Microsoft logo white text version

Zzz

Une image contenant bleu, brouillard, capture d’écran, bleu vert

Description générée automatiquement

Responsible AI Workshop

Implementing a Responsible AI Lifecycle for MLOps processes

An illustration guide with Azure Machine Learning for data engineers, data scientists, AI developers, and other AI practitioners to help putting Responsible AI into practice

Version 1.1 - August 2022 (Updated: June 2024)

This page is intentionally left blank.

Table of contents

[Notice 6](#_Toc170393888)

[About this guide and the learning objectives 7](#_Toc170393889)

[Objectives of this guide 8](#_Toc170393890)

[Non-objectives of this guide 9](#_Toc170393891)

[Guide elements 10](#_Toc170393892)

[Guide prerequisites 10](#_Toc170393893)

[Cloning the project repository 11](#_Toc170393894)

[Using the Jupyter notebook in Azure Data Studio 12](#_Toc170393895)

[Introduction 15](#_Toc170393896)

[Enterprise-grade Machine Learning paradigm for cloud-native applications 17](#_Toc170393897)

[Overview of the Machine Learning Lifecycle 19](#_Toc170393898)

[Overview of the Responsible AI Lifecycle and integration with the ML workflow 23](#_Toc170393899)

[Introducing Azure ML MLOps capabilities 24](#_Toc170393900)

[Phase 1: Defining model requirements with RAIL “assess and prepare” stage recommendations 26](#_Toc170393901)

[Use case and data description 26](#_Toc170393902)

[Impact assessment - System uses, stakeholders and harms modelling 27](#_Toc170393903)

[Phase 2: Implementing the inner development loop with RAIL “design, build*,* and document” stage recommendations 31](#_Toc170393904)

[Initial setup 31](#_Toc170393905)

[Data pipeline 32](#_Toc170393906)

[Modeling pipeline 35](#_Toc170393907)

[Phase 3: Implementing the outer deployment loop with RAIL “validate and support” stage recommendations 54](#_Toc170393908)

[Documentation 54](#_Toc170393909)

[Gating - Closing the outer feedback loop with Azure DevOps Pipelines 55](#_Toc170393910)

[Using Confidential Computing for ML 65](#_Toc170393911)

[As a conclusion 69](#_Toc170393912)

[To go beyond 70](#_Toc170393913)

[Appendix – Azure DevOps pipelines step-by-step instructions 71](#_Toc170393914)

[*Build push* pipeline – Detailed instructions 71](#_Toc170393915)

[*Release* pipeline – Detailed instructions 78](#_Toc170393916)

[Bonus: *Model Training* pipeline – Detailed instructions 81](#_Toc170393917)

Notice

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

A drawing of a face

Description automatically generated

Attribution 4.0 International (CC BY 4.0)

Microsoft and any contributors grant you a license to this document under the [Creative Commons Attribution 4.0 International Public License](https://creativecommons.org/licenses/by/4.0/legalcode), see the [LICENSE](https://github.com/microsoft/responsible-ai-workshop/blob/main/LICENSE) file, and grant you a license to any code in the repository under the [MIT License](https://opensource.org/licenses/MIT), see the [LICENSE-CODE](https://github.com/microsoft/responsible-ai-workshop/blob/main/LICENSE-CODE) file.

Microsoft, Windows, Microsoft Azure and/or other Microsoft products and services referenced in the document may be either trademarks or registered trademarks of Microsoft in the United States and/or other countries. The license for this document does not grant you rights to use any Microsoft names, logos, or trademarks. Microsoft's general trademark guidelines can be found at <http://go.microsoft.com/fwlink/?LinkID=254653>.

Privacy information can be found at <https://privacy.microsoft.com/en-us/>

Microsoft and any contributors reserve all other rights, whether under their respective copyrights, patents, or trademarks, whether by implication, estoppel or otherwise.

# About this guide and the learning objectives

Welcome to this guide Responsible AI Workshop - Implementing a Responsible AI Lifecycle for MLOps processes for data engineers, data scientists, AI developers, and other AI practitioners.

As its name indicates, this guide is part of the Responsible AI Workshop and the related tutorials & walkthroughs.

In the last couple of years, artificial intelligence (AI) has been slowly but steadily infiltrating every aspect of cloud-native computing, which is “an approach in software development that utilizes cloud computing to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid cloud.”[[1]](#footnote-2)

It is indeed continuously being incorporated by developers into today’s strong trend of [cloud-native applications](https://azure.microsoft.com/en-us/overview/cloudnative/), these are modern applications that are built from the ground up to be optimized for cloud’s scale and performance. These applications are developed as loosely coupled microservices, running in containers, using managed services, and taking advantage of [Dev(Sec)Ops practices](https://azure.microsoft.com/en-us/solutions/devsecops/#overview) with end-to-end application lifecycle and integrated tools, automated continuous integration and continuous delivery (CI/CD) pipelines, a feedback loop with constant monitoring, etc. to deliver new features faster while maintaining uptime and high performance.

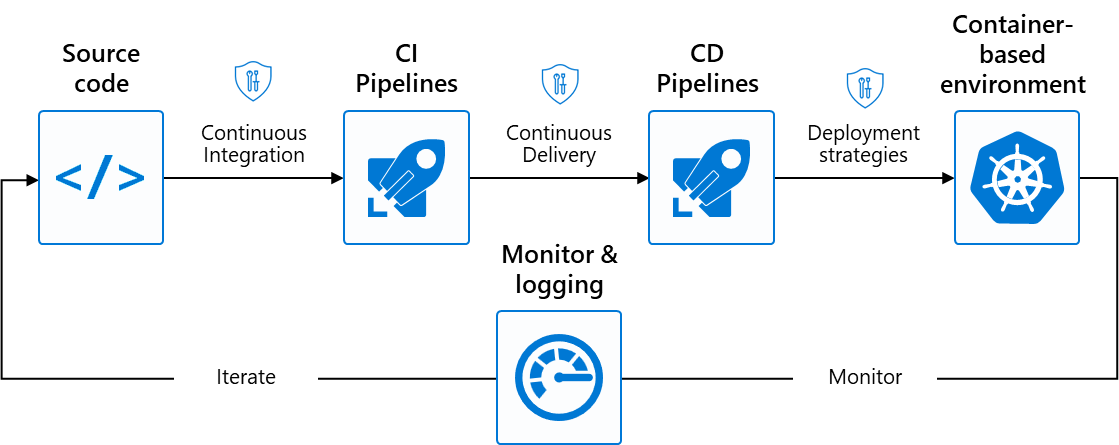


Figure . DevOps practices “in a box”

In this context, developers are using tools that allow them to leverage Machine Learning (ML) models within their microservices in order to ease/permit the implementation of some specific (intelligent, i.e., AI-powered) features, or even to expose Machine Learning (ML) models as data-driven microservices with API endpoints.

This has set the stage to the Enterprise-grade Machine Learning movement where [MLOps](https://azure.microsoft.com/en-us/overview/cloudnative/) (Machine Learning Operationalization), or [DevOps](https://azure.microsoft.com/en-us/overview/what-is-devops/) for Machine Learning (ML), enables data science and DevOps/IT teams to collaborate and increase the pace of model development and deployment via monitoring, validation, and governance of ML models at scale.

In parallel with this exponential expansion of the so-called intelligent cloud-native applications, the tech industry is being called upon to deploy these AI-powered systems more responsibly by addressing core ethical and accountability issues. To meet these challenges, organizations are striving to adopt Responsible AI (RAI) principles and translate them as a set of practices when designing, prototyping, building, deploying, and monitoring their AI systems, so that they benefit their stakeholders while also mitigating potential harms. This implies to drive a consistent end-to-end RAI approach through the entire lifecycle of these systems.

This process includes:

* Starting from a responsible assessment and preparation, i.e., the very first stage with the evaluation of the system’s benefits, the technology, the potential risks, and the team. Such activities enable to identify high priority areas within your AI development, building a way to later track and review the process, and securing approvals.
* Continuing with a responsible design, development (a.k.a. built), and appropriate documentation – from data collection and handling to ensure fairness in performance and representation. This allows to review the impacts, unique considerations, and the documentation practice.
* Ending on a responsible validation and support for the deployment, which starts with the selection of the testing procedures and the support to ensure systems work as intended. This also includes reinforcing practices and empowering people to use AI responsibly through documentation, gating, [confidential computing](https://azure.microsoft.com/en-us/solutions/confidential-compute/#overview), and more. Monitoring these systems once rolled out and gathering insights to detect any drift in their behaviors is also an integral part of this responsible deployment.

One should note that AI systems development often cycles through these different stages iteratively. Additionally, while many resources out there address the different stages of a Responsible AI lifecycle (RAIL) for an AI-powered system with a focus on activities to implement at each stage, very few resources focus specifically on the responsible development of intelligent cloud-native applications, which are (increasingly) becoming the norm today.

## Objectives of this guide

This guide aims to fill that gap by proposing an end-to-end walkthrough of a RAIL for intelligent, i.e., AI-powered, cloud-native applications based on MLOps and Dev(Sec)Ops processes and related common tooling, illustrating for that purpose, a complete end-to-end use case.

It illustrates how to put in place the above-mentioned practices and related automated and reproductible continuous integration (CI) and continuous delivery/deployment (CD) pipelines, while implementing specific activities that pertain to each stage of the above-mentioned RAIL through the lens of [Azure Machine Learning](https://azure.microsoft.com/en-us/services/machine-learning/#product-overview) (Azure ML) and its [MLOps capabilities](https://azure.microsoft.com/en-us/services/machine-learning/), [Azure DevOps](https://azure.microsoft.com/en-us/services/machine-learning/#product-overview), and other relevant Azure managed services in this context such as [Azure Container Instances (ACI)](https://azure.microsoft.com/en-us/services/container-instances/#overview), [Azure Kubernetes Service (AKS)](https://azure.microsoft.com/en-us/services/kubernetes-service/#overview), and [Azure Container Registry (ACR)](https://azure.microsoft.com/en-us/services/container-registry/#overview).

For that purpose, it notably highlights where to find and how to implement specific RAI capabilities and functionalities so you can leverage them in your own end-to-end development lifecycle for your AI-powered cloud-native applications.

Azure ML provides an end-to-end ML platform to enable you to build and deploy models faster on Azure, with experiment tracking, dataset management, and more, which is the reason we decided to showcase RAIL through a concrete use case in Azure ML here.

As such, this guide consists of a series of modules intended for data engineers, data scientists, ML developers and other AI practitioners, as well as DevOps/IT teams to cover the end-to-end development lifecycle aspects involved in the subject for intelligent modern applications, i.e., AI-powered cloud-native applications – for the sake of simplicity, we will refer them as AI systems for the rest of this guide.

It will more particularly explore these AI systems, and the related ML models’ design and management tools vs. Responsible AI (RAI) technology tools, where:

* AI systems design and management tools are resources that you can leverage to implement, build, and deploy your AI systems.
* RAI technology tools help you implement your AI systems with properties like fairness, privacy, security, and other RAI guarantees.

These RAI tools typically fall into three main categories:

1. Tools for a better understanding of the behavior of AI systems,
2. Tools for protecting AI systems’ models and data assets,
3. Tools for gaining control over these AI systems.

By the end of the guide, you will be able to:

* Have on overview of Enterprise-grade Machine Learning for AI-powered cloud-native applications, and the MLOps capabilities of Azure ML, possibly integrated with Azure DevOps.
* Understand specific activities to conduct at each stage of RAIL in this context.
* Implement an end-to-end AI system following RAIL stages with Azure ML, and Azure DevOps.

## Non-objectives of this guide

This guide is neitheraimed at introducing the building blocks of Responsible AI nor at giving a complete overview of RAI tooling for non-Generative AI. As stated above, the learning objectives of this guide are indeed to help moving towards a (more) reliable AI lifecycle in order to gradually strengthen the trust we can have in this technology and therefore facilitate its adoption in contexts where it would have a great responsibility.

For an introduction to RAI, and notably through Microsoft’s ongoing journey in the field, please refer to the guide [Establishing your own Responsible AI journey for your (non-Generative vs. Generative) AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/responsible-ai-journey/docs/establishing-your-own-responsible-ai-journey.docx).

For tutorials of the most prominent tools we open-sourced for non-Generative AI, please refer to the guide [Leveraging Responsible AI Tooling for your non-Generative AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/nongen-ai-tooling-tutorials/docs/leveraging-responsible-ai-tooling.docx).

Furthermore, the suggested RAIL as part of this guide does not integrate specific considerations that pertains to security, privacy, and safety. Please refer to please refer to the guide [Framing a (more) Trustworthy AI Lifecycle for your AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/trustworthy-ai-lifecycle/docs/framing-trustworthy-ai-lifecycle.docx).

These three guides are also part of this Responsible AI Workshop, which is available on GitHub at <https://github.com/microsoft/responsible-ai-workshop>.

**Note** For a complete overview of Microsoft’s resources designed to help you responsibly implement AI systems, please refer to the [Responsible AI resources page](https://aka.ms/rairesources).

## Guide elements

The document is organized as follows.

The Introduction presents Enterprise-grade Machine Learning for AI-powered cloud-native applications (later simply referred to as AI systems for simplicity) and gives an overview of the three stages of the MLOps practices covering the model requirements phase, the so-called “inner loop” for design and development, and the related “outer loop” to distribute the AI system where applicable, with each of these phases mapped to a corresponding phase of the Responsible AI Lifecycle (RAIL). This introductory module also covers Azure ML and its MLOps capabilities.

Then the next three modules provide a step-by step guide for implementing an end-to-end RAIL through a concrete loan decision use case.

Each module corresponds to a specific phase of an end-to-end ML lifecycle and the related MLOps/Dev(Sec)Ops practices for the considered AI systems. MLOps/Dev(Sec)Ops influences the end-to-end lifecycle throughout its assess/plan, develop, deliver, and operate phases. Each phase relies on the previous ones, and the phases are not role-specific regarding the intended audience. (In a true DevOps culture, each role is involved in each phase of the lifecycle to some extent.)

Each module provides in terms of learning objectives a focus on incorporating the activities recommended by RAIL for each related phase in order:

* The first module Phase 1 – Defining model requirements with RAIL envision stage recommendations focuses on the first phase by putting RAIL’s recommendations on responsible assessment into practice when outlining model requirements while the last two modules,
* Then, the modules Phase 2 – Implementing the inner development loop with RAIL define, prototype, and build stages recommendations and Phase 3 – Implementing the outer deployment loop with RAIL launch and deploy stages recommendations revolve around the other phases and focus on the introduction of RAIL activities into the common inner development loop, and the outer deployment loop usually implemented for cloud-native applications, respectively.

## Guide prerequisites

To successfully leverage the code in the Jupyter notebooks in this guide, you will need to run these companion notebooks in Azure ML in a Machine Learning workspace (ML workspace). There are four prerequisites:

* 1. Having an Azure subscription. If you don't have an Azure subscription yet, you can create a free account [here](https://azure.microsoft.com/free/).
  2. Having an Azure Machine Learning workspace. You will find instructions on how to setup one using the Azure portal [here](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-manage-workspace?tabs=python).
  3. Cloning the [project repository](https://github.com/microsoft/responsible-ai-workshop).
  4. Uploading the [notebook](https://github.com/microsoft/responsible-ai-workshop/blob/main/responsible_ai_lifecycle_walkthrough.ipynb) responsible\_ai\_lifecycle\_walkthrough.ipynb from the cloned repo to [Azure Data Studio](https://azure.microsoft.com/en-us/services/developer-tools/data-studio/#overview), a modern open-source, cross-platform hybrid data analytics tool designed to simplify the data landscape. This notebook is located under the lifecycle-walkthrough folder.

Further instructions for completing prerequisites 3 and 4 are provided in the next sections.

## Cloning the project repository

Theillustration guide repository on GitHubnotably contains a Jupyter notebook as well as complementary instructions about the use case. You have various options available to clone this repo.

For the sake of this illustration guide, we assume here that you have a Windows 10 local environment with possibly WSL 2 installed with an Ubuntu distribution. (This also *de facto* covers both Linux and MacOS environment).

Let’s consider them in order.

### Cloning the repo using Git for Windows

To clone the repo on your Windows 10 local machine, perform the following steps:

1. Download [Git for Windows](https://github.com/git-for-windows/git/releases/download/v2.25.0.windows.1/Git-2.25.0-64-bit.exe) and run it if not already done.
2. In the pop-up window, click Install. Follow the instructions.
3. Open a PowerShell console, and run the following command:

PS C:\> md rai-workshop

PS C:\> cd C:\rai-workshop\

PS C:\> git clone https://github.com/microsoft/responsible-ai-workshop

### Cloning the repo using Git on Ubuntu

Likewise, to clone the repo on your Ubuntu environment (WSL2), from a Bash terminal console, run the following commands:

$ cd $home

$ mkdir rai-workshop

$ cd rai-workshop

$ git clone https://github.com/microsoft/responsible-ai-workshop

### Cloning the repo using GitHub Desktop

To clone the repo on your Windows 10 local machine, perform the following steps:

1. Download the [GitHub Desktop installer](https://central.github.com/deployments/desktop/desktop/latest/win32) and run it.
2. In the pop-up window, click Install. Follow the instructions.
3. Open a browser session and navigate on GitHub to the main page of the workshop’ repo at <https://github.com/microsoft/responsible-ai-workshop>.
4. Click Clone or download.
5. Click Open in Desktop to clone the repository. GitHub Desktop opens up and a dialog shows up.
6. Click **Choose...** and, using Windows Explorer, navigate to a local path where you want to clone that repo. Throughout this guide, we will use the rai-workshop folder as an illustration.
7. Click **Clone**.

