| Component/Concern | Requirements | Extracts | Sources |
|---|---|---|---|
| Pipeline Coordinator | Orchestration-style coordination of pipelines should be supported | An orchestration tool to orchestrate and execute your automated ML workflow steps.<br><br>Orchestrate the workflow and interaction between the data warehouse, data pipeline, and features written out to a storage or processing pipeline | S2, S7, S10, S52, S53 |
| | Choreography-style coordination of pipelines should be supported | message broker (serves as the middle-man to help coordinate processes between the data job and the training job). | S2, S7, S10 |
| ML Experimentation/ ML Project | Create and manage ML workspaces | Azure Machine Learning is a cloud service for training, scoring, deploying, and managing machine learning models at scale. This architecture uses the Azure Machine Learning Python SDK to create a workspace, compute resources, the machine learning pipeline, and the scoring image. An Azure Machine Learning workspace provides the space in which to experiment, train, and deploy machine learning models.<br><br>packaging format (for projects) to reproduce models<br><br>1. MLflow projects –MLflow projects provide a convenient way to package machine learning code. Each project is a simple repository with multiple files and properties to run the project, such as Project Name, Entry – Points and the environment, library dependencies, etc. | S3, S16, S29, S38, S53, S56 |
| | Create, publish, discover, use, customize, and pipelines | A training pipeline will be needed because we'd need to automate the retraining process as fresh articles come in every day.<br><br>Enabling fast development life cycles by offering out-of-the-box (correlated) sampling of relevant data sets<br><br>Models are different than code for reuse, since data scientists must tune models based on input data and scenario. To reuse a model for a new scenario, you may need to fine-tune/transfer/learn on it. You need the training pipeline.<br><br>1. Development and experimentation: You iteratively try out new ML algorithms and new modeling where the experiment steps are orchestrated. The output of this stage is the source code of the ML pipeline steps that are | S3, S24, S5, S17, S29, S28, S32, S33, S34, S35, S42, S44, S47, S49, S53, S56 |

| | | | |
|---|---|---|---|
| | | then pushed to a source repository.<br>2. Pipeline continuous integration: You build source code and run various tests. The outputs of this stage are pipeline components (packages, executables, and artifacts) to be deployed in a later stage.<br>3. Pipeline continuous delivery: You deploy the artifacts produced by the CI stage to the target environment. The output of this stage is a deployed pipeline with the new implementation of the model.<br>"Initially training jobs can be run on the data scientist's local machine. But as the size of the dataset or processing grows then a tool will be needed that can leverage specialised cloud hardware, parallelize steps and allow long-running jobs to run unattended."<br><br>"Steps can be broken out as reusable operations and run in parallel. This helps address needs for steps to split the data into segments and apply cleaning and pre-processing on the data. The UI allows for inspecting the progress of steps. Runs can be given different parameters and executed in parallel. This allows data scientists to experiment with different parameters and see which result in a better model."<br><br>Create reproducible machine learning pipelines. Use machine learning pipelines to define repeatable and reusable steps for your data preparation, training, and scoring processes. | |
| | Experiment with Jupyter Notebooks or pipelines | | S3, S22,S34, S38, S56 |
| | Create and publish scoring Image | The scoring Python script is packaged as a Docker image and versioned in the registry. | S3 |
| | Export experimental code from Jupyter notebooks into deployable pipelines | Another fundamental change was that the data scientists needed the capability to export experimental code from Jupyter notebooks into the MLOps deployment process rather than trigger training and scoring directly.<br><br>That isn't the only possible method. Databricks supports scheduling notebooks as jobs. But, based on current client experience, this approach is difficult to instrument with full DevOps practices because of testing limitations.<br><br>● Experimental-operational symmetry: The pipeline implementation that is used in the development or experiment environment is used in | S4, S17, S22, S49 |

| | | | |
|---|---|---|---|
| | | the preproduction and production environment, which is a key aspect of MLOps practice for unifying DevOps.<br>Moving your experimentation process to a clearly defined steps of the pipeline (starting from data ingestion, preparation and ending on model training) will enable the development of reusable components that can significantly speed up and standardize the whole experimentation process. | |
| | Record and query experiments | Reproducibility: What metadata about ML experiments is collected? (data sets, hyperparameters)<br>1. MLflow Tracking – ML Flow tracking component is organized around the concept of code execution. Each record of execution contains information, like code version, start and end time, code source, input parameters, performance metrics, and output file artifacts.<br>2. Execute and track experiments. In ML, failure is not really a failure; it's the making of a better model in progress. You're usually attempting to solve a problem, considering a hypothesis that's based on business objectives, available datasets, and ML algorithms that are suitable for the use case. Each experiment drives you closer to the best possible solution. It's crucial to track your experiments because it helps maintain continuity and helps the rest of your team understand what was tried and what went wrong.<br>Running hundreds or even thousands of iterations on the way to an optimally trained ML model is not uncommon. In the process, quite a few parameters used to define each experiment and the results of that experiment are accumulated. Often, this metadata is stored in Excel spreadsheets or, in the worst case, in the heads of team members. However, to establish optimal reproducibility, avoid time-consuming multiple experiments, and enable optimal collaboration, this data should be captured automatically. Possible tools here are MLflow tracking [14] or Sacred [15]. To visualize the output metrics, either classical dashboards like Grafana [16] or specialized tools like TensorBoard [17] can be used. TensorBoard can also be used for this purpose | S7, S9, S16, S22, S23, S25, S30, S33, S38, S53 |

| | | | |
|---|---|---|---|
| | | independently of its use with TensorFlow. For example, PyTorch provides a compatible logging library [18]. However, there is still much room for optimization and experimentation here. For example, the combination of other tools from the DevOps environment such as Jenkins [19] and Terraform [20] would also be conceivable. | |
| | Automated deployment and execution of pipelines | Offer endpoint to span an instance<br><br>● Pipeline deployment: In level 0, you deploy a trained model as a prediction service to production. For level 1, you deploy a whole training pipeline, which automatically and recurrently runs to serve the trained model as the prediction service.<br><br>● Automated deployment to a test environment, for example, a deployment that is triggered by pushing code to the development branch.<br>● Semi-automated deployment to a pre-production environment, for example, a deployment that is triggered by merging code to the main branch after reviewers approve the changes.<br>● Manual deployment to a production environment after several successful runs of the pipeline on the pre-production environment. | SX, S17, S23, S33 |
| Continuous integration and continuous delivery (CI/CD) | Automation (CI/CD) for managing ML assets | Treat ML assets (training/testing/validation data sets, training scripts, models, experiments, and service wrappers) as first class citizens in DevOps processes<br><br>Integration of ML assets with existing CI/CD solutions<br><br>Extending Cloud-native build tools to support allocation of ML assets, data and hardware during training builds<br><br>Use a CI/CD framework to build, test, and deploy software. It offers the benefits of reproducibility, security, and code version control<br><br>Use CI/CD tools such as repos and orchestrators (borrowing devops principles) to automate the pre-production pipeline.<br><br>What are the non-functional requirements for the ML model (efficiency, fairness, robustness, interpretability, etc.)? How are | S1-S5, S7, S8, S9, S22, S25, S27, S29, S31, S32, S35, S45, S55, S57, S56 |

| | | | |
|---|---|---|---|
| | | they tested? Are these tests integrated into the CI/CT workflow?<br><br>Data are often reflections of real-life processes and, hence, errors and misrepresentations. Many AI projects therefore include explicit activities where data are cleaned, scrubbed, massaged, pre-processed, and transformed. The best practice here is to automate such data preparation steps, as suggested by many sources<br><br>Putting AI models into production is not much different than putting regular software into production, including the desire to automate the process. Automation prevents error, repetition of tedious effort, and increases speed, at least in theory | |
| | Track DevOps Metrics | Do we track deployment frequency, lead time for changes, mean time to restore, and change failure rate metrics? | S8, S9, S24 |
| | Continuous training (CT) | Generally, continuous integration denotes the building, testing, and packaging of data and model pipelines. Continuous training is a new property, unique to ML systems concerned with automatically retraining ML models.<br><br>● Frequently retrain your production models: To capture the evolving and emerging patterns, you need to retrain your model with the most recent data. For example, if your app recommends fashion products using ML, its recommendations should adapt to the latest trends and products.<br>Training never ends. Once the drop in performance is spotted, the model must be retrained with the fresh data and validated before rolling out into production again. Thus, in MLOps, continuous training and validation replace continuous testing, performed in DevOps. | Sx, S16, S31, S53 |
| Testing | Testing ML assets appropriately<br>Runs validation tests for code, data, and models (ML assets) | Run for new code<br><br>These tests verify that the data samples conform to the expected schema and distribution. Customize this test for other use cases and run it as a separate data sanity pipeline that gets triggered as new data arrives. For example, move the data test task to a data ingestion pipeline so you can test it earlier.<br><br>Run for new code<br>Unit test. These tests make sure the code works, has adequate code coverage and is stable. | S3, S4, S5, S24, S9, S25, S33, S39, S45, S47 |

| | | | |
|---|---|---|---|
| | | These tests ensure that the code conforms to the standards of the team.<br><br>Code validation tests for machine learning focus on validating the quality of the code base. It's the same concept as any engineering project that has code quality tests (linting), unit tests, and code coverage measurements.<br><br>Model validation typically refers to validating the full end-to-end process steps required to produce a valid machine learning model. It includes steps like:<br>● Data validation: Ensures that the input data is valid.<br>● Training validation: Ensures that the model can be successfully trained.<br>● Scoring validation: Ensures that the team can successfully use the trained model for scoring with the input data.<br>Algorithm validation - track classification / regression metrics based on business problem as well as ensure algorithm fairness.<br><br>- Conventional Unit / Integration / BDD / UAT testing<br>- Adversarial testing<br>- Bias detection<br>- Fairness testing<br>- Ethics testing<br>- Interpretability<br>- Stress testing<br>- Security testing<br><br>Data preprocessing test<br>Test for training-serving skew<br>Data preprocessing - check that data is preprocessed efficiently and in a scalable manner and avoid any training-serving skew<br><br>Is pre-assertion for input data implemented?<br>What fallback method for an inadequate model output (post-assertion) is implemented? (Do we have a heuristic benchmark?)<br>Learning models are meant to solve a task, such as classification or prediction. Given this task, it's often clear that performance of the models can be tested against the known data sets. Preferably, the testing data also originates from a different source than the training dat | |
| | Test prioritization and scheduling | Ideally, have your build pipeline finish quickly and execute only unit tests and a subset of other tests. This allows you to validate the changes quickly and fix them if issues arise. Run long-running tests during off-hours. | S3, S4, S33 |

