control (most commonly a Git repository) to building, testing and finally to deployment in the production environment.

Can we leverage DevOps processes for developing & deploying models?

Machine learning (ML) models are developed & trained s process to provide predictive capabilities for many scenarios. Examples include recommender systems, image classifiers, facial recognition scanners, fraud detection pipelines for advertising clicks, and speech-to-text translation services. We describe these as "AI-infused" scenarios.

A CI/CD solution for models requires supporting numerous types of data sources, a variety of training tools, a validation solution to analyze and validate models (for functionality and performance) and supporting deployment to the infrastructure used to serve models in production. This becomes particularly challenging when data changes over time and fresh models need to be produced regularly, as is the case in many large-scale, AI infused systems. Complexity only grows as models need to deployed to a hybrid of the Intelligent Edge + Intelligent Cloud.

Large-scale AI systems have three dimensions which define their complexity:

1. Volume of data
2. Number of models
3. Relationship between models used and performance requirements of scoring pipeline

Because the AI field is young compared to traditional software development, best practices and solutions around lifecycle management for these AI systems have yet to solidify. Orchestration is often done manually or ad-hoc, using glue code and custom scripts – this was also the case in traditional app development before DevOps best practices emerged. (1)

Our primary goals here are to:

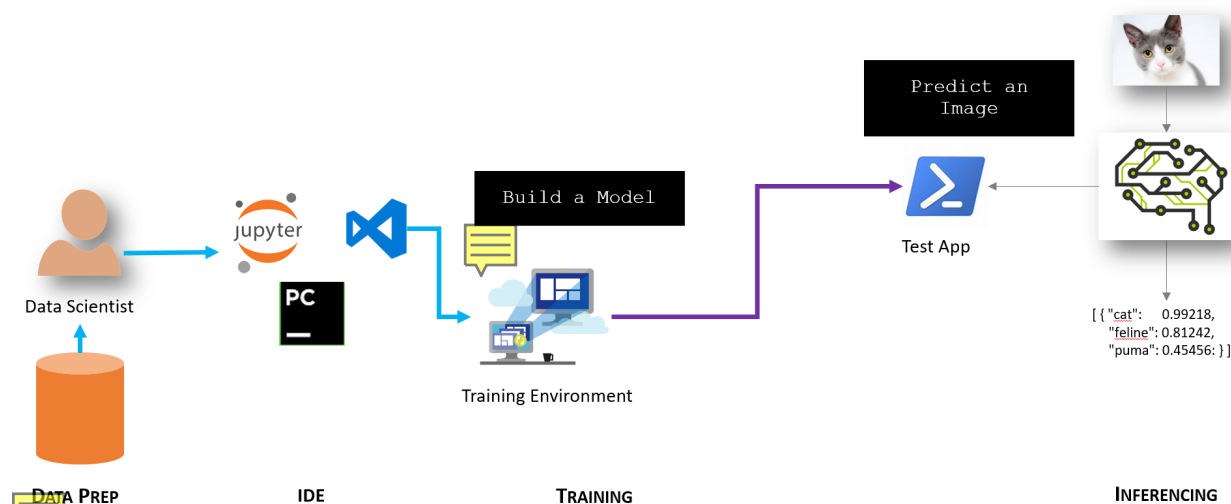
- standardize the components leveraged for model lifecycle management – model training, model validation, model storage / versioning, model and health monitoring
- provide lightweight process and templates to simplify the data scientist / app developer collaboration
- reduce the time from model creation to production deployment from the order of months to weeks to days

This paper focuses on the evolution of processes around the model CI/CD flow and key initiatives in this space.