See [Cloning a repository from GitHub to GitHub Desktop](https://docs.github.com/en/desktop/contributing-and-collaborating-using-github-desktop/adding-and-cloning-repositories/cloning-a-repository-from-github-to-github-desktop) for more details if needed. You can instead install Git for Windows.

## Using the Jupyter notebook in Azure Data Studio

In this illustration guide, you are advised to [use Jupyter notebooks on Azure Data Studio](https://docs.microsoft.com/en-us/sql/azure-data-studio/notebooks/notebooks-guidance?view=sql-server-ver15) to first to train a ML model and then package that model as a container image that can later be deployed as a local web service or another compute target like the aforementioned managed services Azure Container Instances (ACI) or Azure Kubernetes Service (AKS).

Azure Data Studio takes advantage of an Azure ML workspace, which is a foundational block used to experiment, train, and deploy machine learning models. Most of these activities are included in the companion notebook responsible\_ai\_lifecycle\_walkthrough.ipynb available at the root of nongen-ai-lifecycle-walkthrough folder in the repo you (should have) just cloned at this stage.

Step 1: Access the ML workspace you created under recent resources or by looking up Machine Learning in the search bar and selecting your workspace.

Graphical user interface, application, Word

Description automatically generated

Step 2:Click on Launch Studio.

Graphical user interface, text, application

Description automatically generated

Step 3:Select Notebooks from the side bar.

Graphical user interface, application, Word

Description automatically generated

Step 4: Click on the Add file button and then on Upload files.

Graphical user interface, application, Word

Description automatically generated

Step 5:Navigate to the cloned repo and select the responsible\_ai\_lifecycle\_walkthrough.ipynb notebook and hit Open. This will open the notebook in Azure Data Studio, and you will be able to execute its cells after connecting your compute instance created as part of your ML workspace setup.

Graphical user interface, text, application

Description automatically generated

At this stage, you’re all set! You can either continue with the execution of the cells of the notebook in Azure Data Studio or choose Edit in VS Code next to the compute field highlighted above to open the notebook in VS Code. You can download VS Code for your environment here: <https://code.visualstudio.com/>.

It’s high time to move to the introductory module.

If you are only interested in the end-to-end walkthrough of RAIL with a concrete use-case, please feel free to skip this module and jump to the next ones. But we highly encourage you to follow the guide order as it first introduces Enterprise-grade ML for cloud-native applications, and Azure ML MLOps capabilities, which will be used in these next modules along with Azure DevOps.

# Introduction

Intelligent modern applications have been subject to many ground-breaking changes in recent years especially with the “cloud-native” paradigm shift on one hand and calls for a more responsible development of such applications on the other.

Regarding the former, the Cloud Native Computing Foundation (CNCF) [defines](https://github.com/cncf/toc/blob/main/DEFINITION.md) “cloud-native” as follows: *“Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.”[[2]](#footnote-3)*

As such, customers implement cloud-native applications using common practices such as containers for application packaging - The [Cloud Native Computing Foundation 2020 survey](https://www.cncf.io/wp-content/uploads/2020/12/CNCF_Survey_Report_2020.pdf) found that 92% of respondents use containers in production applications, and 29% release daily -, [12 factor app](https://12factor.net/) design and centralized configuration, development environment vs. production environment parity, etc. to achieve reliability and faster time to market.

They indeed also need to accelerate delivery of improved functionality and new features to their customers -whoever they are, to stay ahead of the competition and keep up with the pace of innovation, etc. and thus leverage Dev(Sec)Ops practices to work with their containerized applications quickly and easily with minimal operations and maintenance overhead, leading to shorter release cycles without downtime, and helping their organization achieve its desired continuous delivery/deployment (CD) approach.

The “cloud-native” paradigm has been gaining a lot of ground in recent years, and [Azure is at the forefront](https://azure.microsoft.com/en-us/overview/cloudnative/) of the effort to push its adoption at a large scale - Microsoft [joined the CNCF](https://azure.microsoft.com/en-us/blog/announcing-cncf/) back in 2017 -.

This induces practices that usually aim to establish two “loops” for cloud-native applications lifecycle:

1. A so-called “inner loop” for the design, the development, and the testing of the cloud-native application.
2. Along with a related “outer loop” to rollout the cloud-native applications where necessary.

as illustrated in Figure 2 below.



Figure . Inner and outer loops for Dev(Sec)Ops environment

In such a Dev(Sec)Ops environment, in terms of capabilities depicted through the lenses of Azure as a cloud platform:

1. Software developers rapidly iterate, test, and debug different parts of a cloud-native application together in the similar environment as to one(s) targeted for the production, typically the same Kubernetes cluster.
2. Code is merged into a source code repository, e.g., a GitHub repo, after which automated builds and tests are run as part of the continuous integration. [Azure Pipelines](https://azure.microsoft.com/en-us/services/devops/pipelines/), an integrated feature of Azure DevOps can be leverage for that purpose to automate your builds and deployments (anywhere) so that you spend less time with the nuts and bolts and more time being creative.
3. A pipeline automatically executes pre-defined deployment strategy with each code change. This will be a release pipeline with Azure Pipelines.
4. Kubernetes clusters, e.g., Azure Kubernetes Service (AKS) managed serverless clusters, [Azure Red Hat OpenShift (ARO)](https://azure.microsoft.com/en-us/services/openshift/#overview) managed OpenShift clusters on demand, etc. are provisioned if needed for production using tools like [Helm](https://helm.sh/) charts that define the desired state of app resources and configurations.
5. Container image is built and pushed to an Image Registry, e.g., Azure Container Registry (ACR) to store, secure, scan, replicate, and manage container images and artifacts, and connect across environments, including AKS and ARO, and across Azure services like Azure ML.
6. Cluster operators define policies in [Azure Policy](https://docs.microsoft.com/en-us/azure/governance/policy/overview) to govern deployments to the Kubernetes clusters at scale.
7. Azure Policy audits requests from the pipeline at the Kubernetes control plane level. [GitOps](https://www.weave.works/technologies/gitops/), as “an operating model for Kubernetes and other cloud-native technologies, providing a set of best practices that unify *git* deployment, management and monitoring for containerized applications » can be leveraged here.
8. App telemetry, container health monitoring, and real-time log analytics are obtained using for example [Azure Monitor](https://azure.microsoft.com/en-us/services/monitor/#features).
9. Insights used to address issues are fed into next sprint plans.

The “cloud-native” paradigm is (slowly) becoming the norm for multiple domains of applications and the (increasing) use of Machine Learning (ML) models to power them is also no exception to the rule.

As previously outlined, ML models indeed ease/permit the implementation of some specific (intelligent, i.e., AI-powered) features, or are event exposed as data-driven microservices with API endpoints.

Thus, as far as the latter is concerned, implementing an AI-powered cloud-native application, referred as to an AI system for simplicity, which in turn integrates all the recommendations and guardrails of the Responsible AI Lifecycle (RAIL) at each stage of the in-place ML Lifecycle is the challenge we decided to tackle with in this illustration guide.

The motivation behind this is twofold:

1. First, we want to demonstrate that a seamless integration of a RAIL in a standard ML Lifecycle and the related workflow are possible. We do this by highlighting a one-to-one mapping between the stages of a standard ML workflow sustained with MLOps processes and those of the RAIL.
2. Second, we want to lead by example and show you through a concrete use case how you can integrate RAIL activities into your own ML Lifecyle and related stages using Azure ML and its MLOps capabilities, which in turn integrates with the Dev(Sec)Ops practices that fuel cloud-native applications for the sake of painting a complete picture.

To get there, we first further introduce the Enterprise-grade Machine Learning paradigm for such cloud-native applications and what it means from a MLOps vs. DevOps perspective, and what the implications are for these applications that ultimately leverage ML model(s) at their core to deliver and/or enhance specific features.

With that, and to move forward on that common ground, we then give an overview of the stages involved in a standard ML Lifecycle and related workflow and those involved in a RAIL as defined by Microsoft and we highlight the one-to-one correspondence between the two sets of stages.

Finally, the last section provides the link between the first two and the next modules as it highlights Azure ML MLOps capabilities, which we will use in the rest of this guide to implement an end-to-end RAIL through a walkthrough of a concrete use-case that covers the complete application lifecycle.

Let’s now jump right into “unveiling” what it means to do Enterprise ML for cloud-native applications.

## Enterprise-grade Machine Learning paradigm for cloud-native applications

Training and deploying a ML model into production previously required concerted (manual) work from an IT team perspective creating and managing custom computing resources for deployment.

This creates a gap between the outer loop for operationalizing ML models and putting ML models into production and the inner development loop for training and maintaining the ML models themselves. Enterprise-grade Machine Learning at scale helps you bridge that gap by using the pre-built (managed) services provided by cloud platforms like Azure to assemble your AI system’s infrastructure, and to enable quick deployment and high availability, low latency data processing, along with a consistent environment across test, control, and production stages.

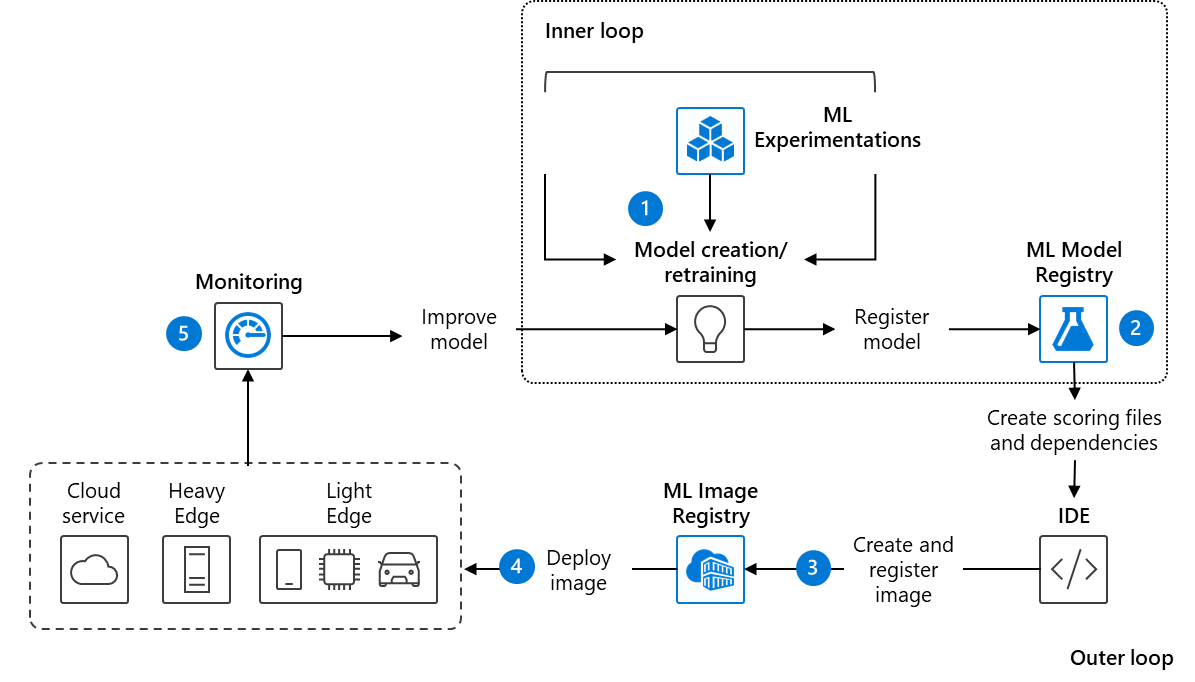


Figure . Data scientist “in a box”

Typically, in such an environment and using Azure capabilities:

1. A data scientist creates the ML model.
2. The model is registered using the [Model Registry](https://docs.microsoft.com/en-us/azure/machine-learning/concept-azure-machine-learning-architecture#register-model) of an [Azure ML Workspace](https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-azure-machine-learning-architecture#workspace).
3. The ML model is in turn registered using the image registry, e.g., and Azure Container Registry (ACR).
4. The image container is then deployed to cloud or to edge devices (on Kubernetes clusters).
5. The ML model is being used by/as part of the cloud-native application(s) and monitored - you can monitor input, output, and other relevant data from your ML model. Software developers can query the model for insights for the cloud-native applications.

Sounds familiar!? You will notice the similarities between this scenario and the one mentioned above for a general cloud-native application Dev(Sec)Ops environment.

However, using ML models as part of these applications brings additional considerations and needs “on the table”, for example in a non-exhaustive manner:

* The reproducibility of data to be used to train the ML model​.
* The validation of the ML model, namely: *does it meet quality bar?* *what about doing A/B comparison?* etc.​
* The storage and the versioning to track lineage and evolution of ML model over time.​
* The deployment of the ML model, its monitoring and the related data/metrics collection (across [intelligent cloud and intelligent edge](https://azure.microsoft.com/en-us/overview/future-of-cloud/)).

In this context, MLOps refers to the collection of practices that ease of collaboration between data engineers, data scientists, ML engineers, software developers, and other DevOps/IT teams to streamline and manage ML Lifecyle at scale.

MLOps processes, tools, and orchestration services are valuable throughout the stages and phases of the end-to-end ML Lifecycle. They:

* Help involved teams collaborate and provide visibility through shared, auditable documentation.
* Provide the ability to save and track changes to data sources and data sets, code, libraries, SDKs, and ML models.
* And create efficiencies and accelerate the lifecycle with automation, repeatable and reproductible workflows, and reusable data assets from specification of model requirements and triggering of the data and modeling pipelines to model deployment and monitoring.

With that, MLOps might sound very close like DevOps, which can be defined as “the union of people, processes, and products to enable continuous delivery of value to your end users.”[[3]](#footnote-4)

MLOps draws on DevOps principles and practices. Built upon notions of work efficiency, continuous integration, delivery, and deployment, DevOps responds to the needs of the agile business – in short, to be able to deliver innovation at scale. MLOps applies these principles to ML delivery, enabling the delivery of ML-based innovation at scale to result in:

* Faster time to market of ML-powered cloud-native applications.
* More rapid rate of experimentation, driving innovation.
* Quality Assurance (QA), trustworthiness, and in the context of this guide incorporating Responsible AI principles.

But one should note that the former also differs from the latter:

* Data/model versioning is different from code versioning, the main question that asks itself is: *how to version data sets as the schema and origin data change?*
* Digital audit trail requirements change when dealing with code + (potentially customer) data
* Model reuse is different from software reuse, as models must be tuned based on input data / scenario.
* Reusing a ML model may entail to fine-tune / transfer learn on it, meaning you need a training pipeline.
* Models performances tend to decay over time, and you need the ability to retrain them on demand to ensure they remain useful in a production context.

The above was a quick introduction to Enterprise-grade ML at scale for cloud-native applications and some of the considerations that pertain to MLOps vs. Dev(Sec)Ops to set the context. We won’t dive much into the details here as we will be building an AI-powered cloud-native application together in the next module.

But before continuing, we need to better understand the overall Machine Learning Lifecycle from outlining model requirements to deployment and monitoring which we will cover in the next section.

## Overview of the Machine Learning Lifecycle

For the purposes of our illustration, we decided to use an adapted version of the nine stages ML workflow proposed by a team from Microsoft Research (MSR) in their paper [Software Engineering for Machine Learning: A Case Study](https://www.microsoft.com/en-us/research/publication/software-engineering-for-machine-learning-a-case-study/)*.* Here we slightly modify the proposed workflow by highlighting the inner development and outer deployment loops to make the process look like software development workflows you might be familiar with.

Figure 4 shows an overview of the stages of the proposed ML workflow. We will go back to these stages frequently throughout this illustration guide so please take your time to carefully understand them, we provide a description of each stage a bit further.

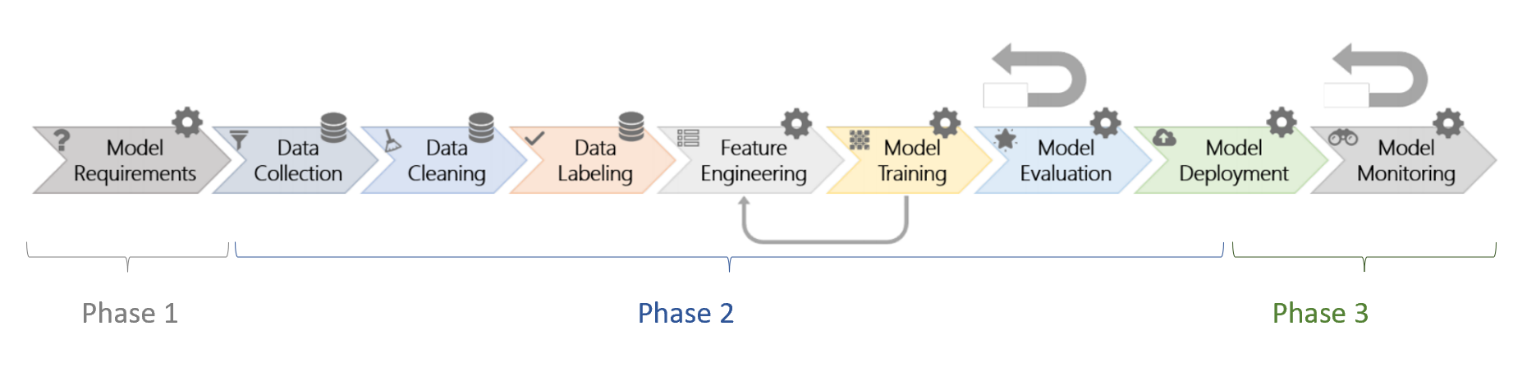


Figure . The nine stages of the ML workflow and the 3 proposed phases

Some stages in the workflow are data oriented (e.g., collection, cleaning, and labeling) marked with the little gear sign in Figure 4 above, and others are model-oriented (e.g., model requirements, feature engineering, training, evaluation, deployment, and monitoring) and are marked with the little database sign in figure one.