| | | | |
|---|---|---|---|
| | | Running this full set of steps on the machine learning environment is expensive and time consuming. As a result, the team did basic model validation tests locally on a development machine. It ran the steps above and used the following:<br>● Local testing dataset: A small dataset, often one that's obfuscated, that's checked in to the repository and consumed as the input data source.<br>● Local flag: A flag or argument in the model's code that indicates that the code intends the dataset to run locally. The flag tells the code to bypass any call to the machine learning environment.<br>The goal of these validation tests isn't to evaluate the performance of the trained model. Rather, it's to validate that the code for the end-to-end process is of good quality. It assures the quality of the code that's pushed upstream, like the incorporation of model validation tests in the PR and CI build. It also makes it possible for engineers and data scientists to put breakpoints into the code for debugging purposes. | |
| | Unit and integration tests throughout your MLOps architecture | You are now ready to bring full continuous integration/continuous deployment (CI/CD) capabilities to your ML pipeline, automating your modularized model training and serving. A fundamental part of continuous integration is to flesh out unit and integration tests throughout your MLOps architecture. Some of these tests are unique to MLOps, such as those concerning model governance, triggers to train or deploy a new model, training on small datasets to test convergence, or performing model validation on a set of specific slices of data to evaluate model fairness or consistency<br>across customer bases.<br><br>● Validating component integration: we can use a similar approach to testing the integration between different services, using Contract Tests to validate that the expected model interface is compatible with the consuming application. Another type of testing that is relevant when your model is productionized in a different format, is to make sure that the exported model still produces the same results. This can be achieved by running the | S25, S33, S53 |

| | | | |
|---|---|---|---|
| | | original and the productionized models against the same validation dataset, and comparing the results are the same. | |
| Model Training | Model checkpoints<br>Parallelise training | Model checkpoints so that training and retraining don't take too long (preferably under a few hours from the time they come in as that is often the peak log-in time for our users).<br><br>There are a few options for parallelisation:<br><br>• The simplest form is to have a dedicated pipeline for each model, i.e. all models run concurrently.<br>• Another idea is to parallelise the training data i.e. the data is partitioned and each partition has a replica of the model. This is preferred for those models that they need all fields of an instance to perform the computation (e.g. LDA, MF).<br>• A third option is to parallelise the model itself i.e. the model is partitioned and each partition is responsible for the updates of a portion of parameters. It is ideal for Linear models, such as LR, SVM.<br>• Finally, a hybrid approach can be used, combining one or more options. (For more info I recommend you read this publication).<br><br>Model training must be implemented with error tolerance in mind and also data checkpoints and failover on training partitions should be enabled — e.g. each partition can be retrained if the previous attempt fails due to some transient issue (e.g. timeout). | S2, S4, S3, S24, S36 |
| | Distributed model training | Is distributed model training required? Do we have an infrastructure for distributed training?<br><br>--> 'Train' - Computation load distribution frameworks for '<br><br>- Distributed Training (e.g., using Horovod or Sonner or custom template)<br>- Standalone training<br>- with support for real time Training Visualization(e.g., TensorBoard/VisualDL)<br><br>Smaller ML models can usually still be trained on one's own laptop with reasonable effort. However, as soon as the models become larger and more complex, this is no longer possible and a central on-premise or cloud server becomes necessary for training. For automated training in such an environment, it makes sense | S8, S9, S10, S24, S23, S38, S53 |

| | | | |
|---|---|---|---|
| | | to build a model training pipeline.<br>This is executed with training data at specific times or on demand. The pipeline receives the configuration data that defines a training cycle. These data are for example model type, hyperparameters and used features. The pipeline can obtain the training data set automatically from the feature store and distribute it to all model variants to be trained in parallel. Once training is complete, the model files, original configuration, learned parameters, and metadata and timings are captured in the experiment and model tracking tools. One possible tool for building one is Kubeflow [23]. Kubeflow provides a number of useful features for automated training and for (cost-)efficient resource management based on Kubernetes. | |
| | Train and execute models on specific heterogeneous hardware | - on accelerated neural network hardware NVIDIA GPU/Google TPU/CPU with the leverage of NVIDIA GPU-accelerated libraries (CUDA,<br>We make the following assertions:<br>● Models may expect to be trained on various combinations CPU, GPU, TPU, dedicated ASICs or custom neuro-morphic silicon, with new hardware entering the market regularly.<br>● Models may expect to be executed on various combinations CPU, GPU, TPU, dedicated ASICs or custom neuro-morphic silicon, again with new hardware entering the market regularly.<br>● It must not be assumed that models will be executed on the same hardware as they are trained upon.<br>It is desirable to have the ability to train once but run anywhere, that is, models are trained on specific hardware to minimise training time, and optimised for different target devices based on cost or performance considerations, however this aspiration may need to be tempered with a degree of pragmatism. The ability to train on a given hardware platform requires, at minimum, dedicated driver support, and in practice usually also necessitates extensive library / framework support. The ability to execute on a given hardware platform may involve having to significantly optimise resource usage to lower product cost or require the production of | |

| | | | |
|---|---|---|---|
| | | dedicated silicon specific to a single task. | |
| | Prioritizing training activities | Training activities are time-consuming. In scenarios where systems support the simultaneous queuing and/or processing of multiple training activities, it must be possible to prioritise those activities to prevent long-running trainings from blocking the deployment of other processes such as security or bug fixes. | S24 |
| | Making Trade-off of training time, data set size, etc. | We have a limited budget, our model will need to train every day at a reasonable amount of time and we may have to cap the amount of data it trains on for each training run as we can't afford to take long to train and on very high volumes of new article releases.<br><br>cap the amount of data it trains | S3 |
| | Use automated machine learning tools | Use popular open source libraries such as scikit-learn and hyperopt to train and improve model performance. As a simpler alternative, use automated machine learning tools such as AutoML to automatically perform trial runs and create reviewable and deployable code.<br><br>```<br>'HPO' - Hyper Parameter Optimization,<br>'Randmoized/Grid Serach',<br>'NAS' - Neural Architecture Search,<br>'AutoML'<br>```<br>The most radical variation in terms of target personas comes with whether and how MLOps platforms incorporate automated machine learning (AutoML). When AutoML is the core focus of the platform, the need for deep data science knowledge and operations knowledge are both reduced; instead, the key role is a 'citizen data scientist' or even an advanced business user.<br><br>Segemaker - It accelerates the deployment process by providing Autopilot that can select the best algorithm for the prediction, and can automatically build, train, and tune models. | S7, S10, S11, S16, S29 |
| | keep track of the training metadata for each run | Since we'll frequently be retraining our model, we need to keep track of the training metadata for each run so that when a model fails to perform, we can audit its training details and troubleshoot.<br><br>Azure Machine Learning provides an easy way to log at each step of the machine learning life cycle. The logs are stored in a blob container. | S2, S3, S16 |

| | | | |
|---|---|---|---|
| | | Neptune focuses on logging and storing of ML Metadata, which makes it easier to query the data to analyze later. | |
| | Online training | • Online learning models: unlike the models we discussed so far, that are trained offline and used online to serve predictions, online learning models use algorithms and techniques that can continuously improve its performance with the arrival of new data. They are constantly learning in production. This poses extra complexities, as versioning the model as a static artifact won't yield the same results if it is not fed the same data. You will need to version not only the training data, but also the production data that will impact the model's performance. | S33, S36, S35, S51 |
| ML Assets | Track ML Assert Lineage (lifecycle logs) | feature engineering has to be reproducible?<br>Do we have a data governance process such that feature engineering has to be reproducible?<br><br>ML algorithm and model development is iterative and experimental. It requires a lot of parameter tuning and feature engineering. ML pipelines work with data versions, algorithm code versions and/or hyper-parameters. Any change in these artifacts (independently) trigger new deployable model versions that warrant experimentation and metrics calculations. MLOps platforms track the complete lineage for these artifacts.<br><br>ML workflow steps auditability, visibility, and reproducibility implemented using Amazon SageMaker Lineage Tracking.<br><br>For decades, version control has been the modus operandi in traditional software engineering. In contrast, version control is often poorly executed in machine learning projects. For starters, data science teams often only push their final model iterations. Consequently, different models and iterations are lost, with recovery impossible. To make matters worse, training a model comprises more than just code. One training iteration alone is defined by code, data, and hyperparameters. Often, poor versioning results in lost value. As a result, Machine Learning engineers are left wondering how they achieved the performance they did and have | S9, S5, S20, S33, S34, S35, S45 |

| | | | |
|---|---|---|---|
| | | no way to re-train a model that performs equally as well.<br><br>Besides reproducibility, proper versioning also helps companies comply with tightening regulatory requirements, e.g., GDPR in the EU or the Algorithmic Accountability Bill in New York City. Regulations require model predictions to be explainable. The first step in explaining why a model performs a certain prediction is knowing how the model was trained. Having data enables you to explain why a model might return biased results.<br><br>An operation MLOps infrastructure can reproduce models through proper data lineage and governance by version code, data, and hyperparameters. To do so, it helps engineers automatically track training iterations through metadata and code flags.<br><br>In addition to code versioning, data versioning is useful in a machine learning context. This allows us to increase the reproducibility of our experiments and to validate our models and their predictions by retracing the exact state of a training dataset that was used at a given time. Tools such as DVC [12] or Pachyderm [13] can be used for this purpose.<br><br>While this allows us to retrain our model when the data changes, it doesn't tell the entire story about data versioning. One aspect is data history: ideally you would want to keep an entire history of all data changes, but that's not always feasible depending on how frequently the data changes. Another aspect is data provenance: knowing what processing step caused the data to change, and how that propagates across different data sets. There is also a question around tracking and evolving the data schema over time, and whether those changes are backwards and forwards compatible.<br><br>"If something goes wrong with running software then we need to be able to recreate the circumstances of the failure. With mainstream applications this means tracking which code version was running (docker image), which code commit produced it and the state of the system at the time. That enables a developer to recreate that execution path in the source code. This is reproducibility.<br><br>Achieving reproducibility for machine learning involves much more. It involves knowing what | |