Evolution of the Data Scientist / App Developer relationship

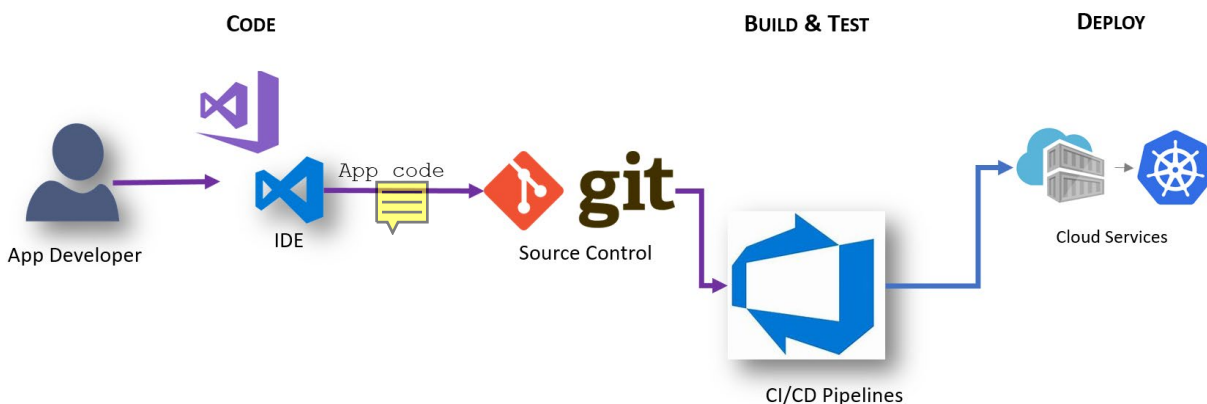
DevOps teams strive to automate all the flows, but data scientists tend to be in a silo outside the automation.

Data scientists spend a lot of time shaping data, figuring out which features are most relevant to their scenario, and figuring out which algorithm is best to solve their problem.



This creates an impediment to using models on the same cadence in the same process as the application.

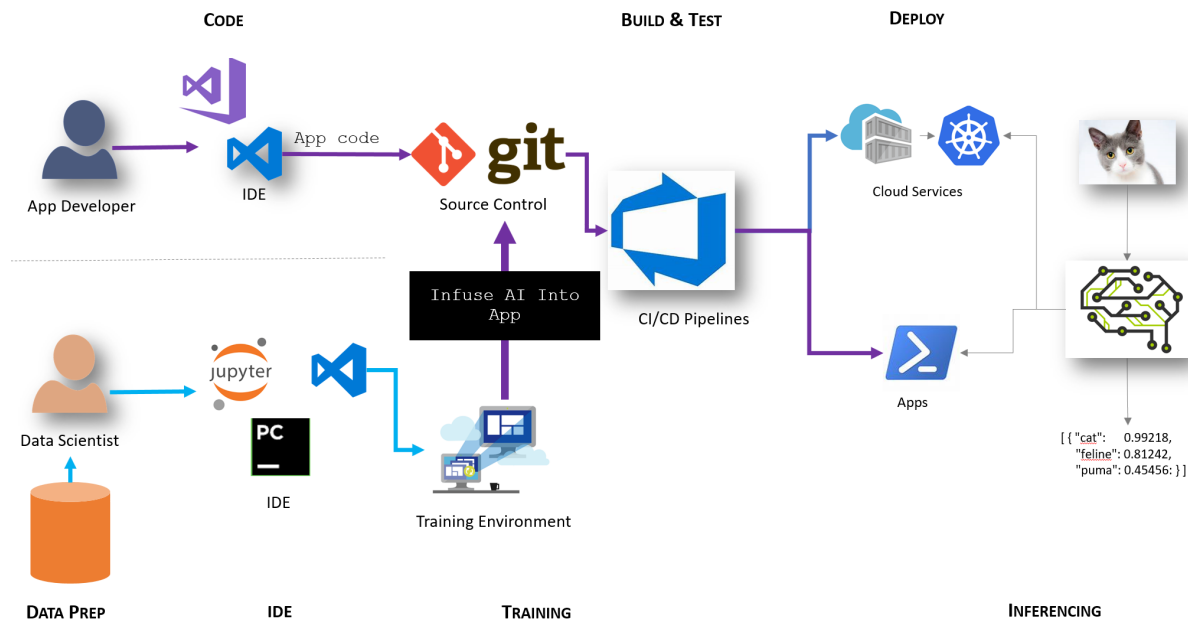
App developers have well established practices around getting their apps built, tested and shipped. The below flow captures this process:



We will now describe the evolution of the relationship between developer and data scientist once they've created their first viable model.