There are many feedback loops in the workflow. The larger feedback arrows denote that model evaluation and monitoring may loop back to any of the previous stages. The smaller feedback arrow illustrates that model training may loop back to feature engineering (e.g., in representation learning).

To highlight the importance of these feedback loops, we propose a new organization of these nine stages into three phases feeding into one another. We will subsequently refer to these as the three phases of the Machine Learning workflow as follows:

* Phase 1 solely includes the very first “Model requirements” stage of the workflow presented above.
* Phase 2 consists of the inner development loop and spans the data pipeline starting with data collection as well as the modeling stages up to model evaluation which can feed back into each of the previous stages. Please notes that phase 2 also comprises a local deployment part for testing the model before the real deployment, which we consider as part of model evaluation stage below - the last step of the modeling pipeline.
* Finally, Phase 3 comprises the deployment and monitoring stages that make up the outer loop which feeds back into itself and into any of the stages of the inner loop.

### Phase 1 - Model requirements

It might look surprising that the first model requirements stage was isolated into a phase of its own. The reason for this is that this is a key first step in the Machine Learning workflow which is often disregarded by designers (data engineers, data scientists, ML engineers or anyone designing the ML workflow) but nevertheless vital for the responsible assessment of the AI system as we will see in the walkthrough of the Responsible AI Lifecycle (RAIL) presented in the second module.

In the *model requirements* stage, designers decide which features are feasible to implement with ML and which can be useful for a given existing product or for a new one. Most importantly, at this stage, they also decide what types of models are most appropriate for the given problem.

### Phase 2 - Inner development loop - Data and modeling pipelines

The second phase of the ML workflow is the core inner development loop, which takes as input the model requirements and produces a ML model satisfying those requirements. This is usually the phase that induces the most attention around it as it includes implementing the model itself and tuning it, but let’s keep in mind that the two other phases are just as important for the overall workflow.

This second phase comprises two pipelines, most of which take place in a local development environment, connected through a feedback loop:

1. The data pipeline, which is composed of three stages:
   1. The *data collection* stage during which teams look for and integrate available datasets (e.g., internal, or open source) or collect their own.
   2. The *data cleaning* stage, which involves removing inaccurate or noisy records from the dataset.
   3. The *data labeling* stage where we assign ground truth labels to each record.
2. The modeling pipeline, which is composed of three stages as well:
   1. The *feature engineering* stage, which includes all activities that are performed to extract and select informative features for ML models.
   2. The *model training* stage during which the chosen models (using the selected features) are trained and tuned on the clean, collected data and their respective labels.
   3. And finally, the *model evaluation* stage where the engineers evaluate the output model on tested or safeguard datasets using pre-defined metrics. As the last step of model evaluation, we perform a *local deployment* of our model to test it before the (real) deployment(s) as part of the outer loop.

A feedback loop exists whereby after model evaluation we can go back to any of the previous stages of this inner development loop to improve the model.

Figure 5 below depicts this inner development loop to which we will go back in the second phase of our use case walkthrough.

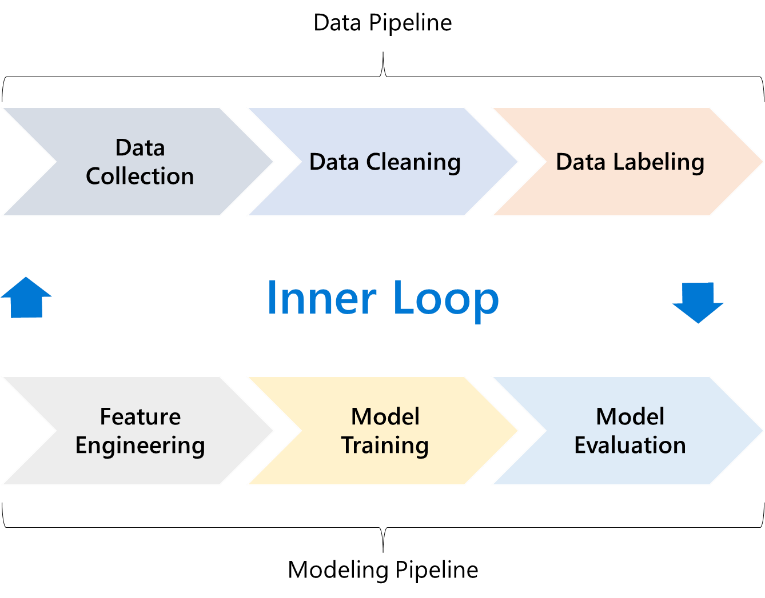


Figure . Overview of phase 2 of the proposed ML workflow - the inner development loop - and the two pipelines it comprises

### Phase 3 - Outer deployment loop – Model packaging, deployment, and monitoring

You might have built a ML model that exceeds your accuracy expectations and impresses your business sponsors. Now it’s time to deploy the considered ML model into production and it might not be as easy as you had expected. There are likely many things to put in place before your model can be used.

Over time, you or one of your colleagues might develop a new model that can do better than the old model. The question you must ask is, *can you put this new model into production without disrupting your business?*

It might be necessary for regulatory purposes to re-create the model and explain the model's predictions if unusual or biased predictions are made. Data inputted to your training and model can also change over time. Because of these changes, it might be necessary to retrain the model periodically to maintain the accuracy of its predictions.

This is exactly where the third and last phase of the ML workflow comes into play, it comprises two stages:

1. The *model deployment* stage, which consists of making the model available in production environments, where it can provide predictions to other software systems.
2. The *model monitoring* stage in which we oversee our ML model for things like errors, crashes, and latency, but most importantly, to ensure that your model is maintaining a predetermined desired level of performance.

As mentioned earlier, another big feedback loop exists from model monitoring to any of the previous stages of both the inner and outer loop as we might uncover problems or inaccuracies that need to be resolved by revisiting the requirements or altering the data, modeling, or deployment pipelines. Figure 6 below illustrates this feedback loop.

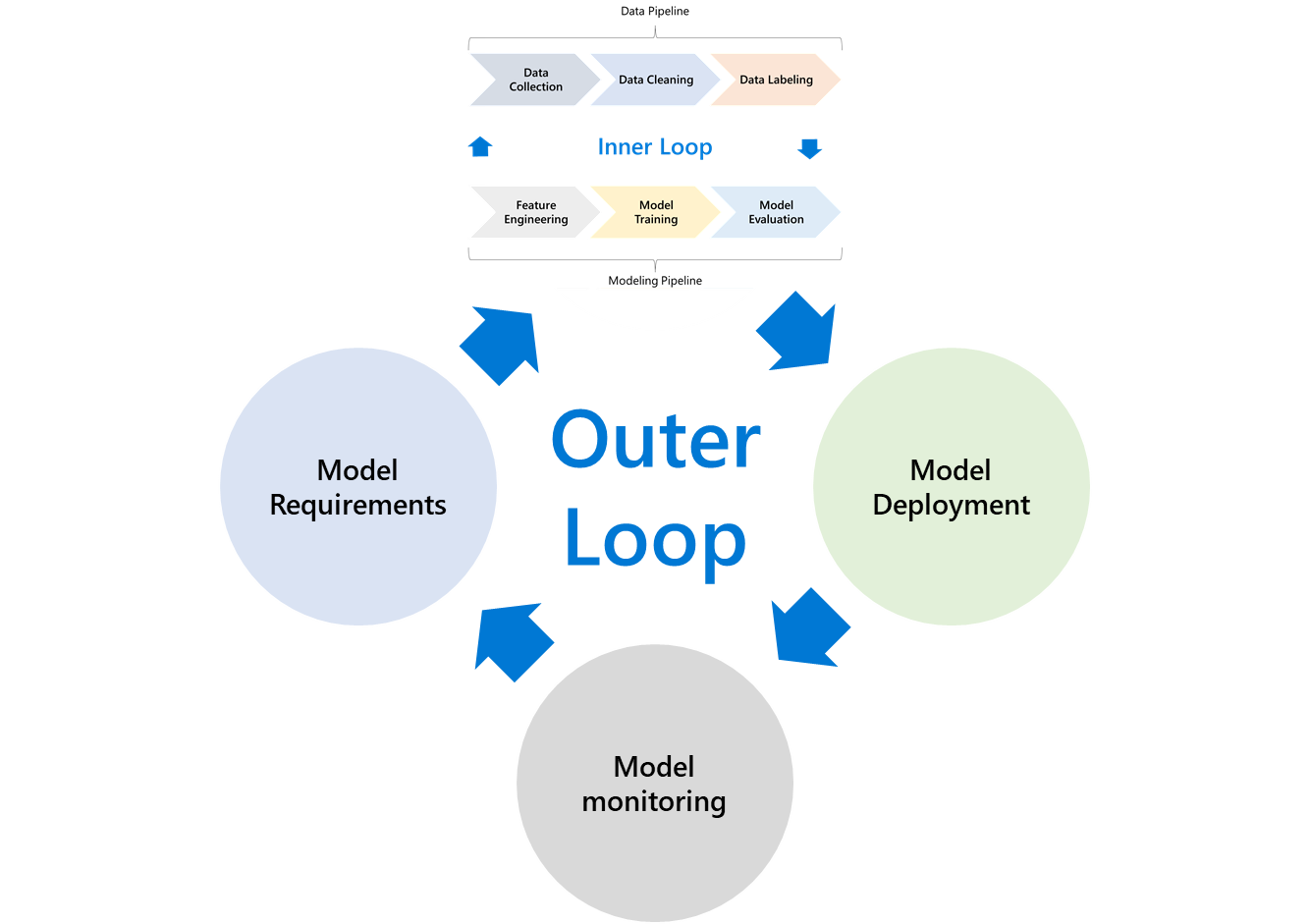


Figure . Overview of phase 3 of the ML workflow - the outer deployment loop - and how it feeds back into the two other phases.

Now that we have a good overview of the three phases of a standard ML workflow and the stages they contain, the question we (might) ask ourselves is what it means to conduct this ML workflow in a responsible manner. Or, in other words: *what are the activities to conduct at each step of this ML workflow to make it more responsible?* *Are they completely orthogonal to the workflow or can they be included into the workflow to accompany each stage in a clever manner?*

This is what we explore in the next section.

## Overview of the Responsible AI Lifecycle and integration with the ML workflow

The Responsible AI Lifecycle (RAIL) is a framework developed by Microsoft to help AI systems developers put evolving Responsible AI guidance and recommendations into practice. RAIL focuses on empowerment and teaching all members of product teams to think about people, impact, and responsible development at a high-level, while enabling deeper dives and linking to useful tools and resources per stage as needed and as they become available.

As introduced, RAIL defines 3 high-level conceptual stages of responsible AI system development, starting from (envisioning,) assessing and preparing a new AI technology, through designing, building, and documenting this technology to validating, deploying, supporting it, and continuing to evolve and improve the AI technology over time. It highlights emerging best practices, guidance, and resources to consider at every stage to ensure our AI products reflect [Microsoft’s AI principles](https://www.microsoft.com/en-us/ai/responsible-ai?activetab=pivot1%3aprimaryr6): Fairness; Reliability and Safety; Privacy and Security; Inclusiveness; Transparency; and Accountability.

Our ultimate goal for the use case we will present shortly is to conduct activities recommended by RAIL and use the tooling it suggests (which was investigated as part of the guide [Leveraging Responsible AI Tooling for your non-Generative AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/nongen-ai-tooling-tutorials/docs/leveraging-responsible-ai-tooling.docx) in this same Responsible AI Workshop) into each stage of the ML workflow. Looking at RAIL stages and at the ML workflow phases presented in the previous section, we see that there is a big similarity between the way these are organized.

In fact, by re-organizing recommendations from RAIL stages, we end up with 3 groups that have a one-to-one correspondence with the ML workflow phases.

We define these three groups as follows:

1. Group 1 includes the recommendations from the assess and prepare stage of RAIL and corresponds to the “Model Requirements” phase and the responsible assessment of the ML use case.
2. Group 2 includes recommendations from the design, build*,* and document stage of RAIL and corresponds to the implementation of the inner development loop of the ML workflow in a responsible manner.
3. Group 3 includes recommendations from the validate and support stage of RAIL and corresponds to the responsible implementation of the outer deployment loop of the ML workflow.

For the rest of this illustration guide, we will be following the three ML workflow phases and trying to implement the most prominent recommendations from the corresponding group of RAIL stages at each step of the workflow.

Now that we have a good overview of the different phases of the ML workflow, and mutualized these phases with RAIL stages, we are all very impatient to put these into practice with our use-case walkthrough. But before we get there, we are still missing one piece of information: knowledge of Azure ML MLOps capabilities. These will be extremely useful for our walkthrough which is the reason we investigate them right away.

## Introducing Azure ML MLOps capabilities

As introduced, MLOps enables data engineers, data scientists, ML engineers, and app developers to leverage DevOps principles and practices to help bring ML models into production and to ultimately to productionize them. The ML workflow presented in the previous section already includes the core stages of a sound MLOps lifecycle.

MLOps allows you to monitor, version, audit, certify, and re-use every asset in your ML lifecycle, as well as providing orchestration services to make managing your ML models easier, with the goal of:

* Faster experimentation and development of models.
* Faster deployment of models into production.
* End-to-end lineage tracking and quality assurance.

By breaking down the process between different release stages for example, you can evaluate how well the model performs on local/dev devices before releasing it to a Quality Assurance environment.

Azure ML contains several MLOps capabilities including asset management and orchestration services to help you manage the lifecycle of your model training & deployment workflows. These capabilities allow you to:

* Create reproducible ML pipelines. ML pipelines allow you to define repeatable and reusable steps for your data preparation, training, and scoring processes.
* Create reusable software environments for training and deploying models.
* Register, package, and deploy models from anywhere. You can also track associated metadata required to use the model.
* Capture the governance data for the end-to-end ML lifecycle. The logged lineage information can include who is publishing models, why changes were made, and when models were deployed or used in production.
* Notify and alert on events in the ML lifecycle. For example, experiment completion, model registration, model deployment, and data drift detection.
* Monitor ML applications for operational and ML-related issues. Compare model inputs between training and inference, explore model-specific metrics, and provide monitoring and alerts on your ML infrastructure.
* Automate the end-to-end ML lifecycle with Azure Machine Learning and [Azure Pipelines](https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops). Using pipelines allows you to frequently update models, test new models, and continuously roll out new ML models alongside your other applications and services.

A complete tour of Azure ML MLOps capabilities can be found under the [official documentation](https://docs.microsoft.com/en-us/azure/machine-learning/concept-model-management-and-deployment), as well as the GitHub repo MLOps on Azure at <https://github.com/Microsoft/MLOps>.

Figure 8 below shows the most important ML services in Azure, with MLOps capabilities outlined in the middle column.

A picture containing text, monitor, screenshot, screen

Description automatically generated

Figure 8. An overview of Machine Learning services in Azure.

This concludes this introductory module, and it is high time we jump straight into applying the ML workflow for the development of a Responsible AI Lifecycle in Azure ML.

# Phase 1: Defining model requirements with RAIL “assess and prepare” stage recommendations

The goal for the three next modules is to walk you through a concrete use case by following the three phases of the ML workflow presented in the introduction and applying the most prominent recommendations from RAIL at each stage. This will be done in a cloud-native manner by leveraging Azure ML MLOps capabilities also presented in the previous introduction.

Although it is possible that due to the nature of the chosen use-case, we might spend more time on some stages of the ML workflow than others, we will try our best to go through each stage - even the one’s less relevant to our specific use case - so that you can adapt this process to your own specific use cases in a straightforward manner.

We begin with a quick description of the use case before diving straight into the model requirements stage.

## Use case and data description

The chosen use case uses the well-known [UCI adult census dataset.](https://archive.ics.uci.edu/ml/datasets/Adult) For our purposes, we will treat this as a loan decision classification problem. For the sake of the illustration, we will pretend that the label indicates whether each individual repaid a loan in the past. We will use the data to train a predictor to predict whether previously unseen individuals will repay a loan or not. The assumption is that the model predictions will be used to decide whether an individual should be offered a loan.

This use case is inspired by the very good article titled [Responsible AI in action, locally and on Azure Machine Learning](https://medium.com/microsoftazure/responsible-ai-in-action-locally-and-on-azure-machine-learning-a515e0585e69) by our colleague Mauro Minella. Here we try to have a more 360 degrees approach to Responsible AI activities to be conducted at every stage of the ML workflow from model requirements to deployment, but please go check this article if you are looking for a more straightforward approach to the use case with a focus on the inner development loop.