| | | | |
|---|---|---|---|
| | | data was sent in (full request logging), which version of the model was running (likely a python pickle), what source code was used to build it, what parameters were set on the training run and what data was used for training. The data part can be particularly challenging as this means retaining the data from every training run that goes to live and in a form that can be used to recreate models. So any transformations on the training data would need to be tracked and reproducible.<br><br>The tool scene for tracking across the ML lifecycle is currently dynamic. "<br><br>Data constitute the starting point of any AI project, whatever algorithm they apply. Many sources state that it's essential to keep track of the origin of data sets for several purposes: compliance, recovery, and good old engineering practice. A data set needs to be provided with metadata that clarify provenance, i.e., where, how, when, and by whom the data was collected. Also, if there have been any changes to the data since collection, those need to be recorded as well. | |
| | ML Asset Versioning and Lineage | hyper-parameters versioning<br>In Machine Learning, output model can change if algorithm code or hyper-parameters or data change. While code and hyper-parameters are controlled by developers, change in data may not be. This warrants the concept of data and hyper-parameters versioning in addition to algorithm code. Note that data versioning is a challenge for unstructured data such as images and audio and that MLOps platforms adopt unique approaches to this challenge.<br><br>Data versioning ()<br><br>There must be versioning of datasets as the schema and origin data changes.<br><br>Since we consider ML models and data as "first-class citizens," data versioning might need to be implemented to analyze the model performance every time new data is available.<br><br>In addition, data versioning might be a regulatory requirement to explain predictions for every version of the learned model. Generally, we distinguish three levels of data versioning as described in "Machine Learning Engineering" by A.Burkov: 1) Data is versioned as a snapshot at training time; 2) Versioning data | S2, S4, S5, S7, S9, S20, S22, S23, S25, S31, S33, S34, S35, S41, S45, S57 |

| | | | |
|---|---|---|---|
| | | and code as one asset; 3) Using specialized data versioning systems.<br><br>Track all changes when dealing with code and client data.<br>A version control system to store, track, and version changes to your ML code.<br><br>● Use ML versioning and source control. By allowing team members to work alongside each other, versioning of ML code makes collaboration much easier. Traditional code versioning can keep track of code, configurations, and the project dependencies. In ML, however, things get complex, so versioning the code that implements a model is not enough. The model might behave drastically different from one input dataset to another. And to capture the complete training state, you also need to perform versioning on training data and generated models. By recording metadata about experiments, like run-time parameter passed at execution time, components executed, and model evaluation metrics, you facilitate reproducibility and comparison across multiple experiments.<br>In addition to code versioning, data versioning is useful in a machine learning context. This allows us to increase the reproducibility of our experiments and to validate our models and their predictions by retracing the exact state of a training dataset that was used at a given time. Tools such as DVC [12] or Pachyderm [13] can be used for this purpose.<br><br>In addition to the results of our experiments, the trained models themselves can also be captured and versioned. This allows us to more easily roll back to a previous model in a production environment. Models can be versioned in several ways: In the simplest variant, we export our trained model in serialized form, for example as a Python .pkl file. We then record this in a suitable version control system (Git, DVC), depending on its size.<br>Another option is to provide a central model registry. For example, the MLflow Model Registry [21] or the model registry of one of the major cloud providers can be used here. Also, the model can be packaged in a Docker container | |

| | | | |
|---|---|---|---|
| | | and managed in a private Docker Registry [22].<br>Besides code versioning, you need a place to save data and model versions. Machine learning involves a lot of experimenting. Data scientists train models with various datasets, which leads to different outputs. So, in addition to code version control utilized in DevOps, MLOps requires specific instruments for saving data and model versions to be reused and retrained.<br><br>Capture the governance data for the end-to-end machine learning lifecycle. The logged lineage information can include who is publishing models and why changes were made. It can also include when models were deployed or used in production.<br><br>There will always be plenty of reasons why a team will want to reflect back to the code that created the model they are currently working with, in the case of issues or requests for improvements of performance. A key practice is therefore to record model configuration code in central version control and ensure that proper tagging is done for model versions that proceed to training and further. | |
| | ML Asset Metadata Management | Metadata management is the management of information about each execution of the experiments, data and model pipeline is recorded to provide data and artifacts lineage, reproducibility, and debug errors.<br><br>The Metadata Store is a cross-cutting component that spans all previous elements of the MLOps infrastructure stack. Depending on your organization and regulatory requirements, you might need to implement an ML model governance process. This process will mainly rely on ML metadata. Therefore, the requirement for ML governance is the ML metadata store component.<br><br>What kind of metadata in code, data, and model management need to be collected? (e.g., the pipeline run ID, trigger, performed steps, start/end timestamps, train/test dataset split, hyperparameters, model object, various statistics/profiling, etc.)<br><br>What operational metrics need to be collected? E.g., time to restore, change fail percentage.<br><br>At this stage, you should already be tracking business metrics for model validation and monitoring, but now it’s time to track everything. This starts with all the metadata from model | S9, S14, S24, S16, S17, S35, S49 |

| | | | |
|---|---|---|---|
| | | training runs (in all environments) and model serving: code versions, dataset metadata, and training hyperparameters, to name a few.<br><br>Tracking and logging outputs from each module in the ML pipeline unlocks the ability to resume a failed pipeline run from the most recent successful step, instead of having to re-execute from the beginning.<br><br>New meta-data techniques must be created to effectively record and manage aggregates of data that represent specific versions of training, testing or validation sets. Given the variety of data storage mechanisms in common usage, this will likely necessitate pluggable extensions to tooling.<br><br>Neptune has categorized ML metadata into three different areas:<br>1. Experiment and model training data – This allows users to log different metrics, hyperparameters, learning curves, predictions, diagnostic charts etc.<br>2. Artifact metadata – This contains information about data such as path to dataset, features details, size, last updated timestamp, dataset preview etc.<br>3. Model metadata – Model metadata contains information such as who created or trained the model, links to training and experiments done as part of modelling, Multiple datasets details etc.<br>Information about each execution of the ML pipeline is recorded in order to help with data and artifacts lineage, reproducibility, and comparisons. It also helps you debug errors and anomalies. Each time you execute the pipeline, the ML metadata store records the following metadata:<br>● The pipeline and component versions that were executed.<br>● The start and end date, time, and how long the pipeline took to complete each of the steps.<br>● The executor of the pipeline.<br>● The parameter arguments that were passed to the pipeline.<br>● The pointers to the artifacts produced by each step of the pipeline, such as the location of prepared data, validation anomalies, computed statistics, and extracted | |

| | | | |
|---|---|---|---|
| | | vocabulary from the categorical features. Tracking these intermediate outputs helps you resume the pipeline from the most recent step if the pipeline stopped due to a failed step, without having to re-execute the steps that have already completed.<br>● A pointer to the previous trained model if you need to roll back to a previous model version or if you need to produce evaluation metrics for a previous model version when the pipeline is given new test data during the model validation step.<br>● The model evaluation metrics produced during the model evaluation step for both the training and the testing sets. These metrics help you compare the performance of a newly trained model to the recorded performance of the previous model during the model validation step.<br>Standardize metadata management with clearly defined locations and types of captured data.<br>Match metadata with source code that generated it. | |
| | ML Asset Storage and Marketplace<br>(Model Registry, Feature Store, etc.) | Model registration<br><br>Model Registry<br>a model registry to store details of the experiments and metadata from our training runs. This will also enable lineage traceability of the model being deployed and that of the data as well.<br>This service provides version control for the models along with metadata tags so they can be easily reproduced.<br><br>A model registry for versioning and tracking the trained model artifacts, implemented using Amazon SageMaker Model Registry.<br><br>The common reason for the machine learning model update is the “model decay,” where the model performance declines with time as new data arrives. All ML models should be versioned and protocolled in regulated industries such as health, finance, or military. At the same time, there might be a need to ensure backward compatibility by rolling back previously built models<br><br>Feature Store | S3,S2, S9, S1, S16, S17, S18, S19, S21, S23, S35, S35, S50, S51, S53, S56 |

| | | | |
|---|---|---|---|
| | | Connection to training via data pipeline<br><br>Connection to serving via data pipeline<br><br>The motivation for the “Feature Store,” as a new component in the MLOps stack, is the need for management, reproducibility, discovery, and reuse of features across ML projects and various data science teams. A feature store is defined as an interface between data engineering and ML model engineering to separate the feature engineering from the CRISP-ML(Q) model development process. Feature stores promise the speed up in the development and operationalization of ML models. However, as an advanced component, feature stores might add complexity and its implementation need to be critically considered for every ML project:<br><br>What databases are involved in feature storage?<br><br>Feature stores capture features that have been engineered from the raw data with cataloging for reuse. The feature store therefore offers a more processed form of the data in the data platform, similar to the role of a data mart alongside a data lake (though a mart typically serves reporting rather than ML)<br><br>1. MLflow registry – MLflow Model Registry is a centralized place to collaboratively manage the lifecycle of an MLflow Model. It has a set of APIs and UI to register a model, monitor the versioning and stage transition. Developers can easily annotate these models by providing descriptions and any relevant information that can be useful for the team.<br>The resulting models are stored in a versioned model repository along with metadata, performance metrics, required parameters, statistical information, etc. Models can be loaded later into batch or real-time serving micro-services or functions.<br><br>Many machine learning models and systems within a company use the same or at least similar features. Now, once we have extracted a feature from our raw data, there is a high probability that this feature can be useful for other applications as well. Therefore, it can be useful not to have to implement feature extraction again for each | |