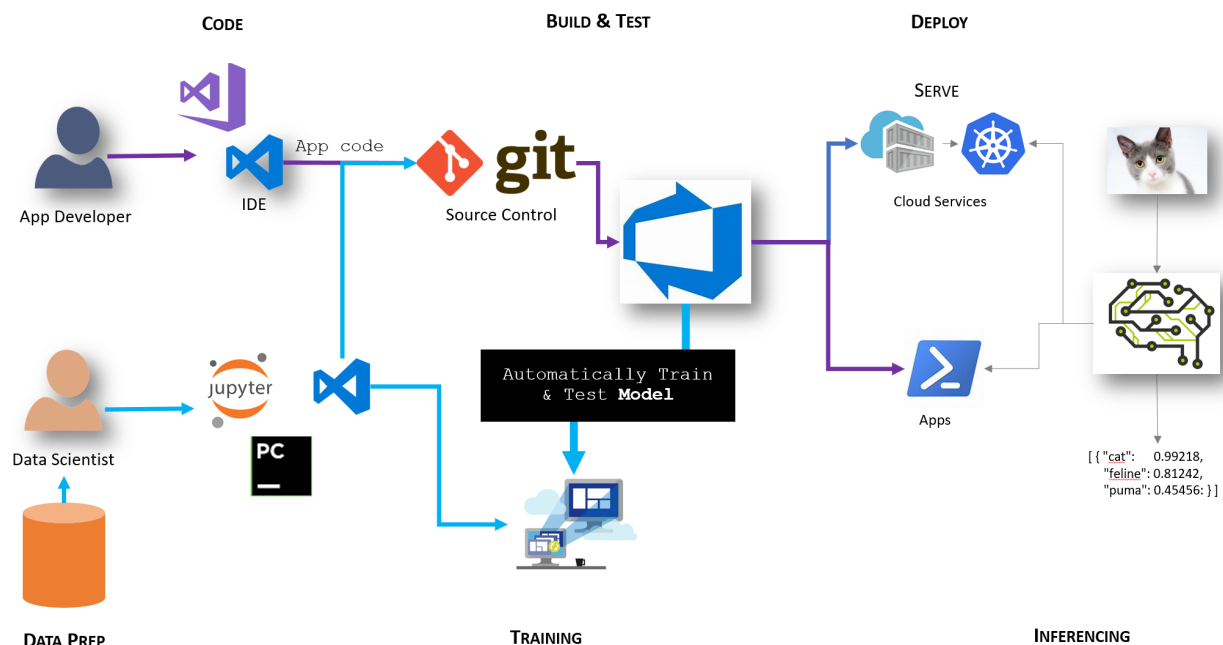


Phase 1 – Enable Model Reproducibility & Auditability



In the above diagram, data scientists arrive at a **complete model which can be handed off to app developers** for integration into their application. Typically this is done via manual embedding or sharing of basic interfaces.

The initial integration cost can be high here due to factors such as rewriting featurization code to work in the inferencing stack, mismatch between data available during training vs. inferencing, etc.

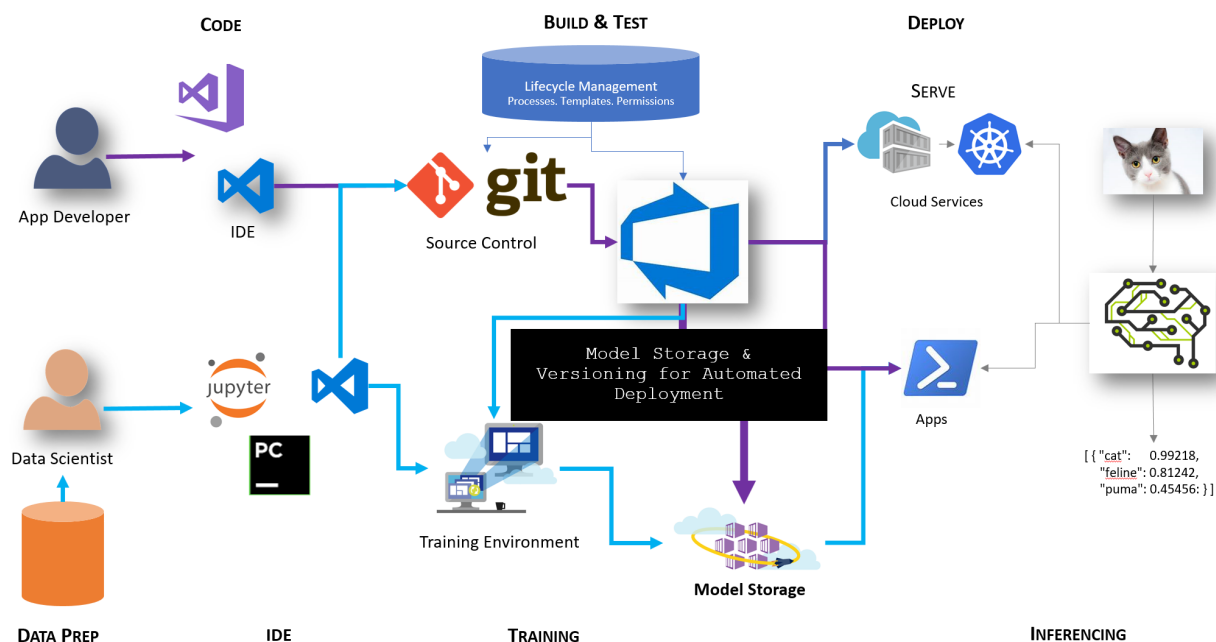


Once the model has matured & is being reliably used in production, a need arises to make it straightforward to **reproduce the model** used in the application.