For now, let’s start with the first stage of the ML workflow. Here we want to provide answers for two important questions:

1. *Is using ML to solve this problem necessary?* We try to answer this question using the impact assessment of our ML use case.
2. *Is there any potential harm?* *If yes, what is it?* *And are there scenarios where the AI system would fail?* We investigate these in the harms modeling section.

## Impact assessment - System uses, stakeholders and harms modelling

Ultimately, we need the requirements we provide here to reflect our business objectives but also to include responsible release criteria. To get there we go through a thorough impact assessment of our use case.

The impact assessment captures the team’s work to evaluate the impact of a system on people and society.

The purpose of impact assessment is to answer the three following questions:

1. Interrogate the system’s implications for people and society to understand and manage RAI challenges.
2. *Who are your key stakeholders and what are the potential benefits and harms for them?*
3. *What potential harms and tensions will you need to address?*

The goal here is to identify the system’s key intended uses and the socio-technical context in which it will be operated as well as the stakeholders involved in the process.

In our case, the intended use of our system is for a bank or financial institution, which wants to automate the loan eligibility process to analyse the data to know which customers (loan applicants) are risky or which are safe among all customers applying for a loan, or for a bank loan officer who wants to use our system’s predictions to inform his decisions.

The context of use is that of a bank or financial institution issuing loans to potential customers. Microfinance loans can [contribute to the socio-economic development](https://garph.co.uk/IJARMSS/Dec2017/3.pdf) of their beneficiaries but can also cause an [unsustainable debt burden](http://www.cadtm.org/Chapter-4-The-socio-economic) imposed by the creditors if issued to the wrong people. The social and economic consequences of loan attribution can range from saving small businesses and contributing to the economic development of people to increasing poverty, rising inequalities, and worsening life conditions beyond the economic loss that a loan default induces for the financial institution itself. Therefore, getting loan decisions right is extremely important and Machine Learning can play a major role in this perspective.

Figure 9 below shows the stakeholder definition based on their roles.

Diagram

Description automatically generated with medium confidence

Figure . Stakeholder analysis for our loan decision system

### Harms modeling

#### Value tensions

In this section, we will be framing harms in terms of value tensions. Value tensions happen when mitigating a harm directly impacts a benefit. These are often represented as polarities.

In our case, one important polarity that we will want to address is the accuracy vs fairness trade-off. Indeed, it might be the case that our model provides very good accuracy overall but does very poorly for a specific group of people defined by gender, race or other sensitive attributes compared to other cohorts.

Rectangle

Description automatically generated

Figure . Illustration of the accuracy - fairness value tension.

To keep this illustration guide as concise as possible and to give you the gist of the process conducted for this use case so that you can replicate it and avoid losing the reader in the details, we decided to only address a single harm (the accuracy - fairness value tension) for the remaining of the white paper although many other potential value tensions arise for a loan decision problem – see section Other types of value tensions.

#### Mitigating value tensions

Solutions for mitigating value tensions can be social, technical, and experimental.

In our case we propose the following solutions to the accuracy – fairness value tension outlined in Figure 11 below.

Graphical user interface, text, application, email

Description automatically generated

Figure . Solutions for mitigating the accuracy - fairness value tension for our loan decision use case.

#### Other types of value tensions

As mentioned earlier, we only focus on a single harm or value tension here. For completeness we provide a quick description of other types of value tensions that can arise so you can be on the lookout for them:

1. A benefit to one stakeholder may result in harm to another (or the same).
2. A mitigation to a harm supporting one AI principle may impact the ability to support another principle…
3. A mitigation to a harm or delivery of a benefit may impact the go-to-market strategy or viability of a key intended use.

### Other practices

Other practices can be leveraged in the responsible assessment phase like the [judgment call](https://docs.microsoft.com/en-us/azure/architecture/guide/responsible-innovation/judgmentcall) game or the [community jury.](https://docs.microsoft.com/en-us/azure/architecture/guide/responsible-innovation/community-jury/) Most of these activities require a multi-disciplinary team and are not illustrated here.

You can learn more about these by exploring the [Microsoft’s best practices toolkit](https://docs.microsoft.com/en-us/azure/architecture/guide/responsible-innovation/) for Responsible innovation.

This concludes the responsible assessment phase of RAIL. The impact assessment suggests the benefits of implementing this AI system outweigh its harms provided the value tension mitigation solutions presented above are considered. This means we can now move to the fun part – the development cycle in Azure.

# Phase 2: Implementing the inner development loop with RAIL “design, build*,* and document” stage recommendations

We now attack the inner development loop of the ML workflow. This is the biggest phase in terms of the number of stages involved, but also the most involved in terms of Responsible AI Lifecycle (RAIL).

It is composed of two stages connected through a feedback loop: The data pipeline preprocesses and feeds the data to the modeling pipeline, which upon model evaluation can inform better data processing or better modeling.

Let’s see how this applies to our use case.

## Initial setup

Before we get any further, we need to go through an initial setup step to make sure your Azure ML workspace is ready to be used.

import azureml.core

from azureml.core import Workspace

ws = Workspace.from\_config()

print(ws.name, ws.location, ws.resource\_group, sep='\t')

The few lines of code above create a workspace object ws from the existing workspace by reading the file config.json file and loading the details into the ws object. The compute instance of your Azure ML workspace has a copy of this config.json file saved in its root directory.

If you run the code elsewhere, you'll need to [create the file yourself](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-configure-environment#workspace).

## Data pipeline

The data pipeline is itself composed of three stages that together produce clean labeled data ready to be used for model training.

### Data collection or loading

The first step of the data pipeline is data collection or loading. In our case, the raw data is provided by UCI, so we won’t be collecting it ourselves. However, in what follows I will be proceeding as if we had collected the data ourselves to show you the standardized process for documenting your datasets.

For this purpose, we will be using a tool called [datasheets for datasets](https://www.microsoft.com/en-us/research/publication/datasheets-for-datasets/). The idea behind it is simple and is inspired from the electronics industry: every component in the electronics industry, no matter how simple or sophisticated it is, comes with a datasheet that details its operational characteristics, test findings, recommended applications, and other details. By comparison, the idea here is that each dataset is accompanied by a datasheet that details the dataset's motivation, composition, collection procedure, recommended uses, and so on.

Usually, we would follow the [datasheets for datasets template](https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE4t8QB) by completing every section of it (which is the recommended activity if you are collecting your own data).

A screenshot of the first section of this template is shown in Figure 13 below for reference. In our case, the UCI already provides a [datasheet for this dataset](https://archive.ics.uci.edu/ml/datasets/Adult) and here is a summary of the most important elements of this datasheet:

1. Motivation: Prediction task to determine whether a person makes over 50K a year to use it as an indicator of whether to attribute a loan to an applicant or not.
2. Composition: The dataset is composed of 48842 instances each with the following 14 attributes: Age, Workclass, Education-Num, Marital Status, Occupation, Relationship, Race, Sex, Capital Gain, Capital Loss, Hours per week, Country.
3. Collection process: Extraction was done by Barry Becker from the 1994 Census database.
4. Recommended uses: To have an overview of previous uses, a list of papers that site this dataset is available within the [UCI datasheet for this dataset](https://archive.ics.uci.edu/ml/datasets/Adult).

Graphical user interface, text, application, email

Description automatically generated

Figure . The motivation section of the datasheets for datasets template.

Once the dataset is well documented, we can load it and have a first look at it.

import shap

# Load the adult cencus dataset

X\_raw, Y = shap.datasets.adult()

print ("X\_raw shape:", X\_raw.shape)

X\_raw.head()

We have 32 561 examples with 12 features each. Here is what the data looks like:

A picture containing text, electronics, black

Description automatically generated

Figure . First look at the features data frame.

Please note that this dataset collected through the shap.datasets.adult() is already a clean version of the [original one from UCI](https://archive.ics.uci.edu/ml/datasets/Adult). Nevertheless, we go through the preprocessing and cleaning steps which were involved even though these do not appear in the accompanying notebook.

### Data preprocessing and cleaning

Real-world data is frequently incomplete, inconsistent, inaccurate (due to errors or outliers), and /or lacking in exact attribute values and trends. This is where data preparation comes into play: it cleans, formats, and organizes raw data, making it ready for Machine Learning models to use.

#### Identifying and handling the missing values

In our case, the original adult census dataset from UCI contained a small number of missing values represented by the character ‘?’. These were converted into nan values and dropped after detection (PS: this code was dropped from the notebook as clean data was used directly).

X\_raw[adult=='?']=np.nan

X\_raw\_new= X\_raw.dropna(axis=0)

Another strategy we could have used is filling the missing values using one of the methods suggested in the pandas.DataFrame.fillna function [documentation](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html), but we chose to omit them here as there is only a small number of them and this won’t affect our models overall performance.

We then print the number of missing values over all columns.

# Number of missing values over all columns

X\_raw.isna().sum().sum()

This prints 0 as expected.

#### One hot encoding categorical features and scaling numeric features

#### All features above look numeric, however some of them are just "numeric codes" and the features they represent are rather categorical.

#### So, for more accurate results, we separate categorical features from “real” numeric ones.

import numpy as np

categorical\_features\_indices = np.where(np.logical\_or(X\_raw.dtypes == np.int8, X\_raw.dtypes == np.int32))[0]

numeric\_features\_indices = np.where(X\_raw.dtypes == np.float32)[0]

We now define a column transformer that can be used in the modeling pipeline for processing the data before it is passed on to the models.

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

column\_transformer = ColumnTransformer ([

('onehot', OneHotEncoder(handle\_unknown='ignore'),

categorical\_features\_indices),

('scaler', StandardScaler(),

numeric\_features\_indices)

])

For categorical features, we one hot encoded them, which usually makes ML models do a better job at prediction.

For numeric features, we perform standard scaling whereby we transform our data such that its distribution will have a mean value 0 and standard deviation of 1, which also helps the ML models perform better.

### Data labeling

This is a straightforward step in our case because data is already labelled, we just convert the True and False labels into 0 and 1 respectively to be able to feed these labels into the modeling pipeline.

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

print("Before label encoding:",Y) # --> [False False False ... False False True]

Y=le.fit\_transform(Y)

print("After label encoding:",Y) # --> [0 0 0 ... 0 0 1]

For general supervised learning, if you are collecting data yourself, data labeling is a critical stage of the data pipeline since data labels must be highly accurate to teach your model to make correct predictions.

Data labeling can be done in-house or crowd-sourced, but in both cases, one often overlooked but very important step is the Quality Assurance (QA), whereby checks are performed using humans-in-the-loop to check the quality of the data before feeding it to the modeling pipeline.

## Modeling pipeline

The modeling pipeline is the second big step involved in the inner development loop. It takes as input clean preprocessed and labeled (in the case of supervised learning) data, builds a model and iteratively trains and evaluates the model each time altering the model itself or the features through feature engineering until a plausible model satisfying the requirements outlined in Phase 1 of the ML workflow is reached.

As such, the modeling pipeline as defined in this illustration guide is composed of three stages: feature engineering, model training, and model evaluation.

### Feature selection and engineering

Feature selection and engineering is a stage, which falls between the data and modeling pipelines. We chose to include it in the latter because it is a stage that is usually revisited multiple times iteratively after model training and evaluation, much more often than any of the stages involved in the data pipeline, which, if done right from the beginning, are usually only performed once.

[Feature engineering](https://en.wikipedia.org/wiki/Feature_engineering) consists of using domain knowledge to extract features from raw data, the produced features are then used by the models and highly influence prediction results.

In our case, if you look at the columns in our original dataset collected from the *shap* library compared to the original dataset from UCI, you’ll see that we performed some feature selection be eliminating the fnlwgt column which is the number of people the census believes the entry represents and does not provide any meaningful info for our loan decision task.

More sophisticated feature engineering can be done in other use cases, but for ours, you’ll see that we can already achieve pretty good accuracy with the curated features fetched from the *shap* library, so let’s go to model training now.

### Model design and training

In this section, we will train three ML classifiers. The goal behind training multiple models is to be able to find the right tradeoff between performance and Responsible AI principle guarantees:

1. Our first model will be a blackbox unmitigated model whereby transparency and fairness are sacrificed for the sake of performance.
2. Our second will be a glassbox [Explainable Boosting Machine (EBM)](https://interpret.ml/docs/ebm.html) model trying to provide better transparency while giving up some accuracy. Another approach is to use InterpretML for blackbox explainability of the first model.
3. Our third and final model will be our interpretable glassbox or blackbox model from step 2 on which we use fairness issues mitigation techniques such as threshold optimization and the reductions approach.

We will then judge which of the three above models provide the better tradeoff for our purposes during model evaluation and deploy the chosen model in the next module.

Before we get there, we give a quick overview of the Responsible AI resources we will use for our model design and training. A thorough investigations of these resources is provided in the [Leveraging Responsible AI Tooling for your non-Generative AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/nongen-ai-tooling-tutorials/docs/leveraging-responsible-ai-tooling.docx) of this Responsible AI Workshop, and only the tooling of use to us will be succinctly presented here.

#### Responsible AI resources for the modeling stage of the ML workflow

Tooling and other resources are needed to implement Responsible AI principles at each stage of the ML workflow. The modeling stage is one of the most critical stages of this workflow and subsequently requires a large amount of Responsible AI tooling addresses this stage in particular. This tooling can be categorized into 3 categories as shown in Figure 14 below:

1. Tools tounderstand the behavior of AI systems. These are used to make AI systems more fair, transparent, and inclusive.
2. Tools to protect AI systems data. These are used to make AI systems more secure and privacy-preserving.
3. Tools to establish control and governance throughout AI systems development cycle. These are used to make AI systems more reliable and allows people who design and deploy AI systems to be held accountable for how their systems operate.

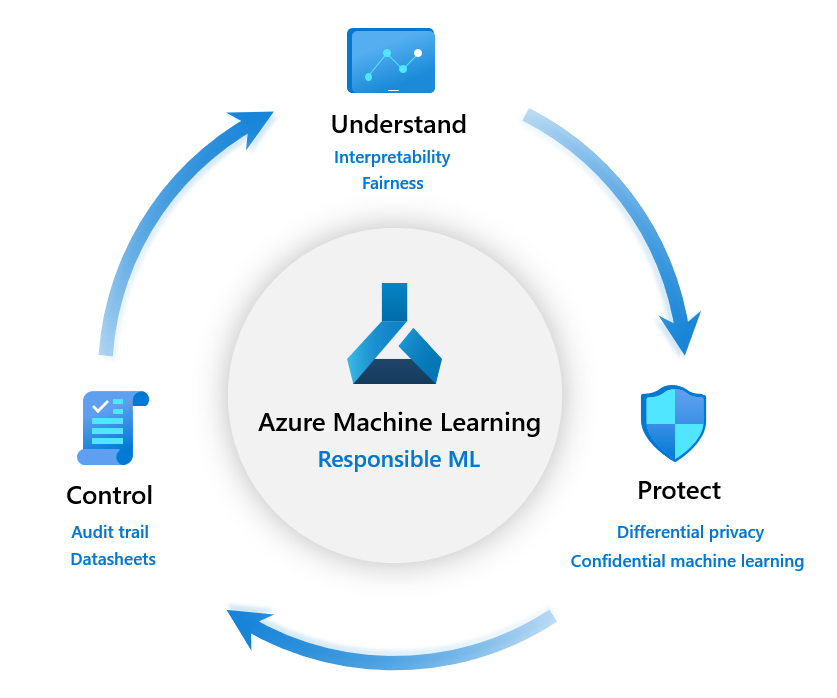


Figure . The three categories of Responsible ML resources for modeling

Due to the nature of our use case, we decided to focus on the Understandsection as follows. More specifically, we will concentrate on Interpretability and Fairness of our models, which are two major issues for our specific use case.

Tooling used to protect AI systems and data assets is carefully explored in module 3 of the abovementioned guide [Leveraging Responsible AI Tooling for your non-Generative AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/nongen-ai-tooling-tutorials/docs/leveraging-responsible-ai-tooling.docx) and accompanying tutorials can also be found [here](https://github.com/microsoft/responsible-ai-workshop/blob/main/nongen-ai-tooling-tutorials/hands-on-tutorials/protect_rai_tools.ipynb), while more information on the control Responsible AI tooling can be found in module 1 of the same guide.

Here we will use two RAI tools in particular:

1. [InterpretML](https://interpret.ml/) and its Azure ML built-in dashboards to achieve model transparency and EBM as an example of a glassbox model.
2. [Fairlearn](https://fairlearn.org/)for fairness issues detection with its Azure ML built-in dashboards and mitigation techniques for addressing these issues.

#### First model: Unmitigated Catboost classifier

Our first intuition is to start with a powerful classification model, try to fit it to our data and just see how it performs in terms of accuracy without worrying much about other metrics.

When we think about performance for classification, we think about gradient-boosted tree algorithms like [XGBoost](https://xgboost.readthedocs.io/en/latest/) or [LightGBM](https://lightgbm.readthedocs.io/en/latest/index.html). There is only one problem with such methods which is that they are not optimized for addressing problems where (one-hot encoded) categorical variables are involved, which is the case here.