| | | | |
|---|---|---|---|
| | | application. For this, we can store known features in a feature store. This can be done either in a dedicated component (such as Feast) [11], or in well-documented database tables populated by appropriate transformations. These transformations can be mapped automatically using Apache Airflow.<br>Register, package, and deploy models from anywhere. You can also track associated metadata required to use the model. | |
| | ML Asset marketplace / hub | 'AI Collaboration, Sharing & Exchange'<br>Authenticated, Authroized and Logging based Market place for sharing & exchange of MODELS and Model related artefacts<br>- AI/ML APIs,<br>- Fature Set,<br>- Training metrics,<br>- Model Binaries,<br>- Notebook /Model Code, | S10 |
| Managing and tracking trade-offs | Managing and tracking trade-offs | What is the delivery format for the model?<br><br>'Model Serialization & Quantization' (e.g.,Using TF-Serving, ONNX Runtime, Seldon Core et al)<br>- Serializaion support for Protobuff, hdf5, pickle et al<br>- Serialization interoperability<br>- Model binary/servable quantization & Pruning<br>Stored models are currently often little more than serialised objects. To decouple training languages, platforms and hardware from operational languages, platforms and hardware it is necessary to have broadly supported standard intermediate storage formats for models that can be used reliably to decouple training and operational phases.<br><br>Model Pruning and Quantization<br>Both pruning and quantization are model compression techniques that make the model physically smaller to save disk space and make the model require less memory during computation to run faster.<br><br>We make the following assertions:<br>● Models may expect to be trained on various combinations CPU, GPU, TPU, dedicated ASICs or custom neuro-morphic silicon, with new hardware entering the market regularly.<br>● Models may expect to be executed on various combinations CPU, GPU, TPU, dedicated ASICs or custom neuro-morphic | S7, S9, S10, S24, S16, S35, S38, S40, S57 |

| | | | |
|---|---|---|---|
| | | silicon, again with new hardware entering the market regularly.<br>● It must not be assumed that models will be executed on the same hardware as they are trained upon.<br><br>1. MLflow models – Mlflow format defines a standard format that lets you save a model in different flavors, such as python-function, PyTorch, sklearn, and it can be used by different platforms without much trouble at all.<br>Converting your model to Open Neural Network Exchange (ONNX) might improve performance. On average, converting to ONNX can double performance.<br><br>There are two significant problems when it comes to hardware and mobile deployment:<br><br>Embedded and mobile devices have low-processor with little memory, which makes the process slow and expensive to compute. Often, we can try some tricks such as reducing network size, quantizing the weights, and distilling knowledge.<br>Embedded and mobile PyTorch/TensorFlow frameworks are less fully featured than the full PyTorch/TensorFlow frameworks. Therefore, we have to be careful with the model architecture. An alternative option is using the interchange format. | |
| | Managing and tracking trade-offs | Making sure that you test your model for deployment, including infrastructure compatibility and consistency with the prediction service API.<br><br>Verifying the compatibility of the model with the target infrastructure before you deploy your model. For example, you need to verify that the packages that are required by the model are installed in the serving environment, and that the memory, compute, and accelerator resources that are available.<br><br>Use ML continuous delivery (CD). The best trained model needs to be automatically packaged and should be easily deployed at a moment's notice. CD enables you to test the model compatibility (for example, installed libraries that the model needs) with the platform on which it's supposed to be deployed. With CD, you can also validate that the same feature engineering is applied at | S17, S22, S44, S58 |

| | | | |
|---|---|---|---|
| | | serving time to ensure input feature consistency across the training and serving setup.<br><br>Model labels such as API and model compatibility are attributes you can use to drive model management. These attributes are captured in the metadata, and the system confirms these attributes at both deployment and rollback time. The only reliable way to ensure safe rollbacks is to run system tests that verify backward compatibility. These tests should run as the code is built—before each version is deployed. Doing these tests guarantee that the rollback is safe, and the guarantee itself is only possible with the metadata information in the model's signature. This scenario is similar to changing the database schema, except that the SQL queries in your application are updated during database schema changes. The ML Metadata library, available as part of TensorFlow, provides management of this metadata and the hooks needed to automate model deployment, verification, and rollback. | |
| | Managing and tracking trade-offs | ensure it's easy to continuously deploy new model versions after successful evaluation, and track them as training will occur every day.<br><br>streamlining the processes between modeling and production deployments<br><br>Deploy the model in response to model artifact trigger<br><br>Automate permissions and cluster creation to productionize registered models.<br><br>● Continuous delivery of models: An ML pipeline in production continuously delivers prediction services to new models that are trained on new data. The model deployment step, which serves the trained and validated model as a prediction service for online predictions, is automated. | S2, S10, S1, S3, S17, S31, S32, S33 |

| | | | |
|---|---|---|---|
| | Managing and tracking  trade-offs | Also, by tracking several versions of the ML model, it is possible to implement different deployment strategies such as “canary”- or “shadow”-deployment by analyzing the latest trained model’s performance improvement.<br><br>'AI Inference Model Update/ Roll out' Mechanism<br>- Blue Green Deployment<br>- Canary Deployment<br>- Multi Armed Bandit Deployment<br>- A/B<br>Zero downtime model release<br><br>Doesn’t deploying ML-pipelines without any downtime of your application sound like a dream? With a good MLOps infrastructure, it is within reach. This can be achieved if your infrastructure allows for multiple identical environments to run in parallel. The production pipeline will run in the production environment, while tests and adjustments can be made to models running in the testing environment. Then, when deadline day arrives, deploying a new version is just a matter of switching environments with minimal to no downtime at all. The same applies to retrained models within a pipeline: The infrastructure should allow you to only swap out the old model for the retrained one and keep the rest in place as it was, without interrupting the flow of data continuously going through your pipeline.<br><br>Three main production testing patterns exist that allow for mitigating risk in new deployments: shadow test pattern, A/B test pattern, and canary test pattern. | S10, S12, S28, S31, S33, S34, S39, S43, S50, S51, S53 |
| | ● Managing and tracking  trade-offs | What is your model release policy? Is A/B testing or multi-armed bandits testing required? (e.g., for measuring the effectiveness of the new model on business metrics and deciding what model should be promoted into the production environment)<br><br>AI/ML Service Routing (If Broken): Need to A/B Test/ Multi Armed Bandit in Production<br><br>Use a manual release gate<br><br>Champion/ challenger model gating<br>It is now becoming a best practice to introduce any new model (aka ‘challenger’) by first running it in production and measuring its performance vis-a-vis its predecessor (aka ‘champion’) for a defined | S9, S10, S3, S1, S17, S25, S28, S31 |

| | | | |
|---|---|---|---|
| | | timeframe, in order to determine whether the new model is worthy of becoming the new ‘champion’ by outperforming it. While this process ensures increasing levels of quality and continuous improvement of predictions and model stability, the important thing is for this process to become completely automated.<br><br>Testing the prediction service by calling the service API with the expected inputs, and making sure that you get the response that you expect. This test usually captures problems that might occur when you update the model version and it expects a different input.<br><br>Testing prediction service performance, which involves load testing the service to capture metrics such as queries per seconds (QPS) and model latency | |
| | Managing and tracking  trade-offs | A machine learning model’s lineage can be traced to numerous machine learning platforms and various programming languages, the creators of which are typically agnostic of actual production environments and their mission critical considerations. When organizations try to implement them in an environment suited for normal software applications, the manual integration creates friction, to the point where the model can’t be deployed without jeopardizing the stability of the production environment. As a result, the organization loses the option to scale this activity, as well as revenue and forfeits cost savings. | S1 |
| | Managing and tracking  trade-offs | ● Create a image for model<br>● deploy the model as web service on staging/QA environment<br>● Test the web service (QA)<br>● deploy the model as a web service on production environment<br>● Test the web service (production)<br>The Azure Machine Learning SDK provides an option to deploy directly to Azure Kubernetes Service from a registered model, creating limits on what security/metrics are in place. You can try to find an easier solution for clients to test their model, but it's best to develop a more robust deployment to AKS for production workloads.<br><br>● Verifying the compatibility of the model with the target infrastructure before you deploy your model. For example, you need to verify that the packages that are | S3, S17, S28, S48, S49 |

| | | | |
|---|---|---|---|
| | | required by the model are installed in the serving environment, and that the memory, compute, and accelerator resources that are available. | |
| | Managing and tracking trade-offs | Managing and tracking trade-offs Solutions including ML components will be required to manage trade-offs between multiple factors, for example in striking a balance between model accuracy and customer privacy, or explainability and the risk of revealing data about individuals in the data set. It may be necessary to provide intrinsic metrics to help customers balance these equations in production. It should also be anticipated that customers will need to be able to safely A/B test different scenarios to measure their impact upon this balance.<br><br>Also, despite intensive testing, errors may have crept in during the previous steps. For this reason, infrastructure should be provided to continuously collect data on model performance. The input values and the resulting predictions of the model should also be recorded, as far as this is compatible with the applicable data protection regulations. On the other hand, if privacy considerations are only introduced at this point, one has to ask how a sufficient amount of training data could be collected without questionable privacy practices. | S24, S50, S58 |
| Model and Data Monitoring | Model quality and drift should be monitored continuously | Model drift should be monitored in real-time so that users aren't getting stale recommendations as the news events are quite dynamic and users' reading behavior can become uncertain too.<br><br>collect ground truth labels on how long a person spends on an article to measure our model performance and the model drift.<br><br>Create alerts and automation to take corrective action In case of model drift due to differences in training and inference data.<br><br>A model monitoring solution implemented using Amazon SageMaker Model Monitor to monitor the production models' quality continuously.<br>.<br>This provides monitoring for the following: data drift, model drift, the bias in the models' predictions, and drifts in feature attributes. You can start with the default model monitor, which requires no coding. | S2, S7, S11, S15, S1, S3, S10, S13, S4, S16, S17, S19, S20, S23, S25, S35, S28, S29, S31, S32, S33, S34, S35, S43, S44, S48, S52, S54, S57 |

| | | | |
|---|---|---|---|
| | | 2. Some ML-specific monitoring is done in a way that only requires minimal coding/configuration e.g. some approaches to drift detection<br><br>Training data and live data will be different, which can trigger issues since the feature engineering pipeline might mismatch with the training/serving infrastructure, data, and even due to model drift.<br><br>Anomaly, performance, and bias warnings<br><br>Azure Application Insights. This monitoring service is used to detect performance anomalies.<br><br>Biased: Needs to Validate Trained ML Binary/ Service/ servable in any form for Bias detection Before Pushing<br><br>you can assure Continuous Validation of the whole machine learning life cycle. This includes:<br><br>Validation of ML performance measures, protection against concept drift;<br><br>Models tend to decay over time and you need the ability to retrain them on demand to ensure they remain relevant in production.<br><br>Need to Re-Train: After Pushing Your Model to Production, Your Model is already Out of Date. Google updates its SEO model/algorithm on average 500 times per year<br><br>- Monitoring model performance (and drift) over time to feed into thresholds for retraining and deployments<br><br>In DataRobot, users can import models built using different languages and on other available ML platforms. Models are then tested and deployed on leading ML execution environments. DataRobot monitors service health, data drift, and accuracy using reports and alerts systems.<br><br>● Actively monitor the quality of your model in production: Monitoring lets you detect performance degradation and model staleness. It acts as a cue to a new experimentation iteration and (manual) retraining of the model on new data.<br>● Model validation: This step occurs after you successfully train the model given the new data. You evaluate and | |