There are frequently scenarios where the model needs to be retrained and redeployed (for example, when the model is dependent on time series data, or a specific set of input signals which may change for the device being analyzed over time). It is paramount at this point that **training code and test code goes into source control / a reproducible pipeline.**

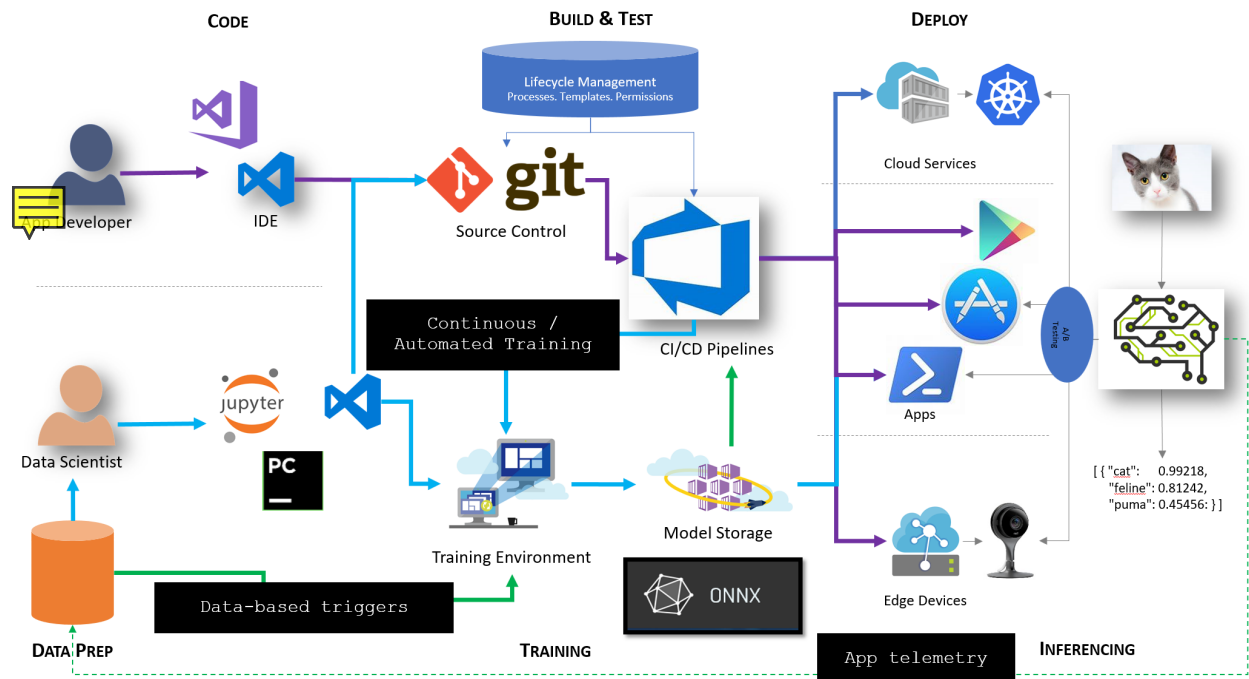
Phase 2 – Enable Model Validation

Introducing a reproducible pipeline also introduces the need to establish practices around **model validation**. This can be as simple as basic unit tests for the training code, an A/B comparison against a previous version of a model, or an end-to-end functional and performance testing suite for the model.



Now that one can to reproduce or iterate on a given model, model generation velocity increases & a need arises for a scalable solution to manage **model storage & versioning.**

Phase 3 – Enable Controlled Rollout & Automated Retraining



Once models are reproducible, validation pipelines are in place, and the story around model storage and versioning has been solidified, we now have a solution which can enable **continuous** delivery of new models to a variety of platforms & **continuous retraining of models** across scenarios.