Fortunately, [Catboost](https://catboost.ai/) classifier does exactly that, it is a boosting technique [optimizing decision trees for problems where categorical variables are involved](https://arxiv.org/pdf/1810.11363.pdf). Let’s fit a Catboost classifier to our data and see what the performance looks like:

# Training a Catboost Classifier

# !pip install catboost

from catboost import CatBoostClassifier

model\_1 = CatBoostClassifier(

random\_seed=42, logging\_level="Silent", iterations=150)

X\_train\_new = column\_transformer.fit\_transform(X\_train)

Y\_train\_new = Y\_train

model\_catboost = model\_1.fit(X\_train, Y\_train)

This prints the following: catboost\_classifier.score: 0.873637340703209.

We see that this score is quite good without any involved feature engineering or additional tuning. However, our goal is not to achieve the best possible performance here but rather to see how performance evolves when we start taking RAI principles into consideration in our modeling.

Let’s have a look at the Fairlearn dashboard for this model to see if we can assess any fairness issues with the Catboost classifier.

from raiwidgets import FairnessDashboard

Y\_pred = unmitigated\_predictor3.predict(X\_test)

FairnessDashboard(sensitive\_features=A\_test,

y\_true=Y\_test,

y\_pred=Y\_pred)

This shows the following dashboard:

Chart

Description automatically generated

Figure . Fairlearn dashboard 1 - Selection rate for males and females

Chart, bar chart

Description automatically generated

Figure . Fairlearn dashboard 2 - False positive (orange) and false negative (blue) rates for the two groups

We clearly see from these two dashboard sections displayed in Figure 15 and Figure 16 above that there are fairness issues with the Catboost model. The selection rate is almost three times higher for males compared to females and the same goes for the false positives rate. This means that our loan decision model tends to offer three times more loans to males than females and making three times more mistakes when deciding the men should loans. This is clearly a fairness bias that discriminates against females that should be further investigated.

Overall, the Catboost classifier we used here achieves good performance but has two major problems in terms of RAI principles we are taking into account as it is both not easily interpretable (black box) and has fairness flaws as well.

Let’s jump right into trying to solve the first problem by using EBMs for more interpretability.

#### Model Interpretability with InterpretML

If you refer to Module 2 of the guide [Leveraging Responsible AI Tooling for your non-Generative AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/nongen-ai-tooling-tutorials/docs/leveraging-responsible-ai-tooling.docx) of this Responsible AI Workshop, you’ll notice that there are two ways to make modeling more transparent:

1. Glassbox interpretability, which consists of using ML models designed from the ground-up for interpretability.
2. Blackbox explainability, which consists of techniques for explaining existing ML models, usually by exploring inputs and outputs to explain global trends in the model’s predictions both globally and locally for individual predictions.

In what follows we will show be showing how easy it is to fit a glassbox model to our problem and still achieve similar performances to our powerful Catboost classifier. Then we will proceed as if we didn’t know about EBMs to show you how you can still explain the blackbox Catboost classifier before we dive into mitigating fairness issues assessed earlier.

##### Glassbox Interpretability with EBM

We won’t dive into the details of what EBMs are here since this was already discussed in the InterpretML section of the Module 2 and the appendix of the guide [Leveraging Responsible AI Tooling for your non-Generative AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/nongen-ai-tooling-tutorials/docs/leveraging-responsible-ai-tooling.docx) we mentioned above, but in summary EBM is a powerful, interpretable, glassbox model, designed to have accuracy comparable to state-of-the-art machine learning methods like Random Forest or the Catboost classifier we explored earlier, while being highly intelligible and explainable.

We try to fit an EBM to our problem:

from interpret.glassbox import ExplainableBoostingClassifier

seed = 1

model\_2 = ExplainableBoostingClassifier(random\_state=seed, n\_jobs=-1)

pipeline\_2 = Pipeline(steps=[

('preprocessor', column\_transformer),

('classifier\_EBM', model\_2)])

ebm\_predictor = pipeline\_2.fit(X\_train, Y\_train)

print('ebm\_predictor.score:', ebm\_predictor.score(X\_test, Y\_test))

This prints the following: ebm\_predictor.score: 0.8704130201136189

We see that in terms of performance, EBM is super close to the Catboost classifier. But beyond performance, as a glassbox model EBM has the advantage of being fully interpretable with accurate feature importance, global and local explanation of predictions. We won’t be focusing on EBMs here as they are not as popular among data scientists, and you might even not be very familiar with EBMs yourself. Instead, we rather focus on explainability of our blackbox model to show achieving some transparency for such a model is still possible.

##### Catboost classifier model explainability

For this purpose, we use the [Responsible-AI-Widgets](https://github.com/microsoft/responsible-ai-widgets) interpretability dashboard. This allows us to understand which factors have the most impact on our loan acceptance/denial decisions. We can observe this for the entire population, a subset of applicants and individuals.

For example, if we are interested in the global explanation/feature importance for our blackbox model we can use TabularExplainer, which is included in Microsoft *interpretml.ext.blackbox* library as follows.

from raiwidgets import ExplanationDashboard

from interpret.ext.blackbox import TabularExplainer

explainer = TabularExplainer(catboost\_predictor,

                             X\_train)

# explain overall model predictions (global explanation)

global\_explanation = explainer.explain\_global(X\_test)

ranked\_global\_importance\_names = global\_explanation.get\_ranked\_global\_names() ranked\_global\_importance\_values = global\_explanation.get\_ranked\_global\_values()

shap.summary\_plot(np.array([ranked\_global\_importance\_values]),

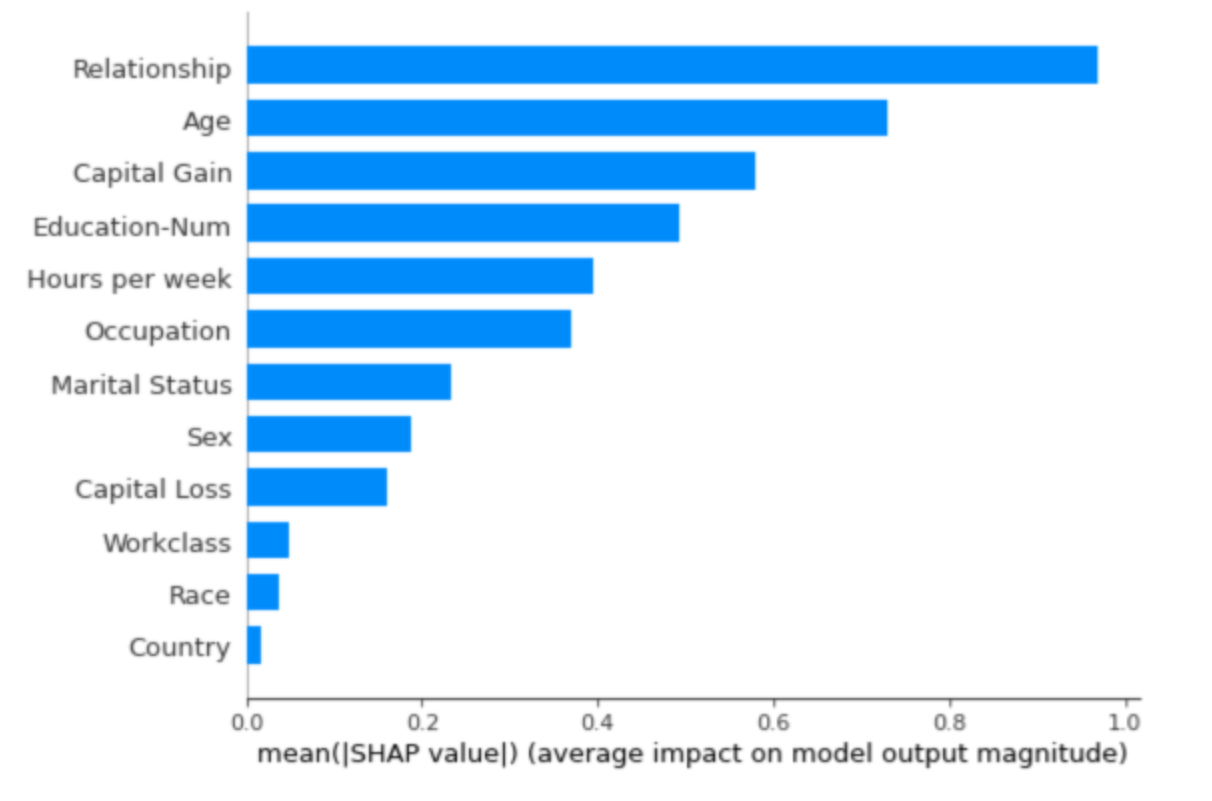
ranked\_global\_importance\_names, plot\_type="bar")

Figure . Feature importance for the Catboost classifier

We see that apparently whether a client is in a relationship or the client’s age are more important for our model’s decision than the capital gain or the number of years of education of the client, which could seem counter intuitive but achieves high accuracy and might be worth investigating.

#### Assessing and mitigating fairness issues

By using InterpretML to explain the trends in the predictions of our Catboost classifier or using EBM instead, we have tackled the transparency problem; now our model is interpretable. But a second major problem remains, that of fairness between predictions for men and women we showed in the beginning of this section through the Fairlearn dashboard.

To solve these fairness issues, we use mitigation algorithms from the Fairlearn package available by default in Azure ML. These were discussed in detail in module 2 of the guide [Leveraging Responsible AI Tooling for your non-Generative AI-powered solutions](https://github.com/microsoft/responsible-ai-workshop/blob/main/nongen-ai-tooling-tutorials/docs/leveraging-responsible-ai-tooling.docx).

Which mitigation technique to choose depends on several factors, like the specific industry which the model is used for. They may also depend on existing constraints: for example, if the model cannot be retrained then we may use different thresholds per group to calibrate it in a post-processing step.

Here, since we do not have any such constraints, we choose the most powerful “Reduction approach”, which is the mitigation technique that provides the best tradeoff between fairness and accuracy by tuning the model it receives as a blackbox. It does so by generating a set of retrained models using a sequence of reweighted training datasets as the code snippet hereafter illustrates.

from fairlearn.reductions import GridSearch

from fairlearn.reductions import DemographicParity, ErrorRate

sweep = GridSearch(

model\_1,

constraints=DemographicParity(),

grid\_size=70)

sweep.fit(X\_train, Y\_train, sensitive\_features=A\_train.Sex)

mitigated\_predictors = sweep.\_predictors

ys\_mitigated\_predictors = {} # it contains (<model\_id>, <predictions>) pairs

# the original prediction:

ys\_mitigated\_predictors["census\_unmitigated"]=catboost\_predictor.predict(X\_test)

base\_predictor\_name="mitigated\_predictor\_{0}"

model\_id=1

for mp in mitigated\_predictors:

id=base\_predictor\_name.format(model\_id)

ys\_mitigated\_predictors[id]=mp.predict(X\_test)

model\_id=model\_id+1

FairnessDashboard(

sensitive\_features=A\_test,

sensitive\_feature\_names=np.array(A\_test.columns),

y\_true=Y\_test,

y\_pred=ys\_mitigated\_predictors)

We leave the discussion of the results to the next section on Model evaluation.

### Model evaluation

To evaluate our models, we use several metrics from *scikit-learn* and the *fairlearn* packages. We define the get\_metrics\_df function below, which returns these metrics as a pandas DataFrame given a set of models.

# Metrics

from fairlearn.metrics import (

MetricFrame,

selection\_rate, demographic\_parity\_difference, demographic\_parity\_ratio,

false\_positive\_rate, false\_negative\_rate,

false\_positive\_rate\_difference, false\_negative\_rate\_difference,

equalized\_odds\_difference)

from sklearn.metrics import balanced\_accuracy\_score, roc\_auc\_score

# Some helper functions to be used later

def get\_metrics\_df(models\_dict, y\_true, group):

metrics\_dict = {

"Overall selection rate": (

lambda x: selection\_rate(y\_true, x), True),

"Demographic parity difference": (

lambda x: demographic\_parity\_difference(y\_true, x, sensitive\_features=group), True),

"Demographic parity ratio": (

lambda x: demographic\_parity\_ratio(y\_true, x, sensitive\_features=group), True),

"------": (lambda x: "", True),

"Overall balanced error rate": (

lambda x: 1-balanced\_accuracy\_score(y\_true, x), True),

"Balanced error rate difference": (

lambda x: MetricFrame(balanced\_accuracy\_score, y\_true, x, sensitive\_features=group).difference(method='between\_groups'), True),

" ------": (lambda x: "", True),

"False positive rate difference": (

lambda x: false\_positive\_rate\_difference(y\_true, x, sensitive\_features=group), True),

"False negative rate difference": (

lambda x: false\_negative\_rate\_difference(y\_true, x, sensitive\_features=group), True),

"Equalized odds difference": (

lambda x: equalized\_odds\_difference(y\_true, x, sensitive\_features=group), True),

" ------": (lambda x: "", True),

"Overall AUC": (

lambda x: roc\_auc\_score(y\_true, x), False),

"AUC difference": (

lambda x: MetricFrame(roc\_auc\_score, y\_true, x, sensitive\_features=group).difference(method='between\_groups'), False),

}

df\_dict = {}

for metric\_name, (metric\_func, use\_preds) in metrics\_dict.items():

df\_dict[metric\_name] = [metric\_func(preds) if use\_preds else metric\_func(scores)

for model\_name, (preds, scores) in models\_dict.items()]

return pd.DataFrame.from\_dict(df\_dict, orient="index", columns=models\_dict.keys())

After running this get\_metrics\_df function for the unmitigated Catboost classifier, we obtain the following metrics:

* Overall selection rate: 0.340396
* Demographic parity difference: 0.305226
* Demographic parity ratio: 0.306794
* Overall balanced error rate: 0.15928
* Balanced error rate difference: 0.0426844
* False positive rate difference: 0.195099
* False negative rate difference: 0.10973
* Equalized odds difference: 0.195099
* Overall AUC: 0.927269

The overall performance measure we consider here is the Overall AUC which is equivalent to the balanced accuracy for classification problems and is suited for our use case.

As the fairness metric we use Equalized odds difference, which quantifies the disparity in accuracy experienced by different demographics. Our goal is to assure that neither of the two groups ("male" vs. "female") has substantially larger false-positive rates or false-negative rates than the other group. The equalized odds difference is equal to the larger of the following two numbers:

1. The difference between false-positive rates of the two groups
2. The difference between false-negative rates of the two groups.

Which means that the closer to zero the Equalized odds difference is, the better. So, in the graph below, the closer we are to the right lower corner of the graph the better.

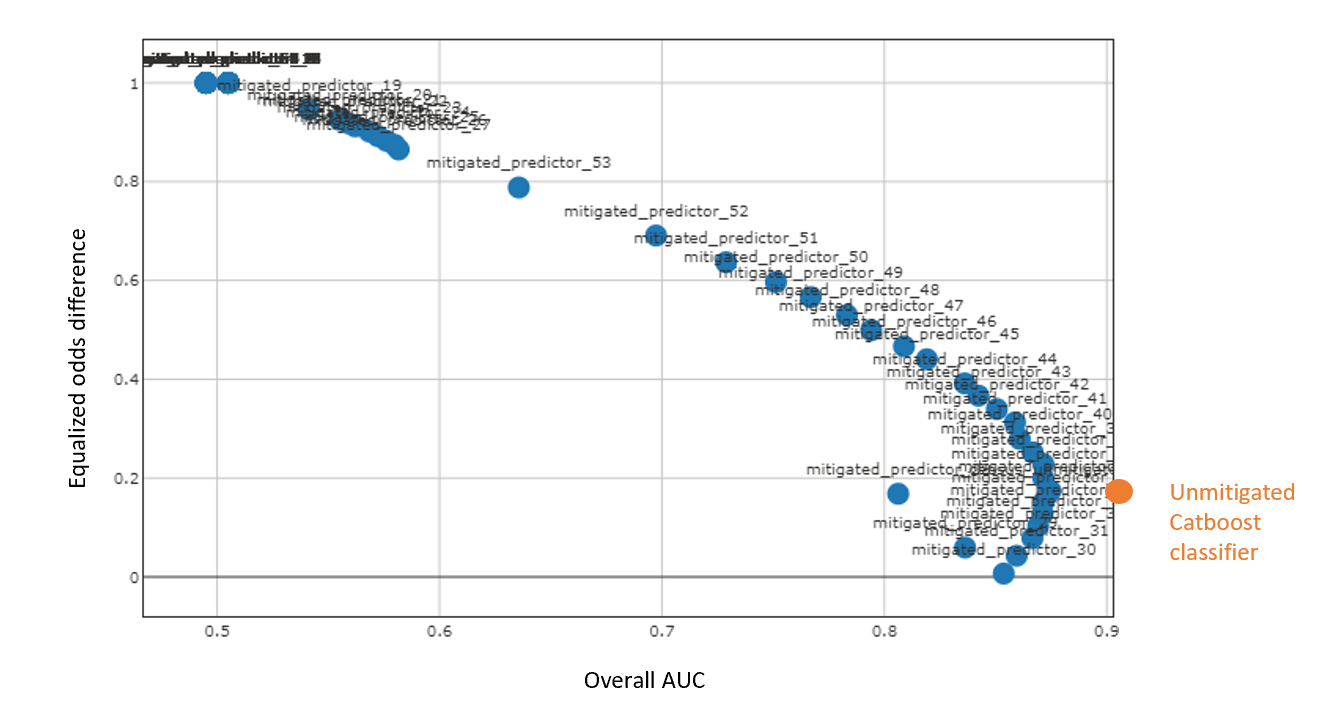


Figure . Tradeoff between fairness (Equalized odds difference) and accuracy (Overall AUC) for the models we trained.