| | | | |
|---|---|---|---|
| | | validate the model before it's promoted to production. This offline model validation step consists of the following.<br>○ Producing evaluation metric values using the trained model on a test dataset to assess the model's predictive quality.<br>○ Comparing the evaluation metric values produced by your newly trained model to the current model, for example, production model, baseline model, or other business-requirement models. You make sure that the new model produces better performance than the current model before promoting it to production.<br>○ Making sure that the performance of the model is consistent on various segments of the data. For example, your newly trained customer churn model might produce an overall better predictive accuracy compared to the previous model, but the accuracy values per customer region might have large variance.<br>○ Making sure that you test your model for deployment, including infrastructure compatibility and consistency with the prediction service API.<br>complexity. ML teams need to add data, code and experiment tracking, monitor data to detect quality problems, monitor models to detect concept drift and improve model accuracy through the use of AutoML techniques and ensembles, and so on.<br>Nothing lasts forever—not even carefully constructed models that have been trained using mountains of well-labeled data. In these turbulent times of massive global change emerging from the COVID-19 crisis, ML teams need to react | |

| | | | |
|---|---|---|---|
| | | quickly to adapt to constantly changing patterns in real-world data. Monitoring machine learning models is a core component of MLOps to keep deployed models current and predicting with the utmost accuracy, and to ensure they deliver value long-term.<br><br>Model performance monitoring observes potential negative effects of data drift and concept drift by measuring how well the model handles real-world data and how real-world data is different from training data.<br><br>In this architecture inference inputs are provided to the model serving api's /predict route as either a batch or streamed input. The model serving framework processes the input using the machine learning model and returns the result of the prediction.<br><br>Within the model serving framework, metrics, including custom metrics, can be collected and recorded to the time series database. Additionally the model serving framework can pass the inputs to separate drift and outlier detection frameworks to record additional metrics to the time series database.<br><br>2. Unlike code, models degrade over time, which requires monitoring . After a trained model reaches production, it starts generating predictions from real data. In a stable environment, its accuracy wouldn't ever decline. But, alas, "Life changes and so does live data our model takes in," — Alexander Konduforov acknowledges. "This results in so-called model degradation — in other words, its predictive performance decreases over time. To prevent errors, we need continuous model monitoring which is not typical for DevOps practices." | |
| | Data quality and drift should be monitored continuously | Data drift should also be monitored alongside model drift to capture changes in user preferences and article features<br><br>A robust monitoring infrastructure should be able to proactively monitor data drift, feature importance, and model accuracy issues.<br><br>A model monitoring solution to monitor production models' performance to protect against both model and data drift. You can also use the performance metrics as feedback to help improve the models' future development and training.<br><br>Output Validation | S2, S1, S7, S13, S17, S22, S23, S25, S32, S33, S34, S5, S48, S52, S54 |

| | | | |
|---|---|---|---|
| | | While software works with explicit rules and relatively clear input-output patterns, machine learning models treat problems numerically and produce less clear outcomes. Validating their results requires a different approach.<br><br>For this reason, model validation is a core element of MLOps. We can validate our models to ensure they do what we intend to do and their performance does not degrade in time with new data (Concept Drift).<br><br>Overfitting is another common issue for which we want to look out. We can introduce a check to alert us when our model test substantially performs behind training.<br><br>you can assure Continuous Validation of the whole machine learning life cycle. This includes:<br><br>Data integrity alerts to ensure a consistent data schema and correspondence between training and production data sets.<br>Data monitoring and alerting in case of data drift;<br>Checking against data leakage by maintaining best practices in sampling and protecting the training process from label information;<br><br>We should also monitor the quality of the data so that it's what our automated workflow expects.<br><br>● Data validation: This step is required before model training to decide whether you should retrain the model or stop the execution of the pipeline. This decision is automatically made if the following was identified by the pipeline.<br>○ Data schema skews: These skews are considered anomalies in the input data, which means that the downstream pipeline steps, including data processing and model training, receives data that doesn't comply with the expected schema. In this case, you should stop the pipeline so the data science team can investigate. The team might release a fix or an update to the pipeline to handle | |

| | | | |
|---|---|---|---|
| | | these changes in the schema. Schema skews include receiving unexpected features, not receiving all the expected features, or receiving features with unexpected values.<br>○ Data values skews: These skews are significant changes in the statistical properties of data, which means that data patterns are changing, and you need to trigger a retraining of the model to capture these changes.<br>Concept drift<br>In a model, the target variable is what the model is trying to predict. It could be, for example, if a financial transaction is suspected as fraud or not fraud. When the statistical properties of a model change over time in an unexpected way (for example, a new fraud scheme appears that increases the overall amount of fraud), we have concept drift.<br><br>Data drift<br>If, however, the statistical properties of the input features change over time in an unexpected way, it will negatively impact the model's performance. For example, if users execute many more financial transactions than normal due to it being a holiday period, but the model was not trained to handle holiday periods, then the model performance may degrade (either missing fraud or flagging up too many transactions as suspicious).<br><br>Check the incoming files (if you receive data via a CSV/TSV/JSON dump). Don't assume the file format will be consistent—a manual job could be creating them.<br>Check the data's overall distribution.<br>Check the features' distributions.<br>Check for training-serving skew. | |
| | Pipelines should be monitored continuously | •We should also be able to get reports in the form of dashboard views and alerts when a pipeline fails, as we want to make sure our model is constantly updated.<br><br>Use the Azure Machine Learning UI and look under the section of the pipeline for the logs. Alternatively, these logs are | S2, S3, S1, S17, S19, S25, S35 |

| | | | |
|---|---|---|---|
| | | also written to blob and can be read from there as well using tools such as Azure Storage Explorer<br><br>Are all my pipelines available or are there errors? If the time it takes to handle a request suddenly rises from 0.01 to 52 seconds, that's an indication something is going wrong. Monitoring of the pipeline should be accessible to the Data Science team and the IT-department, as both have a role to play in maintaining a healthy system.<br><br>•We're dealing with pipelines, so we will need monitoring components to gauge the health of the pipelines.<br>•Training pipelines and the overall system should be monitored in terms of how much resources they consume; I/O, CPU usage, and memory.<br><br>● Have all their inputs (code, package dependencies, data, parameters) and the outputs (logs, metrics, data/features, artifacts, models) tracked for every step in the pipeline, in order to reproduce and/or explain our experiment results.<br>Feature pipeline changes<br>If there are changes in how a feature is computed in a feature pipeline, and an online model enriches its feature vector with that feature data from the online feature store, then this can negatively impact the model's performance. For example, if you change how to compute the number of transactions a user carries out, it may negatively impact the model's performance. | |
| | Create alerts and automation to take corrective actions | A robust monitoring infrastructure should be able to proactively monitor data drift, feature importance, and model accuracy issues<br><br>A system that can self-diagnose problems with predictions and notify relevant risk management stakeholders allows for tighter enterprise controls over machine learning projects in production.<br><br>Additional key MLOps elements that risk and compliance teams should be on the lookout for are built-in prediction explanation capabilities,<br>predictions-over-time analysis, and audit logs<br><br>Therefore, in addition to data drift monitoring, the next step is real-time protection. This gives us the option to take action on any individual prediction as soon as it occurs. To do this, we need | S1, S2, S16, S17, S25, S35 |

| | | | |
|---|---|---|---|
| | | the ability to define the rules that may lead to action on any individual prediction. These rules must be based on how certain we are that the prediction just generated is an accurate one.<br><br>The more certain we are of expected conditions at the time of prediction, the more we will trust that prediction. Furthermore, if the system itself tells us how sure it is, or how unsure it is, we can start to think of it as a humble system, not an arrogant system, and trust it even more.<br><br>Business leaders are now looking for assurance that their predictions are fast, accurate, and above all can be unbiased and trusted.<br><br>Troubleshooting and Triage: simply monitoring models is not sufficient without the ability to triage, troubleshoot, and rectify suspicious or poorly performing models.<br><br>Notify and alert on events in the machine learning lifecycle. Event examples include experiment completion, model registration, model deployment, and data drift detection. | |
| | Business evaluation metrics should be monitored | 3. Some monitoring is specific to the use case and model. For these cases we<br>find vendors offering export of data for integration into other tools<br><br>What ML-specific and business evaluation metrics need to be computed?<br><br>1. Basic information about the number and rate of predictions served and how<br>much resource this is using may be exposed by default<br><br>Guardrail metrics - MLOps processes always carry inherent risk so systems should support the use of guardrail metrics to aid in mitigating these risks.<br><br>● Monitor the model in production. Change is the only constant, and monitoring in-production models helps mitigate the risk from changes. You can watch performance drift and automatically inform the responsible team member to take the appropriate actions, like retraining on new data. The performance of a model might degrade due to a change in the data distribution (for example, data gathered from new sensors) and/or concept drift (the | S11, S1, S9, S11, S24, S22 |

| | | | |
|---|---|---|---|
| | | relationship between input and output data changes over time). Monitoring also confirms whether your hypothesis is valid on real-world data and aligns with your business objectives. | |
| | Actionable Multi-User/View Monitoring Dashboard (Governance) | •We should also be able to get reports in the form of dashboard views and alerts when a pipeline fails, as we want to make sure our model is constantly updated.<br><br>Use the Azure Machine Learning UI and look under the section of the pipeline for the logs. Alternatively, these logs are also written to blob and can be read from there as well using tools such as Azure Storage Explorer<br><br>Are all my pipelines available or are there errors? If the time it takes to handle a request suddenly rises from 0.01 to 52 seconds, that's an indication something is going wrong. Monitoring of the pipeline should be accessible to the Data Science team and the IT-department, as both have a role to play in maintaining a healthy system.<br><br>'User Interface' (PWA/ MobileApp / Web App)<br>Business Performance Dashboard<br>Model Performance Dashboard<br>System Operation Dashboard<br>Alerts<br><br>"For metrics that can be monitored in real-time it may be sufficient to expose dashboards with a tool such as grafana. However, sometimes the information that reveals whether a prediction was good or not is only available much later. For example, there may be a customer account opening process that flags a customer as risky. This could lead to a human investigation and only later will it be decided whether the customer was risky or not. For this reason it can be important to log the entire request and the prediction and also store the final decision. Then offline analysis run over a longer period can provide a wider view of how well the model is performing."<br><br>So that we can always inspect, compare, troubleshoot and rollback any models (and know where the changes are) consistently and reliably. Finally that brings us to monitoring, which is not only about producing a nice dashboard and maybe some alert threshold, but also about designing the environment and code in such a way that it's easy to get into the | S2, S3, S10, S34, S40, S54 |