Focus areas

The AI Lifecycle consists of these major buckets: **data preparation**, **model training**, and **model serving**. In this paper, we are going to specifically focus on the process which begins once prepared data streams are available to the data scientist (and how to properly stitch together an end-to-end solution to turn that input data into a production-quality model).

MLOps initiatives are focused on the following 4 areas:

1. **Model source control / reproducibility** - Processes and procedures to make **models reproducible** (from source control to data retention policies)
2. **Model validation** – unit testing, functional testing and performance testing - both in isolation and when embedded in an application
3. **Model versioning and storage** – provide a consistent way to store & share models, plus a way to track where models are embedded / running
4. **Model deployment** – describing an efficient process to get a model build into an application or service and leveraged to light up an end-user scenario.

1. Model source control / reproducibility


While many data scientists begin model development in a notebook or environment which doesn't necessitate the usage of source control, they find over time that it is useful from a reusability and collaboration point of view.

These are issues which arise when data scientists do not use source control:



- No reproducibility for the model or traceability for how the model was produced
- **Makes governance, compliance difficult upstream & downstream**
- **Teams can't collaborate across models**
- **Source code is structured differently for each model**
- **Can't clone and train (no up-to-date instructions)**
- **Hard for app developers to figure out how the model works / how to use it**

Automation & reproducibility

The use of source control is the first step in enabling the convergence of the app developer and data science processes. Beyond supporting gluing components together, the CI/CD solution also needs to be simple to set up, simple to automate, and should make models reproducible.

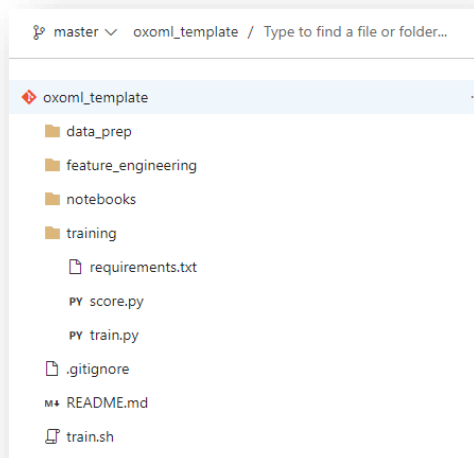
To reproduce a model, you need to be able to effectively keep track of the training code as well as the  input data used to generate the model. There is a delicate balance here, particularly when dealing with compliant data. While you want to keep data for long periods of time for reproducibility, you also want the ability for customers to keep control of their data and being able to delete it.

Templates for source control

Code and source control templates make it easy to get off the  ground with a model (by providing stub methods and factoring which allows for a consistent way to develop models across teams.) We propose that there are specific "DevOps AI" templates available for source control and CI/CD which can easily be embedded into any user's project. 

Example:





Templates, such as this one, establish a pattern for consistent development of models. They make it easy to reproduce a model and easy to work across team / product boundaries. They also simplify the lives of app developers who are working to leverage the model for inferencing.

2. Model validation

Standardized solutions and processes around model validation will help streamline the ability to train continuously and release models with a high degree of confidence in quality.

Model validation consists of **unit testing** of the model as well as **functional and performance testing** of the model embedded in an app / service – it aims to prevent bad models from reaching production. For example, you may train a model with a different format of input data than what is available to the inferencing service - this would generate a superior quality model during the training process, but worse performance in production.

In addition to meeting the production quality bar for functionality / performance, a model is considered safe to serve when it does not crash or cause errors in the serving system when being loaded, or when sent bad or unexpected inputs, and it doesn't use too many resources (such as CPU or RAM).

Testing the model

Traditional unit and integration testing runs on a small set of inputs and is expected to produce stable results. The test will either pass or fail.

In machine learning, part of the application has **statistical results** — some of the results will be as expected, some not. The most crucial difference in testing a model vs. a traditional application is that there are a range of acceptable values (an **expected target** and its **evolution**) versus a finite signoff criteria.

Testing of the model may also involve testing of the data used to produce the model (to ensure it aligns in terms of schema and features with what is available in the scoring scenario).

Testing the app and model together

Besides testing your model, you may want to ensure it behaves correctly in the context of your larger application. This can be done by pulling in an existing version of your application (or a stubbed-out variant of it) and attempting to execute the host application's tests against the model itself.

As they mature, most large organizations end up building a custom stack for this type of validation. At Microsoft, Bing, Cortana and Office 365 have learned to focus on validation because customer trust relationships are paramount throughout the entire model lifecycle. These processes around testing proper functionality and performance prevent issues in the live-site and maintain the quality of the brand.