With the [GridSearch](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) algorithm, we can see from Figure 18 above that we can the mitigated predictors it provides offer a wide range of accuracy/fairness trade-offs to choose from. Some mitigated models achieve by z very low Equalized odds difference while maintaining the same order of performance as the unmitigated initial model.

Choosing which model is best will depend on the business stakeholder priorities. If we can afford to lose a little bit of performance (around 7% accuracy lost from 92% to 0.85%) to achieve a perfect level of parity then that’s what we should go for, if not, we can choose another trade-off that best suits our goals.

Thus, at this stage, we have made good progress in solving both the transparency and fairness issues of our Catboost loan decision model and we are ready to move to the deployment stage.

### Local model deployment

We now shift our attention to model deployment, which is both tackled at the end of the current Phase 2, i.e., the inner development loop, and at the beginning of Phase 3, i.e., the outer deployment loop.

*Why is this separation needed?* *Why not address model deployment entirely as part of the outer loop?* Those are all very valid questions.

The answer to them lies in the separation of concerns between the different stakeholders in the value chain of model deployment: the data scientist and the DevOps/software engineer:

* A data scientist job requires him to test his model locally before shipping the model artifact (an .onnx file for example). This is to make sure that the model’s registration and inference configuration work when deploying the model to a local endpoint. So local deployment is an integral part of what the data scientist should do.
* A DevOps/software engineer (with possibly the help of [AI engineer](https://searchenterpriseai.techtarget.com/definition/machine-learning-engineer-ML-engineer)) will usually receive a model artifact as an input and will be tasked to deploy this artifact on some compute targets (for example a Kubernetes cluster) using custom *release* pipelines.

The goal here is to tackle local deployment to complete the inner development loop and the tasks a data scientist should undertake, then focus on the deployment of models on different (intended) remote compute targets using pipelines in Phase 3, which will be targeted at DevOps/software engineers.

That being said, there is a lot of common ground between deployment on a local and remote compute target. But before we get there let’s first look at the different compute target options. With Azure ML, the most used [compute targets](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-and-where?tabs=azcli#choose-a-compute-target) for ML model deployment include:

* [Local web service](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-local-container-notebook-vm) for testing/debugging.
* [Azure Container Instance (ACI)](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-azure-container-instance) for real-time inference, recommended for dev/test purposes only.
* [Azure Kubernetes Service (AKS)](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-azure-kubernetes-service) for real-time inference as well, recommended for production workloads.
* [Azure ML compute clusters](https://docs.microsoft.com/en-us/azure/machine-learning/tutorial-pipeline-batch-scoring-classification) for batch inference.

Here is a diagram to help you choose the right compute target:

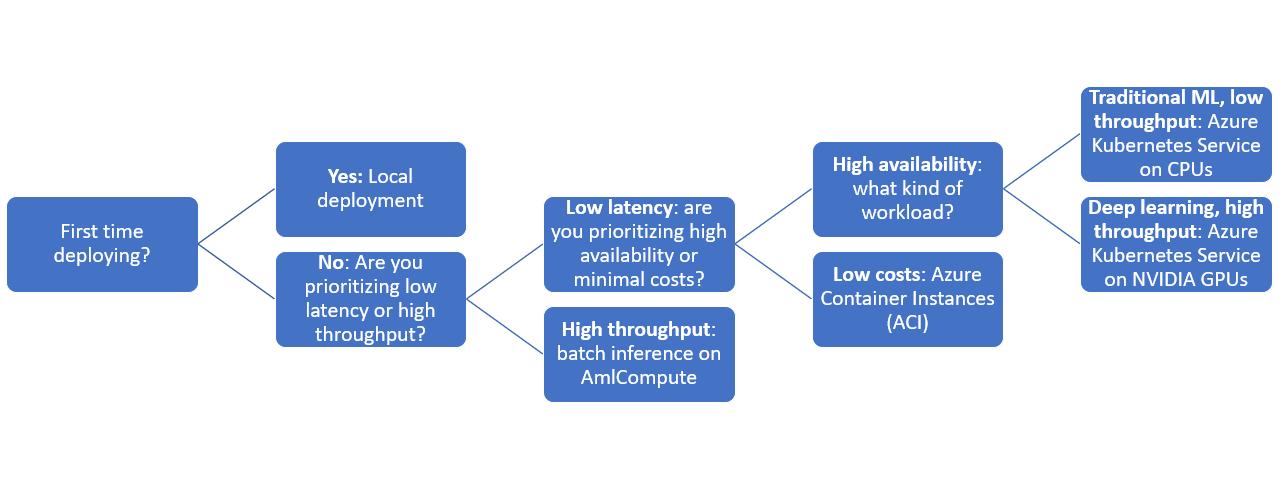


Figure . Block diagram for choosing a compute target for your ML model deployment.

But the good news is that the [model deployment workflow](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-and-where?tabs=python) is similar no matter where you choose to deploy your model and consists of the following steps:

1. Save and register the model
2. Prepare an entry script
3. Prepare an inference configuration
4. Define a deployment configuration based on the chosen compute target
5. Deploy the model
6. Test our model deployment by consuming the endpoint

**Note** Steps 1 to 3 are to be completed once and are independent of the chosen compute target. So, if you have already deployed your model to any compute target (locally, ACI, AKS etc.) and want to deploy to another target then only steps 4 to 6 are to be undertaken.

We will follow these steps sequentially. We will be deploying our model locally and on an [Azure Container Instance (ACI)](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-azure-container-instance) here, but you should be able to adapt this code to any other compute target you choose following the same workflow.

#### Step 1: Model saving and registration

For a typical ML service deployment, you need the following components:

* A resource representing the specific model that you want deployed (for example: a model file).
* Code that you will be running in the service, that executes the model on a given input.

Azure ML allows you to separate the deployment into these two separate components, so that you can keep the same code, but merely update the model as needed. We define the mechanism by which you upload a model separately from the code running it as "registering the model".

When you register a model, you upload the model to the cloud (in your workspace's default storage account), and then mount it to the same compute where your webservice is running for deployment.

We first save the model to a file. In our illustration, we use [Open Neural Network Exchange (ONNX)](https://onnx.ai/) (but you can use other formats if you wish. The pickle file format is used for example when the model is saved in the attached [tutorial](https://github.com/kenzabenjelloun/responsible-ai-workshop/blob/main/nongen-ai-lifecycle-walkthrough/responsible_ai_lifecycle_walkthrough.ipynb)).

ONNX is an open-source standard for representing ML algorithms and providing ML interoperability. For that purpose, and as outlined on the related [GitHub repo](https://github.com/onnx/onnx), it defines an extensible computation graph model, as well as definitions of built-in operators and standard data types - the building blocks of traditional ML models and deep learning models - (currently focused on the capabilities needed for inferencing (scoring)).

As such, the open-source file format ONNX is widely supported and can be found in many frameworks, tools, and hardware. So, it is best suited for our purposes in so far as the related open ecosystem empowers ML engineers to use models with a variety of frameworks, tools, runtimes, and compilers.

Enabling interoperability between different frameworks and streamlining the path from research to production helps increase the speed of innovation in the AI community.

# Save model to ONNX-ML format

catboost\_predictor.save\_model(

"catboost\_predictor.onnx",

format="onnx",

export\_parameters={

'onnx\_domain': 'ai.catboost',

'onnx\_model\_version': 1,

'onnx\_doc\_string': 'loan decision model',

'onnx\_graph\_name': 'CatBoostModel\_for\_loan)decision'

}

)

We can quickly verify if our model was correctly saved using an onnxruntime inference session.

# Check model was correctly saved

import numpy as np

import onnxruntime as rt

df = X\_test.head()

sess = rt.InferenceSession('catboost\_predictor.onnx')

input\_name = sess.get\_inputs()[0].name

pred\_onx = sess.run(None, {input\_name: df.to\_numpy().astype(np.float32)})

print(pred\_onx[0].tolist())

This prints the result [0, 0, 0, 0, 0] as expected.

We can then register our model to Azure ML workspace, where it is represented by a name and a version, using the [Model class](https://docs.microsoft.com/en-us/python/api/azureml-core/azureml.core.model.model?view=azure-ml-py).

# Register model to Azure ML model registry

from azureml.core.model import Model

model = Model.register(workspace=ws,

model\_name='catboost\_predictor\_onnx', # Name of the registered model in your workspace.

model\_path='catboost\_predictor.onnx', # Local ONNX model to register as a model.

model\_framework=Model.Framework.ONNX , # Framework used to create the model.

model\_framework\_version='1.3', # Version of ONNX used to create the model.

description='Onnx loan decision model')

We do a quick sanity check to verify the model is correctly registered:

# Check model is correctly registered

print( "Name: ", model.name)

print("Version: ", model.version)

This prints the following:

Name: catboost\_predictor.onnx

Version: 1

Now that the model is registered to Azure ML model registry, we are ready to write the code that will call into our model.

#### Step 2: Prepare an entry script

The entry script receives data submitted to a deployed web service and passes it to the model. It then returns the model's response to the client. The script is specific to your model.

The two things you need to accomplish in your entry script are:

1. Loading your model (using a function called init())
2. Running your model on input data (using a function called run())

For our initial local deployment, we use the following entry script.

import joblib

from azureml.core.model import Model

import os

import json

import pandas as pd

def init():

# Create a global variable called model

global model

# Load the model using the name of the model you registered

model\_path = Model.get\_model\_path("catboost\_predictor")

print("Model Path is ", model\_path)

#Load the model from the path

model = joblib.load(model\_path)

def run(request):

try:

data = json.loads(request)

df = pd.read\_json(data)

result = model.predict(df)

return {'data' : result.tolist() , 'message' : "Successfully classified loan"}

except Exception as e:

error = str(e)

return {'data' : error , 'message' : "Failed to classify loan"}

Here is what the above entry script does:

* In the init method, we load the model so that it is ready to use.
* In the run method, upon reception of a request, we load the data in json format, convert it to DataFrame format, run our model on it and return the results or an error message if an error occurs.

You can [download this entry script directly from the repository](https://github.com/microsoft/responsible-ai-workshop/blob/main/nongen-ai-lifecycle-walkthrough/scoring_pkl/score.py) and put in a new directory called source\_dir at the root of your workspace (next to the responsible\_ai\_lifecycle\_walkthrough.ipynb notebook).

A screenshot of a computer

Description automatically generated with medium confidence

Figure . Screenshot of the Azure workspace after entry script creation

#### Step 3: Define an inference configuration

An inference configuration describes the Docker container and files to use when initializing your web service. All the files within your source directory, including subdirectories, will be zipped up and uploaded to the cloud when you deploy your web service.

Back into the notebook, the inference configuration below specifies that the ML deployment will use the file score.py in the source\_dir directory to process incoming requests and that it will use the Docker image with the Python packages specified in the project\_environment environment that we define.

from azureml.core import Environment

from azureml.core.conda\_dependencies import CondaDependencies

from azureml.core.model import InferenceConfig

env = Environment(name="project\_environment")

conda\_dep = CondaDependencies()

# Installs azure-ml-api-sdk package

conda\_dep.add\_pip\_package("azure-ml-api-sdk")

# Installs catboost package

conda\_dep.add\_pip\_package("catboost")

# Installs numpy package

conda\_dep.add\_pip\_package("numpy")

# Installs pandas package

conda\_dep.add\_pip\_package("pandas")

# Installs sklearn package

conda\_dep.add\_pip\_package("sklearn")

# Adds dependencies to PythonSection of env

env.python.conda\_dependencies=conda\_dep

dummy\_inference\_config = InferenceConfig(

environment=env,

source\_directory="./source\_dir",

entry\_script="./score.py",

)

In the code snippet/cell above, we add few pip packages to our environment’s conda dependencies. This was an iterative process whereby we went back to this after trying to deploy the model using the steps below and finding out these dependencies were missing. Please do the same with any missing dependencies from your own use-case deployment.

For more information on the use of software environments in Azure ML you can refer to the article [Create & use software environments in Azure Machine Learning](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-use-environments).

#### Step 4: Define a deployment configuration

In general, a deployment configuration specifies the amount of memory and cores to reserve for your webservice will require to run, as well as configuration details of the underlying webservice.

The options available for a deployment configuration differ depending on the compute target you choose. In a local deployment, all you can specify is which port your webservice will be served on.

from azureml.core.webservice import LocalWebservice

deployment\_config = LocalWebservice.deploy\_configuration(port=6789)

#### Step 5: Deploy the machine learning model

We are now ready to deploy our ML model.

service = Model.deploy(

ws,

"myservice",

[model],

dummy\_inference\_config,

deployment\_config,

overwrite=True,

)

service.wait\_for\_deployment(show\_output=True)

print(service.get\_logs())

The Model.deploy() function deploys a webservice as a real-time endpoint that can be used for inference requests. For more information, see the documentation for [Model.deploy()](https://docs.microsoft.com/en-us/python/api/azureml-core/azureml.core.model.model?view=azure-ml-py#deploy-workspace--name--models--inference-config-none--deployment-config-none--deployment-target-none--overwrite-false--show-output-false-) and [Webservice](https://docs.microsoft.com/en-us/python/api/azureml-core/azureml.core.webservice.webservice).

The first execution of this step takes a lot of time since it sets up Docker configuration, scans for dependencies, build the docker image and launches the container. Further executions normally take less time if no changes are brought to the environment/ the inference configuration.

#### Step 6: Call into your model by consuming the endpoint

It is now time to check that our dummy echo model is deployed successfully by calling into the endpoint.

import requests

import json

uri = service.scoring\_uri

headers = {"Content-Type": "application/json"}

json\_test = X\_test.head().to\_json()

data = json.dumps(json\_test)

response = requests.post(uri, data=data, headers=headers)

print(response.json())

Note that service.scoring\_uri retrieves the address on which the model is served, in our case this is <http://localhost:6789/score>. Then we retrieve the first five examples from our test set in the json format and include them in the post request.

The cell above prints the following:

{'data': [0, 0, 0, 0, 0], 'message': 'Successfully classified loan'}

This means that our model was correctly deployed, and we successfully classified the first 5 entries of the test set we submitted in our request to the model serving endpoint.

#### Deployment on Azure Container Instance (ACI): Repeating Steps 4 to 6 with another deployment target

In what follows, we [redeploy our model on an ACI webservice](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-azure-container-instance) this time. We do this by repeating steps 4 to 6, that is using the same inference configuration (and entry script) but defining a new deployment config and redeploying the model with this new config specifying [ACIWebservice](https://docs.microsoft.com/en-us/python/api/azureml-core/azureml.core.webservice.aciwebservice#deploy-configuration-cpu-cores-none--memory-gb-none--tags-none--properties-none--description-none--location-none--auth-enabled-none--ssl-enabled-none--enable-app-insights-none--ssl-cert-pem-file-none--ssl-key-pem-file-none--ssl-cname-none--dns-name-label-none--primary-key-none--secondary-key-none--collect-model-data-none--cmk-vault-base-url-none--cmk-key-name-none--cmk-key-version-none-) as a compute target.

We do all this in a single cell below.

from azureml.core.webservice import AciWebservice, Webservice

from azureml.core.model import Model

# Step 4: Define deployment config

deployment\_config = AciWebservice.deploy\_configuration(cpu\_cores = 1, memory\_gb = 1)

# Step 5: Deploying the model

service = Model.deploy(ws, "aciservice", [model], inference\_config, deployment\_config)

service.wait\_for\_deployment(show\_output = True)

# Step 6: Consuming the endpoint

uri = service.scoring\_uri

headers = {"Content-Type": "application/json"}

json\_test = X\_test.head().to\_json()

data = json.dumps(json\_test)

response = requests.post(uri, data=data, headers=headers)

print(response.json())

As expected, the cell above also prints the following:

{'data': [0, 0, 0, 0, 0], 'message': 'Successfully classified loan'}

This shows how easy it is to switch from one compute target to the other because of the intersection in the steps require to perform these deployments, especially the inference configuration step.

We can’t explore all compute targets in this illustration guide, but you should be able to [deploy your models on Azure Kubernetes Service cluster](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-azure-kubernetes-service?tabs=python) (potentially using [Dev Spaces for AKS](https://azure.microsoft.com/en-us/blog/introducing-dev-spaces-for-aks/) for simpler deployments) or more advanced deployments using [Azure Machine Learning pipelines for batch scoring](https://docs.microsoft.com/en-us/azure/machine-learning/tutorial-pipeline-batch-scoring-classification) by following the tutorials in the links pointing to Azure ML documentation.

This concludes this module and the inner development loop implementation. Let’s now investigate model deployment considerations and deploy our loan decision model.

# Phase 3: Implementing the outer deployment loop with RAIL “validate and support” stage recommendations

Now that we have completed the modeling phase and chose the model that provides the best tradeoff between performance and Responsible AI principles for our purposes during model evaluation and tested the deployment of our model locally, we are ready to deploy our model on a larger scale in production-like environments and automate these deployments with Azure DevOps pipelines. By combining Azure ML and Azure DevOps, you can effectively and cohesively manage your datasets, experiments, models, and ML-powered cloud-native applications.