| | | | |
|---|---|---|---|
| | | guts and other nasty details of any (set of) model(s) for diagnosis. | |
| | Monitor infrastructure performance metrics | I'm not only talking about ML-metrics like accuracy and F1-score, but also infrastructure performance metrics, like latency (the time that is added by the infrastructure when running a request) and errors:<br><br>– Define latency and network bandwidth performance requirements for your models.<br>– Continuously monitor and measure system performance.<br><br>Infrastructure is monitored per memory and compute requirements and to scale as needed.<br><br>Testing prediction service performance, which involves load testing the service to capture metrics such as queries per seconds (QPS) and model latency | S12, S2, S24, S5, S17, S35, S51 |
| Model Management (Or Model Lifecycle Mgt)<br><br>MLOps -automate each of these | Keep Track of each step of the machine learning life cycle | Azure Machine Learning provides an easy way to log at each step of the machine learning life cycle. The logs are stored in a blob container.<br><br>● Have all their inputs (code, package dependencies, data, parameters) and the outputs (logs, metrics, data/features, artifacts, models) tracked for every step in the pipeline, in order to reproduce and/or explain our experiment results.<br>● Use versioning for all the data and artifacts used throughout the pipeline.<br>● Store code and configuration in versioned Git repositories<br>● Use Continuous Integration (CI) techniques to automate the pipeline initiation, test automation, review and approval process.<br>Log your configuration details. This extends to both experimentation and production workflows. Model hyperparams? Log it. The threshold to convert probabilities to binary? Log it. The period of historical training data used? Log it. Heck, might as well log the commit hash too. | S3, S17, S19, S43, S46, S53, S57 |
| | Model updates and roll back | The ability to swap models without disturbing the production workflow is key to business continuity.<br>•Model versioning will also need to be in place for each retrained and redeployed model version so we can easily roll back if the retrained model performs worse. | S1<br>S2, S14, S43 |

| | | | |
|---|---|---|---|
| | | If the model is performing poorly enough, or deteriorating in performance at an alarming rate, the trigger you set will automatically rollback to a prior model stored in the model registry. Multiple models can also be deployed simultaneously, acting as a shadow model or a live A/B test, validating the new model's performance in the wild before it replaces the previous model. | |
| | Managing multiple models | There's often a need for multiple models to solve for difficult machine learning scenarios. One of the challenges of MLOps is managing these models, including:<br>● Having a coherent versioning scheme.<br>● Continually evaluating and monitoring all the models.<br>Traceable lineage of both code and data is also needed to diagnose model issues and create reproducible models. Custom dashboards can make sense of how deployed models are performing and indicate when to intervene. The team created such dashboards for this project. | |
| | Management of shared dependencies between training and operational phases | A number of ML approaches require the ability to have reusable resources that are applied both during training and during the pre-processing of data being passed to operational models. It is necessary to be able to synchronize these assets across the lifecycle of the model. e.g. Preprocessing, Validation, Word embeddings etc.<br><br>Training data and live data will be different, which can trigger issues since the feature engineering pipeline might mismatch with the training/serving infrastructure, data, and even due to model drift. | S24, S15, S41 |
| Model/Asset Governance | Model decisions should be explainable and interpretable (model analysis) | •Model decisions should be explainable because misinformation could be audited and needs to be regulated.<br><br>A model governance framework to make it easier to track, compare, and reproduce your ML experiments and secure your ML models.<br><br>- Managing root cause analysis for incident investigation<br>- Observability / interpretability<br>- Explainability and compliance<br><br>complexity. ML teams need to add data, code and experiment tracking, monitor data to detect quality problems, monitor models to detect concept drift and improve model accuracy through the use of AutoML | S2, S7, S19, S22, S34, S41, S43, S48, S54, S55 |

| | | | |
|---|---|---|---|
| | | techniques and ensembles, and so on.<br>Nothing lasts forever—not even carefully constructed models that have been trained using mountains of well-labeled data. In these turbulent times of massive global change emerging from the COVID-19 crisis, ML teams need to react quickly to adapt to constantly changing patterns in real-world data. Monitoring machine learning models is a core component of MLOps to keep deployed models current and predicting with the utmost accuracy, and to ensure they deliver value long-term.<br>• Evaluate model interpretability and explainability. Most trained models lack transparency, which usually results in violation of governance and accountability requirements. Model interpretability is the ability of a system to determine cause and effect; for example, if a patient has only one tumor, their chance of survival is 90%. Explainability is the extent to which the internal workings of an ML architecture can be defined in human-friendly terms. For example, a model's average impact on prediction of diabetes due to glucose is 40%, blood pressure is 15%, and age is 15%. It's important to note that without knowing what data was used to train the model, interpretability and explainability are difficult to implement in a system.<br>"end up making decisions based on Race or Gender.<br><br>Given concerns about bias, some organisations are putting an emphasis on being able to explain why a model made the prediction that it did in a given circumstance. This goes beyond reproducibility as being able to explain why a prediction was made can be a data science problem in itself ('explainability'). Some types of models such as neural networks are being referred to as 'black box' as it is not easy to see why a prediction would come about from inspecting their internal structure. There are black-box explanation techniques emerging (such as Seldon's Alibi library) but for now many | |

| | | | |
|---|---|---|---|
| | | organisations for whom explainability is a key concern are currently sticking to white box modelling techniques." ifferent-384z32f1 | |
| | Model approval and promotion process (Governance processes for managing the release cycle of MLOps assets) | a formal process designed to minimize risks associated with deploying the model, ensuring that all relevant business or technical stakeholders have signed off on the model.<br><br>Any trained model that's produced will be added to the model registry and tagged, to await promotion after validation is completed. For batch prediction, promotion can be the publishing of a scoring pipeline that uses this version of the model. For real-time scoring, the model can be tagged to indicate that it has been promoted.<br><br>- Managing release of preferred models into staging environments for integration and UAT<br>- Managing release of specific model versions into production environments for specific clients/deployments<br><br>- Managing release of new training sets to data science team<br>- Establishing thresholds for acceptable models<br><br>The last step is policy evaluation. A model is rejected from proceeding to deployment if it does not meet a policy requirement. Policies can be technical (for example, the model must meet or exceed a specific accuracy threshold) or nontechnical (for example, the model does not negatively target a specific population or does not use facial recognition at the individual level). The enforcement of the policy can be manual (that is, human authorized) or automated, depending on the maturity of the pipeline. In the manual case, the policy evaluation is completed using peer or group reviews. In the automated case, the policy evaluation is completed using programmatic tests such as performance metric tests or ethical standards tests.<br><br>Once the model goals have been met it needs to be approved for serving and versioning. This is a manual step and is usually done by a senior engineer or manager. The ideal process would require multiple approvers for redundancy and ensuring a high level of quality in deliveries. Once the model is approved it can go on to versioning and serving. | S1, S2, S24, S28, S19, S27, S50, S54 |

| | | | |
|---|---|---|---|
| | | On top of this, RS adds a lot of value through extra functionality, including a web portal that allows to search and browse all the available models. Each model has its own page with detailed information such as experiments using the model, monitoring tools, documentation, link to the training code, etc. It also provides a basic state machine that allows the author to transition the model through states like in-testing, production-ready, disabled, etc. Another interesting feature is the Playground that allows occasional users to just go ahead and use a model to see what it does. All these helps a lot with the Observability and Reusability requirements. | |
| | Model lineage tracking<br>Model version control<br>● Automated documentation<br>● Complete and searchable lineage tracking and audit trails for all production models | In these situations, organizations need to maintain complete model lineage tracking (approvals, model interactions, versions deployed, updates, etc.), something that is practically impossible to perform manually.<br><br>● Model version control<br>● Automated documentation<br>● Complete and searchable lineage tracking and audit trails for all production models<br><br>Track model lineage, model versions, and manage model artifacts and transitions through their lifecycle. Discover, share, and collaborate across ML models with the help of an open source MLOps platform such as MLflow.<br><br>This point relates to the monitoring of your pipeline (requirement 5), but there's a difference: monitoring allows you to see if and when performance is going down, version control/data lineage tracing allows you to pinpoint exactly which model version or batch of data caused it. Since ML applications often intertwine code, model and data, it's important that both code+model and data is tracked. Moreover, it will make comparing versions of models side-by-side much easier.<br><br>- Managing competitive training of model variants against each other in dev environments<br><br>Lineage, aka model lineage, is the inheritance chain of a machine learning model. Lineage encodes decisions made and lessons learned based on previous experiments in the chain of model-building decisions, and the difference between what was tried in the | S1, S7, S12, S24, S29, S32 |

| | | | |
|---|---|---|---|
| | | current experiment and what was tried in the past.<br><br>Lineage is essential to answering one of the core questions in machine learning development: the "have you tried X?" question. In large teams, it's impossible for everyone to know everything that's been tried already and what should be done next, without automation by tools supporting this need. In other words, lineage is the so what of model training automation. | |
| | Longevity of ML assets | Decision-making systems can be expected to require very long effective operating lifetimes. It will be necessary in some scenarios to be able to refer to instances of models across significant spans of time and therefore forward and backward compatibility issues, storage formats and migration of long running transactions are all to be considered. | S24 |
| | Managing and tracking trade-offs | Solutions including ML components will be required to manage trade-offs between multiple factors, for example in striking a balance between model accuracy and customer privacy, or explainability and the risk of revealing data about individuals in the data set. It may be necessary to provide intrinsic metrics to help customers balance these equations in production. It should also be anticipated that customers will need to be able to safely A/B test different scenarios to measure their impact upon this balance. | S24 |
| Serving Predictions | Batch and Online serving models | This is arguably the simplest architecture to use to serve your validated model in production. Basically, your model makes predictions offline and puts them in a data storage that can be served on-demand.<br><br>to serve predictions to users in real-time, as they request them<br><br>for use cases where data and/or compute must stay on-premise or on an edge device (like a mobile phone or micro-controller).<br><br>Manage the frequency of model refresh, inference request times and similar production-specifics in testing and QA.<br><br>● Online learning models: unlike the models we discussed so far, which are trained offline and used online to serve predictions, online learning models use algorithms and | S2 , S24, S10, S7, S34, S35 |