This also helps to ensure that data schemas (input / output) and behaviors for all base cases in an application are sufficiently covered.



3. Model versioning and storage



There are several issues which arise as the model development lifecycle begins to mature. As more models are produced by a growing number of data scientists, the following pain points tend to pop up:



- Sharing and collaboration issues – need a consistent story for model storage
- No easy way to control who has access to models (or who can provide them) – need to manage permissions & traceability
- Too many model formats make reuse difficult – need a standardized model format

There are several strategies involved in versioning and validating models. As their size varies widely (especially comparing traditional ML classifiers with deep learning networks), they may be embedded directly into source control or stored on a versioned package feed.

Mature teams are concerned with validated, reusable learning. This requires tackling experiment history in some fashion – whether in a notebook, in source control, or in a centralized database. Each of these experiments provides information on the model produced and the relative accuracy of the model for a specific scenario. These experiments may be coming from a variety of source locations (from an SDK, from an IDE such as VS/VSCode, or from a CI job) – all are valid to track.



Sharing & collaboration

As organizations invest in Data Science and AI models, they find that work is being duplicated all over the organization. For most, there is no centralized system for logging all models, making them searchable, and allowing engineers to deploy them as microservices.



This means that many teams may be duplicating efforts, and other teams may have the need for a model but not the resources to develop it.

Permissions & traceability

The idea of making an organization's most valuable assets available to anyone in the company may send shivers down the spine of some in IT governance. Solutions around model storage need to incorporate access control as a first-class scenario.

Model format


One of the key tenets of machine learning scientists is to be able to experiment with a wide range of flexible tools. Enabling interoperability makes it possible to get great ideas into production faster.

An intermediate, cross-framework compatible language for storing models (such as **ONNX**) will make it easy to re-use models across a variety of inferencing stacks – this enables switching which frameworks are used in the training process without requiring massive code rewrites on the inferencing side.

4. Model deployment

One of the largest pain points which appears at any phase of the data science / app developer interaction is the model deployment process. The following pain points typically pop up:

- Hard to get model baked into an app or service
- No gates that prevent shipping a functional / performance regression
- Staging & release are a manual & time consuming process

By providing the tools necessary  to help simplify and manage model deployment, engineers and scientists have more time to focus on the modeling tasks.

The goal of these best practices is to **make models easy to deploy, with quality and security, across a wide variety of inferencing targets.**


What are the targets for a model?

A model could be:

- deployed as a standalone containerized inferencing service
- used as part of a batch processing pipeline
- embedded into an existing application or service

The **unit of deployment** varies depending on which customer you are targeting for a given scenario. The goal is to make the creation and release of the deployment unit simple, fast and efficient as possible.

Easy to deploy

A good CI/CD solution for models needs to expose simple  user interfaces to make it easy for engineers to deploy and monitor models with minimal configuration. Furthermore, it also needs to help users with various levels of machine-learning expertise understand and analyze their data and models.

Some things which make model deployment easier:

- Providing a friendly format for models (such as PNNL or ONNX)
- Simplifying the process to interact with the model (through code-generation, API specifications or other methods)
- Supporting a variety of inferencing targets out of the box (cloud / app / edge) (including specialized hardware such as FPGAs or dedicated frameworks such as CoreML and WinML)
- Provide secrets / service endpoint management to remove friction from configuring the release process.

Easy to gate

Governing the release process of your models can be a daunting process—you'll need to ensure access controls, manual and automated checks, and provide auditability of the deployment process. This is particularly true when dealing with compliant release environments & customer data.

Easy to automate

Most training pipelines are set up as workflows or dependency graphs that execute specific operations or jobs in a defined sequence. If a team needs to train over new data, the same workflow or graph is executed again. As many real-world use-cases require continuous training. CI/CD solutions will need to support triggering based on the availability of new data (where the training code itself remains static) as well as other signals (for example, data drift).

The CI/CD platform should support training a single model over fixed data, but also the case of generating and serving up-to-date models through continuous training over evolving data (e.g., a moving window over the latest n days of a log stream).

What's Next?

MLOps is a complex and continuously evolving ecosystem. With Azure DevOps + Azure Machine Learning, we have the tooling needed to address pain points around source control, model validation, model versioning / storage / sharing, and model deployment. In future we will provide templates, tasks, triggers and processes directly into the product to reduce friction between app developers and data scientists and improve overall agility in AI-infused engineering.