This module is mainly targeted towards DevOps/software engineers (along with the ML engineers) who want to include a ML model serving microservice into their cloud-based system.

There are two objectives here:

1. From one side, we want to show you how to set up your pipelines and automate your deployment with a step-by-step approach,
2. And from the other side of things, we want to show you how to do so in a more responsible manner.

To get there, we will be following RAIL launch and deploy stages recommendations and giving you a high-level overview of the deployment workflow, while leaving the step-by-step instructions to Appendix; not to overwhelm you with the details. In particular, we will focus on three responsible deployment activities: documentation, gating and scenario attestation.

You are highly encouraged to go through the steps in Appendix in parallel with reading the second section of this module if you have a model artifact (an .onnx file for example) obtained by following the steps in the first two modules. If you just want an overview of the deployment workflow and the associated RAI activities, you can skip Appendix and read through the sections of this module sequentially.

## Documentation

Documenting software is one of the most important phases of any software application. Intelligent, i.e., AI-powered, cloud-native applications are no exception to this rule.

Through the [Team Data Science Process (TDSP)](https://docs.microsoft.com/en-us/azure/architecture/data-science-process/overview#tools-and-utilities-for-project-execution), Microsoft provides a GitHub repository containing [templates for the folder structure](https://github.com/Azure/Azure-TDSP-ProjectTemplate) for documenting the ML workflow. This folder structure organizes the files that contain code for data exploration and feature extraction, and that record model iterations and document them. These templates make it easier for team members to understand work done by others and to add new members to teams if the same folder structure is replicated for all data science projects. In our case we will include all the documentation in a Docs folder.

We propose the following structure for the Docs folder of your repo:

Docs

└── Datasheets\_for\_datasets

└── Data\_reports

└── Model

└── Project

Here is a short description of each of these subfolders:

* Projectcharter to document the business problem and scope of the project.
* Datasheets for datasets include the documentation of each dataset as described in the Data pipeline section of Phase 2.
* Data reportsdocument the structure and statistics of the raw data.
* Model reports document the derived features and performance metrics.

## Gating - Closing the outer feedback loop with Azure DevOps Pipelines

### Overview

To bring the loan decision model into production and to simplify future (automated and reproductible) deployments, you can create via Azure DevOps custom pipelines to configure continuous integration/continuous delivery/deployment (CI/CD) from commit changes to your model to releases of your project.

Azure DevOps provides developer services to support teams to plan work, collaborate on code development, and build and deploy (cloud-native) applications.

Amongst the integrated features, Azure Pipelines, equipped with built-in Azure ML tasks, allow you to easily configure pipelines to train and to deploy newer versions of your model. These pipelines are meant to ease collaboration between data scientists and software/ML engineers, making model versioning and lifecycle management easier.

The goal of gating in such a Dev(Sec)Ops environment is to ensure a stable production environment: the next pipeline is not executed if the previous pipeline fails. This is ensured by using triggers on the success of the previous pipeline or on an artifact produces by the successful execution of this pipeline.

You will learn in this module how to set up pipelines to automate the deployment of your loan decision ML model as well as automate its deployment. We will go again through all the previous modules and build pipelines to connect each step.

The complete scenario we consider here is as follows:

1. The data scientist writes/updates the loan decision data processing and/or modeling code, uploads training/testing data if needed and pushes it to the *git* repo, which triggers the first Azure DevOps model training pipeline.

Once the model training pipeline is triggered, it performs data preprocessing and launches the training of the model using the new data processing / model training code and data pushed to the repo. It then publishes a model artifact (an .onnx file in our case) and registers it to the Model Registry of our Azure ML Workspace.

1. Once the ML model is registered on Azure ML it acts as a *release* pipeline trigger, which will run the second Azure Devops *build-push* pipeline. The set-up of this pipeline is done by a DevOps/software engineer.

It fetches the model from the Model Registry and builds a Docker image which it pushed to our [Azure Container Registry (ACR)](https://azure.microsoft.com/en-us/services/container-registry/).

1. The last Azure DevOps *release* pipeline will fetch the model from the registry and deploy it on specified compute targets. This process can be split into different release stages; a Quality Assurance (QA) stage where we deploy a model on [Azure Container Instances (ACI)](https://azure.microsoft.com/en-us/services/container-instances/) to test our deployment, and a Production (prod) stage where we deploy on [Azure Kubernetes Service (AKS)](https://azure.microsoft.com/en-us/services/kubernetes-service/).

We’ve deliberately given you the full version of the scenario above. However, in this section we omit talking about the model training pipeline because this module is dedicated to DevOps/software engineers (along with ML engineers) who want to include ML capabilities in their microservices architecture of the considered cloud-native application. Nevertheless, we have given some step-by-step directions on how to set-up a training pipeline in the bonus section of the Appendix should you wish to do so.

So, for this scenario to work as planned from the DevOps/software engineer perspective, we need to define two Azure Devops pipelines:

1. A build-push pipeline which is triggered by the model artifact and produces a containerized version of the model stored as a docker image.
2. A *release* pipeline which will deploy images into specified compute targets like ACI or AKS following release stages.

The first two pipelines are continuous integration (CI) pipelines and are meant to be conducted by the person implementing and maintaining the ML models, a data scientist in general. The last pipeline is to be implemented and launched by a DevOps/software engineer integrating the ML model as/into a microservice into a larger (AI) system.

We will now provide detailed instructions on configuring the two pipelines. But before we get there, we need to correctly setup our Azure DevOps environment and link it to our Azure ML workspace and the other Azure services we will use with service connections. Which means that this section will be divided into 4 subsections, the first one for setting up the Azure DevOps environment, and the subsequent two to build the 3 pipelines mentioned above.

### Setting up an Azure DevOps environment

To set up the Dev(Sec)Ops environment needed for the rest of this section, perform the following steps:

1. Sign in to [Azure DevOps](https://azure.microsoft.com/en-us/services/devops/?nav=min) (with your Azure account). You can click on Start free if it is your first time using Azure DevOps or [Sign in to Azure DevOp](https://go.microsoft.com/fwlink/?LinkId=2014676&githubsi=true&clcid=0x409&WebUserId=50395eaee24b4a028eaf6120cb6392cc)sif you already have an account.

Graphical user interface, text, application, email

Description automatically generated

Figure . Azure DevOps sign in page

1. An organization is created based on the account you used to sign in by clicking Create New Organizationif it is your first-time using Azure DevOps. Now you can create a new project, fill in the project name and description and hit Create.

Graphical user interface, application

Description automatically generated

Figure . Create new project pane on Azure DevOps

1. You now need to install the Azure Machine Learning extension to your organization from the [Visual Studio Code marketplace](https://marketplace.visualstudio.com/items?itemName=ms-air-aiagility.vss-services-azureml), so that you can set up a service connection to your Azure ML workspace. You can simply click on Get it free*,* and then follow the instructions to install the extension to your Azure DevOps organization.

Graphical user interface, text, application, email

Description automatically generated

Figure . Machine Learning Extension for Azure DevOps

1. The fourth step is creating a service connection to your Azure ML workspace. Back to your Azure DevOps project’s pane, go to your project settings by clicking on the wheel icon on the bottom left.

Graphical user interface, application, Teams

Description automatically generated

Figure . Project settings location in the project overview screen

Then go to the Service connections section and select Azure Resource Manager when creating New service connection. You can then select the recommended Service principal authentication method.

Graphical user interface, text, application

Description automatically generated

Figure : Service connections settings location.

Graphical user interface, text, application

Description automatically generated

Figure . New service connection pane - select Azure Resource Manager as connection type

Graphical user interface, text, application

Description automatically generated

Figure . Authentication method pane

Graphical user interface, text, application, email

Description automatically generated

Figure . Service connection creation pane

**Note** Creating service connection using Azure Machine Learning extension requires 'Owner' or 'User Access Administrator' permissions on the Workspace.

1. You now need to link your GitHub repository to your Azure DevOps workspace. To do that go to the Repos section in the Azure DevOps Dashboard. You’ll land on the page below where you can select Import then paste your cloning URL from GitHub and hot Import again.

Graphical user interface, text, application, email

Description automatically generated

Figure . The Repos pane of Azure DevOps

Graphical user interface, text, application, email

Description automatically generated

Figure . Importing your git repository to Azure DevOps

At this stage, your Azure DevOps environment is all set now, and we can jump right off to implementing the first training pipeline.

### Pipeline 1: *Build Push*

The first pipeline we want to set up is a continuous integration (CI) pipeline, which should be triggered once a new model is registered to Azure ML, i.e., say when the training pipeline (like the one described in the appendix), if any, succeeds or if the model is registered manually to Azure ML from a notebook (like in our case) or a training script for example.

For that, you can set up a *release* pipeline with a model artifact that will automatically trigger the Build Push pipeline every time there is a new model.

Here is an overview of the pipeline and the tasks it contains along with a quick description of each task.

Graphical user interface

Description automatically generated

Figure . Build Push pipeline overview

Graphical user interface, text, application

Description automatically generated

Figure . Build Push pipeline tasks

The tasks undertaken in the Build Push pipeline are the following:

1. Use Python 3.7: this task only specifies the python version to use when running Python scripts for the rest of the pipeline.
2. Install azureml-core package: this task installs the package needed to connect to the Azure ML workspace to fetch the models.
3. Create docker\_context folder: this folder will be used later one when building the docker image.
4. Run retrieve\_models Python script: this script downloads the Azure ML model to the docker\_context folder.
5. Copy source to docker\_context: this task copies Dockerfile, scoring script and requirements file from repo to *docker\_context*.
6. buildAndPush: this task builds the docker image using the Dockerfile and the docker context and push it to Azure Container Registry.

Detailed instructions for setting up this pipeline are provided in the Appendix. Please refer to them if needed.

### Pipeline 2: *Release*

The last Azure DevOps pipeline we implement is the *release* pipeline. It is triggered by a Docker image being pushed to Azure Container Registry (in other words, the completion of the *build-push* pipeline) and deploys a container on different compute targets following release stages as described in the overview; a Quality Assurance (QA) stage where we deploy a model on Azure Container Instances (ACI) to test our deployment, and a Production (prod) stage where we deploy on Azure Kubernetes Service (AKS).

Here is an overview of the Release pipeline on Azure DevOps.

Graphical user interface

Description automatically generated

Figure . Release pipeline overview

Couple resources are needed on your Azure workspace to be able to run this pipeline, namely:

* An Azure Container Registry, you can find instructions on how to create it [here](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-get-started-portal).
* An Azure Kubernetes Service (AKS) cluster, you can find instructions on how to create it [here](https://docs.microsoft.com/en-us/azure/devops/pipelines/apps/cd/deploy-aks?view=azure-devops&tabs=python#create-an-aks-cluster-to-host-your-app).

This pipeline has two artifacts and contains a single task. An overview of the artifacts fed into the pipeline and the tasks undertaken at each release stage are described below.

In terms of artifacts, the pipeline takes as input:

* The Azure Container Repository: The repository of your Azure Container Registry in which you stored the image in the Build Push pipeline. We called it \_loan-decision-container in our illustration.
* The Access to the GitHub repository.

The first stage has a single task which uses a PowerShell script to run an Azure CLI command deploying the container on ACI.

As for the Build Push pipeline, detailed instructions for the release pipeline are left for the Appendix, you are encouraged to check them out and follow them to build your own Release pipeline.

This concludes this module and the last phase of the ML workflow and the outer deployment loop and the related Responsible AI activities.

## Using Confidential Computing for ML

The next core Responsible AI principle we would like to focus on for our model deployment is privacy and security.

While the previous production stage deployment options might prove sufficient in terms of privacy and security guarantees, an increasing number of use cases require stronger guarantees. Take finance organizations as an example in the context of our illustration or healthcare ones, both require strong privacy and security guarantees for their ML use cases to protect their customers vs. patients’ data.

Our loan decision model falls under this category, which is the reason we investigate confidential computing below. We first introduce confidential computing and Azure Confidential Computing before we give an overview and pointers to instructions to use confidential computing nodes as part of your previous AKS deployment.

### Introducing confidential computing and Azure Confidential Computing capabilities

Confidential computing offers a protection that for years has been missing from public clouds: encryption of data while in use. It extends the baseline security guarantees of data encryption at rest and in transit, to hardware-enforced cryptographic protection of data during computation. This means that data can be processed in the cloud with the assurance that it is always under customer control.

Microsoft has been at the forefront of the Confidential computing effort, with Microsoft Research (MSR) [Confidential computing team](https://www.microsoft.com/en-us/research/theme/confidential-computing/) at the driving seat with the following vision: Transforming the Azure cloud to the Azure confidential cloud enabling the industry to move from computing in the clear to computing confidentially in the cloud and the edge.

With the [announcement](https://azure.microsoft.com/en-us/blog/introducing-azure-confidential-computing/) of Azure Confidential Computing in May 2018, Microsoft became the first cloud services provider to enable new data security capabilities that protect customer data while in use.

[Azure Confidential Computing (ACC)](https://azure.microsoft.com/en-us/solutions/confidential-compute/#overview) provides data security using trusted execution environments or encryption, providing protection of sensitive data across the Machine Learning Lifecycle. Simply put, it allows you to isolate your sensitive data while it's being processed in the cloud. The underlying confidential computing infrastructure protects this data from other applications, administrators, and the cloud provider with a hardware based trusted execution environment (TEE).

TEE is an important component that is used to provide strong assurances through hardware and software measurements from [trusted computing base (TCB)](https://en.wikipedia.org/wiki/Trusted_computing_base) components. Verifications of these measurements help with validation of the expected computation and verify any tampering of the container apps.

Resulting value proposition can be broken down into two elements:

1. Prevent unauthorized data access in three critical scenarios:
2. Hacker exploits a bug in the OS or the Hypervisor.
3. Malicious insider with administrator privileges.
4. Government asks for data access without customer’s knowledge.

Graphical user interface

Description automatically generated with low confidence

Figure : Unauthorized data access an app can be subject to.

1. Enable new businesses :

* Enable customers to move their sensitive workloads to the cloud.
* Enable new services/use cases (e.g., multi-party dataset analytics and confidential/private ML, both training and inference).

There are multiple options to get started using Azure Confidential Computing for your ML use-case.

* The first option is to [deploy an Azure confidential computing VM in the Azure portal](https://docs.microsoft.com/en-us/azure/confidential-computing/quick-create-portal) backed by [Intel® Software Guard Extensions (SGX)](https://software.intel.com/content/www/us/en/develop/videos/intel-software-guard-extensions-sgx.html?wapkw=SGX). This option is only of interest if you want to deploy a confidential compute virtual machine with custom configuration, which is not our case here.
* The second option is [to deploy your model to an AKS cluster with confidential computing nodes support](https://docs.microsoft.com/en-us/azure/confidential-computing/confidential-containers). This option is more in line with our previous pipelines, which is the reason we will give an overview of it below.

### A (very) first look at the confidential computing nodes on AKS

Here are some of the most important information to understand how confidential computing nodes on AKS work:

* Azure Kubernetes Service (AKS) supports adding [confidential computing nodes](https://docs.microsoft.com/en-us/azure/confidential-computing/confidential-computing-enclaves) powered by Intel SGX.
* These confidential computing nodes allow you to run sensitive workloads within a hardware-based trusted execution environment (TEE).
* TEEs allow user-level code from containers to allocate private regions of memory to execute the code with CPU directly. These private memory regions that execute directly with CPU are called enclaves.
* Enclaves help protect the data confidentiality, data integrity and code integrity from other processes running on the same nodes.

Below is an illustration of how an AKS confidential compute node works. As per the Intel SGX execution model, it removes the intermediate layers of Guest OS, Host OS and Hypervisor thus reducing the attack surface area.

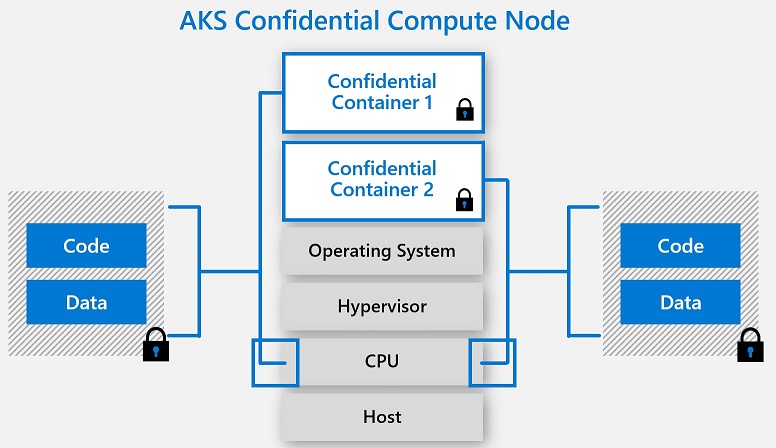


Figure . Illustration of how AKS confidential compute node works.

Instructions on how to deploy an AKS cluster with confidential computing nodes by using the Azure CLI can be found in the article “[Quickstart: Deploy an AKS cluster with confidential computing Intel SGX agent nodes by using the Azure CLI](https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-enclave-nodes-aks-get-started)”.

[Enclave aware containers](https://docs.microsoft.com/en-us/azure/confidential-computing/enclave-aware-containers), which are developed to run in enclaves, have two components:

1. An untrusted component (called the host)
2. And a trusted component (called the enclave).

As far as ONNX is concerned, the open-source enclave-based ONNX runtime can be containerized in such a way. It can then establish a secure channel between the client and the inference service - ensuring that neither the request nor the response can leave the secure enclave.