| | | | |
|---|---|---|---|
| | | techniques that can continuously improve its performance with the arrival of new data. They are constantly learning in production. This poses extra complexities, as versioning the model as a static artifact won't yield the same results if it is not fed the same data. You will need to version not only the training data, but also the production data that will impact the model's performance. | |
| | Support common serving patterns | They can be implemented using five patterns to put the ML model in production: Model-as-Service, Model-as-Dependency, Precompute, Model-on-Demand, and Hybrid-Serving.<br><br>Near real-time serving architecture<br><br>Embedded serving architecture<br><br>An enterprise-grade MLOps system should allow organizations to plug in their models and generate consistent API access for application teams on the other end, regardless of deployment environments and choice of cloud services and providers.<br><br>Enable REST API model endpoints.<br><br>Store the predictions to be accessed by the client<br><br>1. Model serving: The validated model is deployed to a target environment to serve predictions. This deployment can be one of the following:<br>○ Microservices with a REST API to serve online predictions.<br>○ An embedded model to an edge or mobile device.<br>○ Part of a batch prediction system.<br>On a technical level, any model training pipeline must produce an artifact that can be deployed into a production environment. The prediction results may be bad, but the model itself must be in a packaged state that allows it to be deployed directly into a production environment. This is a familiar idea from software development: continuous delivery. This packaging can be done in two different ways. Either our model is deployed as a separate service and accessed by the rest of our systems via an interface. Here, | S2, S7, S24, S9, S1, S17, S23, S28, S31, S33, S34, S35, S51 |

| | | | |
|---|---|---|---|
| | | the deployment can be done, for example, with TensorFlow Serving [24] or in a Docker container with a matching (Python) web server.<br>An alternative way of deployment is to embed it in our existing system. Here, the model files are loaded and input and output data are routed within the existing system. The problem here is that the model must be in a compatible format or a conversion must be performed before deployment. If such a conversion is performed, automated tests are essential. Here it must be ensured that both the original and the converted model deliver identical prediction results. | |
| | distributed model serving<br><br>multi-model prediction serving | 'AI Inference Service Routing' (e.g.,'Service Mesh' like Istio) | S9, S10 |
| | Model heterogeneity | MLOps simplifies model deployment by streamlining the processes between modeling and production deployments. It should not matter which platform or language the model was built on. | S1 |
| | Quantize and Speed Up Predictions | Slow: Need to Quantize and Speed Up Predictions<br><br>Low-latency prediction use cases requiring GPUs for inference<br><br>The model must make predictions really fast. Many models only decide if an icon will be shown on top of an accommodation card in search results. This is a tiny part of the whole web page so we can't really take a lot of time to do that. Furthermore, many models collaborate to construct a page, so even if individual models are not slow, the aggregated latency might become prohibitively high. | S10, S11, S36, S50 |
| Trigger | Data-driven trigger<br>Metrics-driven trigger<br>Schedule-driven trigger (need a Scheduler)<br>Ad-hoc manual trigger<br>ML Asset Lifecycle Event Trigger (Asset Storage Triggers) | Training architecture for event-based scenarios where an action (such as streaming data into a data warehouse) causes a trigger component to turn on either: workflow or message broker<br><br>retrain your model at scheduled intervals<br><br>The most significant of these shortcomings led to the development of a bridge between Azure Data Factory and Azure Machine Learning, which the team implemented by using a built-in connector in Azure Data Factory. They created this component set to facilitate the triggering and status monitoring necessary to make the process automation work. The most significant of these shortcomings led to the development of a bridge | S2, S4 , S5, S3, S7, S9<br>S4<br>S17, S19, S20 |

| | | | |
|---|---|---|---|
| | | between Azure Data Factory and Azure Machine Learning, which the team implemented by using a built-in connector in Azure Data Factory. They created this component set to facilitate the triggering and status monitoring necessary to make the process automation work.<br><br>Models are continuously monitored using configured KPIs like changes in input data distribution or changes in model performance. Triggers are set for more experimentations with new algorithms, data, and hyper-parameters that generate a new version of the ML pipeline.<br><br>A model retraining implementation that is based on the metric-based model retraining strategy. There are three main retaining strategies available for your model retraining implementation:<br>○ Scheduled: This kicks off the model retraining process at a scheduled time and can be implemented using an Amazon EventBridge scheduled event.<br>○ Event-driven: This kicks off the model retraining process when a new model retraining dataset is made available and can be implemented using an EventBridge event.<br>○ Metric-based: This is implemented by creating a Data Drift CloudWatch Alarm (as seen in Figure 1 above) that triggers your model retraining process once it goes off, fully automating your correction action for a model drift.<br><br>With a data change trigger or a timer trigger, the system can trigger the model validation CD pipeline and the scoring CD pipeline from the model orchestrator to run the same process that's run for code changes in the release branch prod environment.<br><br>Azure Data Factory does the following: | |

| | | | |
|---|---|---|---|
| | | Triggers an Azure Function to start data ingestion and a run of the Azure Machine Learning pipeline.<br>Launches a durable function to poll the Azure Machine Learning pipeline for completion.<br><br>to schedule the extraction and processing, as well as the retraining of the model on fresh data.<br><br>•Since a large number of articles come in at random based on events, we can set up a pull-based architecture that executes a workflow on schedule to retrain our model on new data.<br><br>Model Artifact trigger. Release pipelines get triggered every time a new artifact is available. A new model registered to Azure Machine Learning Model Management is treated as a release artifact. In this case, a pipeline is triggered for each new model is registered.<br><br>ML pipeline triggers<br>You can automate the ML production pipelines to retrain the models with new data, depending on your use case:<br>● On demand: Ad-hoc manual execution of the pipeline.<br>● On a schedule: New, labelled data is systematically available for the ML system on a daily, weekly, or monthly basis. The retraining frequency also depends on how frequently the data patterns change, and how expensive it is to retrain your models.<br>● On availability of new training data: New data isn't systematically available for the ML system and instead is available on an ad-hoc basis when new data is collected and made available in the source databases.<br>● On model performance degradation: The model is retrained when there is noticeable performance degradation.<br>● On significant changes in the data distributions (concept drift). It's hard to assess the complete performance of the online model, but you notice significant changes on the data distributions of the features that are used to perform the prediction. These changes suggest that your model has gone stale, and that needs to be retrained on fresh data. | |

| | | | |
|---|---|---|---|
| | | ML pipelines can be triggered manually, or preferably triggered automatically when:<br>1. The code, packages or parameters change<br>2. The input data or feature engineering logic change<br>3. Concept drift is detected, and the model needs to be re-trained with fresh data<br>Continuous retraining, on the other hand, retrains and redeploys machine learning models. Retraining occurs on a new training set that contains data from production. In other words, retraining happens with a fresh batch of feedback from production. In most cases, retraining is triggered when data drift or concept drift is observed. However, retraining can also be triggered when a test reveals the trained model is too biased, or when your business users give negative feedback on the results of your model. | |
| Security | Access control for ML assets | Security<br>– Restrict Access to ML systems.<br>– Ensure Data Governance.<br>– Enforce Data Lineage.<br>– Enforce Regulatory Compliance.<br><br>ML models in production are often part of a larger system where its output is consumed by applications that may or may not be known. This exposes multiple security risks. MLOps needs to provide security and access control to make sure outputs of ML models is used by known users only.<br><br>Secured trained model artefacts implemented using AWS Identity and Access Management (IAM) roles to ensure only authorized individuals have access.<br><br>This is important when assessing MLOps platforms, as some platforms are opinionated about which lifecycle activities are owned by which users — and how permissions are granted.<br><br>Robust access control management (LDAP, RBAC, etc.)<br><br>Security. All secrets and credentials are stored in Azure Key Vault and accessed in Azure Pipelines using variable groups. | S2, S5, S7, S11, S54, S56 |
| | Infrastructure security | network security<br>A network layer that implements the necessary network resources to ensure the MLOps solution is secured.<br><br>development environment security | S6, S7, S51, S56 |

| | | | |
|---|---|---|---|
| | | A secured development environment was implemented using an Amazon SageMaker Notebook Instance deployed to a custom virtual private cloud (VPC), and secured by implementing security groups and routing the notebook's internet traffic via the custom VPC. | |
| | Privacy Compliance for ML Assets | Security Vulnerability (If Hacked): Need to Train With Data Privacy (All your Data & other security is for a toss now as the Makert exposed AI Model has the essence of the underlying data.<br><br>'AI Security'<br>- Privacy Preserving Machine Learning<br>- Differential Privacy (offered by libraries such as Tensorflow Privacy)<br>- ε-differentially private algorithms (Google's Differential Privacy)<br>- Homomorphic Encryption / Data privacy in Real World AI/ML system (Intel n-Graph with Tensorflow, or, Microsoft SEAL)<br>- AI/ML API Security<br>- AI/ML RESTful or gRPC end point Authentication & Authorization<br>- AI/ML Federated User Pool & Identity Pool management (FIdM/ FIM)<br>- DevSecOps (Security as Code) for AI/ML Application development, deployment, monitoring,<br><br>https://github.com/DeepHiveMind/EnterpriseAI_Platform_MLOps/blob/master/README.md#deep-dive-into-ai-security<br>A very important characteristic of proper ML infrastructure is the way security is handled. Some companies would even consider this to be the key feature they base their selection on, as privacy regulations have become strict and (sensitive) data leaks do not look well on your company's resume. Consider the security of the data (uploading data to the cloud vs. data not leaving company servers), but certainly also the security of your precious ML intel.<br><br>- The right to use data for training purposes may require audit<br>- Legal compliance may require removing data from audited training sets, which in turn implies the need to retrain and redeploy models built from that data.<br>- The right to be forgotten or the right to opt out of certain data tracking may require per-user reset of model predictions | S10, S12, S24 |
| | Adversarial Robustness | Manage ML-specific security risks such as adversarial attacks | S10, S24, S10, S54 |