This approach and related solution allow you to bring existing ML model and run them confidentially while providing trust between the client and server through attestation and verifications.

These are beyond the scope of this illustration guide, but you are encouraged to experiment with Azure Confidential Computing on your own and get started with ML model lift and shift to ONNX runtime [here](https://aka.ms/confidentialinference).

.

**Note** To know more about confidential computing and extending hardware-enforced cryptographic protection to data while in use, you can refer to the this MSR article titled [Toward Confidential Cloud Computing](https://www.microsoft.com/en-us/research/uploads/prod/2021/03/Toward-Confidential-Cloud-Computing.pdf). Moreover, a thorough overview of Confidential computing on Azure can be found under the [Azure Confidential Computing](https://docs.microsoft.com/en-us/azure/confidential-computing/overview) documentation.

This concludes this module on the outer deployment loop of the ML workflow and the related Responsible AI activities.

More details on the various pipelines implemented to close this outer loop are available in the appendix so you are encouraged to look at it if you would like to implement these pipelines yourself.

# As a conclusion

This concludes this illustration guide, part of the Responsible AI Workshop. We hope you have enjoyed this guided tour on this end-to-end Responsible AI Lifecycle (RAIL) for MLOps processes to help you put Responsible AI to work.

As part of this guided tour, we have first introduced the Enterprise-grade Machine Learning paradigm for AI-powered cloud-native applications and Azure ML core MLOps capabilities, which i) we then used in the end-to-end walkthrough of the Responsible AI Lifecycle (RAIL) by incorporating related recommendations and activities in every stage of an ML workflow designed to solve a concrete loan decision problem, and ii) we incorporated into global DevOps processes for cloud-native applications.

This walkthrough provides a process you can replicate for a (more) responsible development of your own ML use case, but please keep in mind that this is just an example and is not meant to aggregate all RAI tools and practices.

# To go beyond

To continue learning about the passionate subject of Responsible AI, you can follow the other tutorials and walkthroughs available in this workshop.

For a more complete overview of the Responsible AI tooling and Microsoft’s journey, you can refer to the tooling tutorials in this workshop.

To continue learning about the passionate subject of Responsible AI, Une image contenant texte, motif, point

Description générée automatiquementyou can scan this code or visit <https://aka.ms/RAIresources> where you can access the entirety of already available tools, guidelines, and other additional resources that will help you create your next AI solution in a (more) responsible manner.

Une image contenant texte, capture d’écran, Site web, Page web

Description générée automatiquement

# Appendix – Azure DevOps pipelines step-by-step instructions

## *Build push* pipeline – Detailed instructions

You can create a Build Push pipeline by following the steps below:

1. Create a new *release* pipeline, and then add a GitHub artifact connecting to your repository exactly as you did in step one of the first pipeline. Here is how the artifact should look like.

Graphical user interface, text, application

Description automatically generated

Figure . Add GitHub artifact

1. Add an *AzureML* artifact, select the service connection you just created and for the name, give it the same name as the name of the model (here *catboost-predictor*).

Graphical user interface, text, application

Description automatically generated

Figure . Add an ML model artifact

Don’t forget to enable continuous deployment by clicking on the lightning bolt icon next to the artifact.

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure . Enable continuous deployment trigger

1. Now we can start defining the tasks of the pipeline. The first task is to use Python version 3.7 like we did in the first pipeline.

Graphical user interface, application

Description automatically generated

Figure . Choose Python version

1. Next, we install packages necessary for the good execution of the pipeline, namely the *azureml-core* package in our case.

Graphical user interface, text, application, email

Description automatically generated

Figure / Install required packages.

1. Now we create a docker\_context folder:

Graphical user interface, text, application, chat or text message, email

Description automatically generated

Figure . Create a folder

1. We now run the retrieve\_models.py script [available in the repo](https://github.com/alazraq/responsible-ai-lifecycle/blob/main/scripts/retrieve_model.py). It fetches the model from our Azure ML workspace.

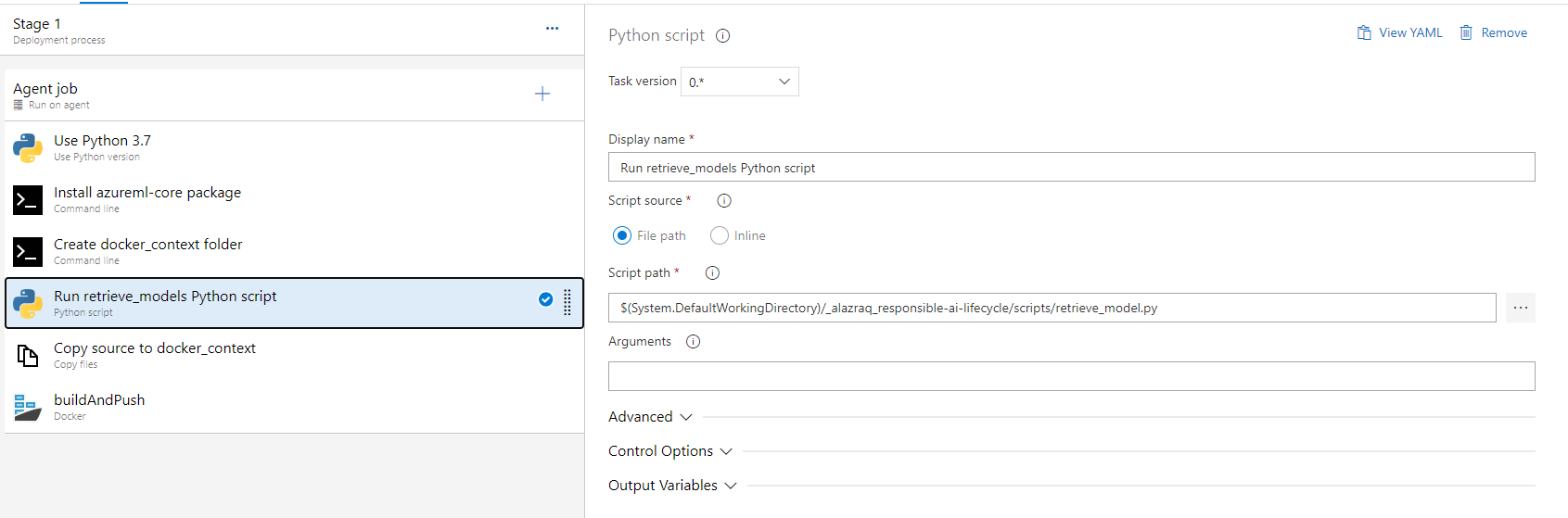


Figure . Run script to retrieve model from Azure ML

1. The next step is to copy the scoring script and the requirements [from the repo](https://github.com/alazraq/responsible-ai-lifecycle/tree/main/scoring_onnx) to the docker\_context folder.

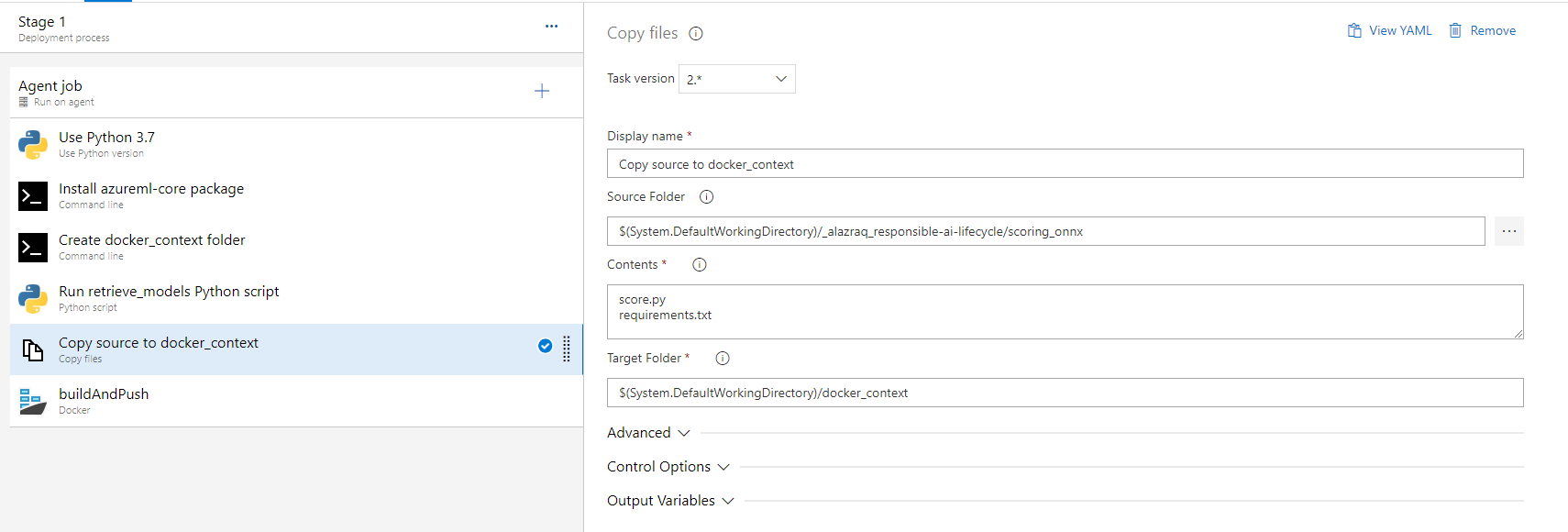


Figure . Copy files to docker context

1. The last and most important step of the pipeline is to build the docker image and push it to ACR.

Graphical user interface, text, application, email

Description automatically generated

Figure . Build image and push to ACR

The variables group defined in the Azure DevOps pipelines library looks as follows:

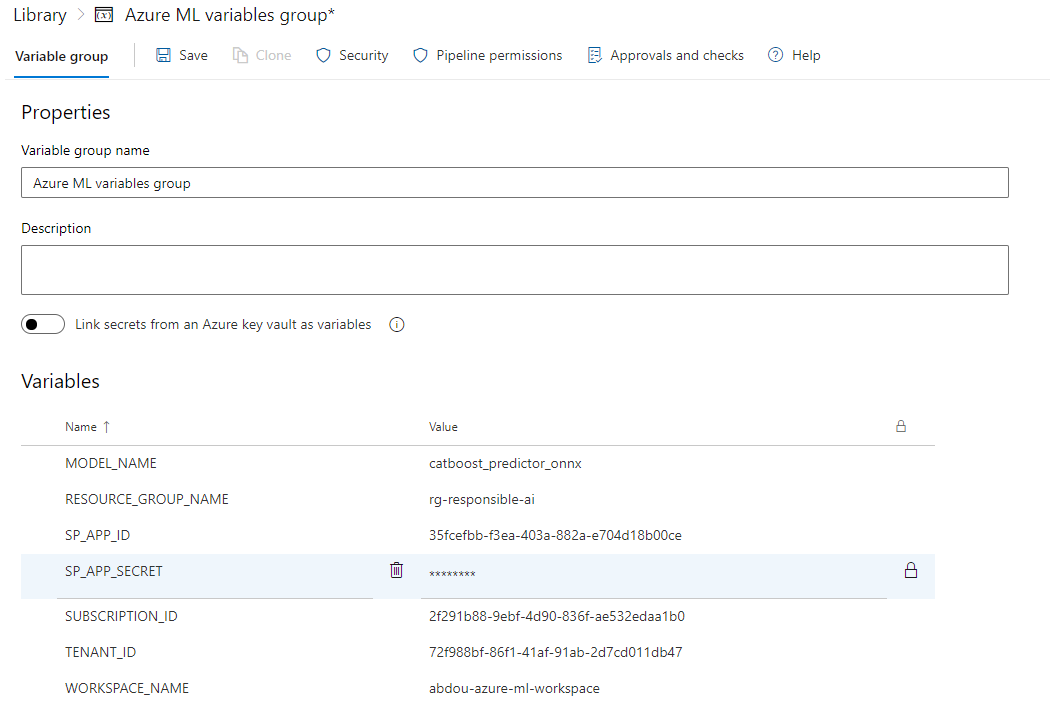


Figure . Variables group for build - push pipeline

Here is what these variables represent:

|  |  |
| --- | --- |
| Name | Suggested value |
| MODEL\_NAME | catboost\_predictor\_onnx |
| SP\_APP\_ID | [see below] |
| SP\_APP\_SECRET | [secret string generated] (**make it secret**) |
| SUBSCRIPTION\_ID | [your subscription id] |
| TENANT\_ID | [see below] |
| WORKSPACE\_NAME | [the name of your Azure ML workspace] |

To generate the App ID and secret, go to your Azure portal via <https://portal.azure.com> and go to your Azure AD/App registrations/Certificates & secrets and select New client secret.

Graphical user interface, text, application, email

Description automatically generated

Figure . Generate new client secret

This secret string acts as a password, it is the way for Azure DevOps to access your Azure account when requesting a service. Once created, save it somewhere because it will be hidden once you refresh the page.

To configure the connection, you need to get:

* Application (client) ID (SP\_APP\_ID above)
* Directory (tenant) ID (TENANT\_IDabove)

Graphical user interface, text, application, email

Description automatically generated

Figure . App ID and Tenant ID location

## *Release* pipeline – Detailed instructions

The Release pipeline has two artifacts and contains two stages with a single task each. Below are the steps to set up your release pipeline in Azure DevOps:

1. Create a new *Release* pipeline then add a GitHub artifact connecting to your repository exactly as you did in step one of the previous pipeline.

Graphical user interface, application

Description automatically generated

Figure : New empty release pipeline.

1. Add an ACR artifact as follows.

Graphical user interface, application

Description automatically generated

1. Add variable groups to the library as follows.

Graphical user interface, text, application, email

Description automatically generated

1. The pipeline itself contains a single task which executes the PowerShell script available [here](https://github.com/alazraq/responsible-ai-lifecycle/blob/main/scripts/deploy_on_aci.ps1).

Graphical user interface, text, application, email

Description automatically generated

Figure . Deploying on ACI using a PowerShell script

Deployment on AKS for the production stage is skipped here, but you will find detailed instruction on how to do so [here](https://docs.microsoft.com/en-us/azure/devops/pipelines/apps/cd/deploy-aks?view=azure-devops&tabs=java).

## Bonus: *Model Training* pipeline – Detailed instructions

The Model Training pipeline is automatically triggered by a commit being pushed to the git repository containing the train/test data and our data processing and modeling code and produces a new model artifact (i.e., a .onnx file in our case), which is registered to Azure ML model registry – triggering the Build Push pipeline which in turn triggers the Release pipeline.

This way, everything will be automated from pushing new data, data processing or modeling code to model deployment.

You can set up your modeling pipeline by following the steps below:

1. On the left-hand side of your Azure DevOps Dashboard, select Releases then New release pipeline. You can then name it “*Model Training*”.

Graphical user interface, application

Description automatically generated

Figure . New empty release pipeline

1. Select Add an artifact and then GitHub, then under service and source, you should be able to navigate to the GitHub repository you linked to your Azure DevOps account if you correctly followed the steps in the previous section. The default branch can be set to main and the default version to Latest from the selected branch.

Graphical user interface, application, email

Description automatically generated

Figure . Adding a GitHub repo as an artifact to the pipeline.

Don’t forget to activate the Continuous deployment trigger by clicking on the lightning icon next to the artifact.

Graphical user interface, text, application, email

Description automatically generated

Figure . Activating the Continuous deployment trigger.

1. Now we can focus on the tasks of the pipeline itself. Here is an overview of how our model training pipeline will look like.

Graphical user interface, text

Description automatically generated

Figure . Overview of the model training pipeline

There is only one stage with an Agent job composed of two tasks.

Graphical user interface, application, Teams

Description automatically generated

Figure . Tasks of the model training pipeline

The first task tells Azure DevOps to use python 3.7.

Graphical user interface, text, application, email

Description automatically generated

Figure . First task of the modeling pipeline specifying the Python version to use.

The second task simply launches the training script available in the repository.

Graphical user interface, text

Description automatically generated with medium confidence

Figure . Second task of the modeling pipeline executing the training python script.

Upon pushing a commit to the linked GitHub repository, the *modeling* pipeline is automatically launched, which executes the training scripts. This script includes all the code for data processing and modeling, it outputs a model artifact which it registers directly to the Azure ML.

This in turn triggers the launch of the Build Push pipeline investigated above and the rest of the model deployment pipelines are triggered one after the other automatically. This completes our continuous integration and continuous delivery/deployment (CI/CD) efforts, it closes and connect the outer and the inner feedback loops.

You have reached the end of the Appendix and of this illustration guide, part of the Responsible AI Workshop. Thank you for taking the time to read through the entire document and hope you have enjoyed the walkthrough!

Une image contenant bleu, brouillard, capture d’écran, bleu vert

Description générée automatiquement

1. <https://en.wikipedia.org/wiki/Cloud_native_computing> [↑](#footnote-ref-2)
2. CNCF Cloud Native Definition v1.0: <https://github.com/cncf/toc/blob/main/DEFINITION.md> [↑](#footnote-ref-3)
3. What is DevOps? with Donovan Brown: <https://devblogs.microsoft.com/devops/what-is-devops-donovan/> [↑](#footnote-ref-4)