| | | | |
|---|---|---|---|
| | | Models are vulnerable to certain common classes of attack, such as:<br>- Black box attempts to reverse engineer them<br>- Model Inversion attacks attempting to extract data about individuals<br>- Membership Inference attacks attempting to verify the presence of individuals in data sets<br>- Adversarial attacks using tailored data to manipulate outcomes<br>It should be anticipated that there will be a need for generic protections against these classes of challenge across all deployed models. | |
| Infrastructure Management | Model and run-time agnostic infrastructure<br><br>Reproducible infrastructure<br><br>Versioned infrastructure<br><br>Infrastructure flexibility and portability (no-vendor lock in) | •Infrastructure has to be model and run-time agnostic.<br>•Our production environment should not require frequent dependency changes that might cause our data and model pipelines to fail while executing. It should mostly be deterministic.<br>•We also need a reproducible environment because it will enable our rollback strategy when the system fails.<br>•We should ensure we properly version every infrastructure package so conflicts due to dependency changes can be easily debugged.<br><br>Kubernetes as solution<br><br>•We would need to try avoiding dealing with serving dependency changes so that our automated pipeline runs don't fail due to these changes.<br><br>•Our production environment should not require frequent dependency changes that might cause our data and model pipelines to fail while executing. It should mostly be deterministic.<br><br>On-premise vs Cloud (for any infrastructure and platform services)<br><br>If you choose to take on the service provided by an infrastructure provider, make sure to check out what software they support. Nothing is as frustrating as being locked-in by a vendor only to find out that the package or language of your choice is not supported. Your code should determine the infrastructure, not the other way around. Is the infrastructure very rigid and will it only allow pre-selected packages by the vendor? Or is switching between a Python model or an R-based model as simple as the click of a button? | S2, S4, S12, S15, S18, S35 |

| | | | |
|---|---|---|---|
| | | Technology landscape may differ between development and deployment, model portability could be challenging – with constant change and development in cloud and container environments, this is a key technological risk that might be realized unless Models and codes become truly portable.<br><br>Some companies have been entrusted with private & sensitive data. It can't leave their servers because in the chance of a small vulnerability, the ripple effect would be catastrophic. This is where Hybrid cloud infrastructure for MLOps comes in.<br><br>Create reusable software environments. Use these environments for training and deploying models. | |
| | Automatic scaling | Azure Machine Learning Compute is a cluster of virtual machines on-demand with automatic scaling and GPU and CPU node options. The training job is executed on this cluster.<br><br>ML applications need scale and compute power that translates into complex infrastructure. For example, GPU may be necessary during experimentations and production scaling may be necessary dynamically.<br><br>Pipelines should be executed over scalable services or functions, which can span elastically over multiple servers or containers. This way, jobs complete faster, and computation resources are freed up once they do, saving significant costs. | S3, S5, S19, S52, S53 |
| | GPU and Hardware Accelerators | ML inference accelerators<br><br>Do you need ML inference accelerators (TPUs)?<br><br>This one depends on your use case, but if real-time predictions are necessary, one cannot have an infrastructure-related latency of 15 seconds. The latency of infrastructure usually depends on how models/pipelines are stored and spun up internally, how data is retrieved and also, how the results are given back again. Be sure to check the added latency of an infrastructure on your application, as it can differ greatly between services. | S28, S35, S37, S38, S44 |
| | QA and staging environment  vs production environment | As part of the release pipeline, the QA and staging environment is mimicked by deploying the scoring webservice image to Container Instances, which provides an easy, serverless way to run a container. | S3, S2, S17, S27 |

| | | | |
|---|---|---|---|
| | | Once the scoring webservice image is thoroughly tested in the QA environment, it is deployed to the production environment on a managed Kubernetes cluster.<br><br>Our production environment should not require frequent dependency changes that might cause our data and model pipelines to fail while executing. It should mostly be deterministic.<br><br>- Managing release of preferred models into staging environments for integration and UAT<br>- Managing release of specific model versions into production environments for specific clients/deployments | |
| | 'AI Scalability'<br>- Autoamted Load Balancing,<br>- Service Routing, AutoScaling,<br>- MultiZone Replication,<br>- Caching | This might be, or should be, a no-brainer, but still: Your architecture should allow you to control resources in a flexible way. You never know how many (or few) resources your pipeline will take up in the future and therefore the infrastructure should always scale (automatically) with it appropriately. This allows your pipeline to handle fluctuating amounts of data/number of requests and will also prevent you from having to adjust it in the future to be more efficient or scale better. Moreover, as unused resources are shut down automatically, there's no need to worry about paying for resources that are not used. | S10, S12 |
| | Use infrastructure as code (IaC) for managing Infrastructure | Use infrastructure as code (IaC) to automate the provisioning and configuration of your cloud-based ML workloads and other IT infrastructure resources. Model deployment pipeline that natively supports model rollbacks was implemented using AWS CloudFormation.<br><br>Maybe one of the most overlooked aspects of ML infrastructure is that it offers you the chance to standardize deployment of pipelines, not unlike 'regular' software deployments.<br><br>● Orchestrate ML experiments. By defining the training environment requirements as code and/or configuration, you can attain environmental-operational symmetry. For example, resources like compute and storage that your team needs during the experimentation phase can be made available in a similar fashion while they work in the production environment. This practice saves data | S7, S12, S22, S54 |

| | | | |
|---|---|---|---|
| | | scientists from the dance of “how do we get this model into production?” And it also promotes a cloud-first approach to elastically scale the training and to pay only when you need the resources. | |
| | Serverless | ```<br><br>--o 'Serverless Framework'<br>- FaaS: Apache OpenWhish/ OpenFaas,<br>- KNative<br><br>--o 'Hybrid Computing/ processing'<br>- Local Edge Processing (EdgeX/AWS Greegrass Core/Azure IoT Core/GCP IoT Core)<br>- Centralized Cloud Processing (AWS/Azure/GCP)<br><br>--o 'Modular Plug & Play of AI/ML system' leveraging<br>- Microservice (uS) governance framework' [API Gateway, uS Service Mesh, Automated uS Service Discovery & Registry, uS Service Config, uS service internal communication protocol etc]<br>- Containerization & CaaS (Docker, Kubernetes)<br>--o Optimized calculation frameworks (cuDF/ cuML)<br><br>``` | S10,.. |
| Non-functional Requirements | Unifying the release cycle for ML and conventional assets<br>Supporting ML assets as first class citizens within CI/CD systems<br>Improving quality through standardization across conventional and ML assets<br>Applying Static Analysis, Dynamic Analysis, Dependency Scanning and Integrity Checking to ML assets<br>Facilitating the re-use of ML approaches through template or ‘quickstart’ projects<br>Managing risk by aligning ML deliveries to appropriate governance processes | Instead of handing off a model artifact to be productionized, reframe your deliverable expectations. The deliverable is a pipeline structure that can build the model itself. This code is easy to package into a Docker image for automated deployments and requires documentation around all dependencies and environment requirements for it to function. | S24 |
| | Modularity and automation | The goal is to build a modularized, automated model training pipeline that can be reproduced in a production environment. Having a simple model with a solid MLOps infrastructure will derive value faster than throwing a complex model over the fence.<br>These tests verify that the data samples conform to the expected schema and distribution. Customize this test for other use cases and run it as a separate data sanity pipeline that gets triggered as new data arrives. For example, move the data test task to a data ingestion pipeline so you can test it earlier.<br><br>● Modularized code for components and pipelines: To construct | S14, S3, S17, S19, S32, S41, S49 |

| | | | |
|---|---|---|---|
| | | ML pipelines, components need to be reusable, composable, and potentially shareable across ML pipelines. Therefore, while the EDA code can still live in notebooks, the source code for components must be modularized. In addition, components should ideally be containerized to do the following:<br>○ Decouple the execution environment from the custom code runtime.<br>○ Make code reproducible between development and production environments.<br>○ Isolate each component in the pipeline. Components can have their own version of the runtime environment, and have different languages and libraries.<br>ML pipelines:<br>● Are built using micro-services (containers or serverless functions), usually over Kubernetes.<br>Production pipelines usually consist of:<br>1. Real-time data collection, validation and feature engineering logic<br>2. One or more model serving services<br>3. API services and/or application integration logic<br>4. Data and model monitoring services<br>5. Resource monitoring and alerting services<br>6. Event, telemetry and data/features logging services<br>The different services are interdependent. For example, if the inputs to a model change, the feature engineering logic must be upgraded along with the model serving and model monitoring services. These dependencies require online production pipelines (graphs) to reflect these changes. | |
| | Zero downtime model release | Doesn't deploying ML-pipelines without any downtime of your application sound like a dream? With a good MLOps infrastructure, it is within reach. This can be achieved if your infrastructure allows for multiple identical environments to run in parallel. The production pipeline will run in the production environment, while tests and | S12 |

| | | | |
|---|---|---|---|
| | | adjustments can be made to models running in the testing environment. Then, when deadline day arrives, deploying a new version is just a matter of switching environments with minimal to no downtime at all. The same applies to retrained models within a pipeline: The infrastructure should allow you to only swap out the old model for the retrained one and keep the rest in place as it was, without interrupting the flow of data continuously going through your pipeline. | |
| | Reliability, Performance Efficiency, Performance Efficiency | Reliability<br>– Manage changes to model inputs through automation.<br>– Train once and deploy across environments.<br>Performance Efficiency<br>– Optimize compute for your ML workload.<br>– Define latency and network bandwidth performance requirements for your models.<br>– Continuously monitor and measure system performance.<br>Cost Optimization<br>– Use managed services to reduce cost of ownership.<br>– Experiment with small datasets.<br>– Right size training and model hosting instances. | |
| Pipeline Types | Build pipeline.<br>Triggered every time code is checked in. | Builds the code and runs a suite of tests. | S3, S4 |
| | Retraining pipeline.<br>Training Pipeline | Retrains the model on a schedule or when new data becomes available.<br><br>We trigger data preparation and model training pipelines whenever the new data is available, or the source code for the pipeline has changed. | S3, S4, S( |
| | Release pipeline.<br><br>Integration tests | Operationalizes the scoring image and promotes it safely across different environments. | S3, S4 |
| | Scoring Pipeline | Scoring is the final step. The process runs the machine learning model to make predictions. This addresses the basic business use case requirement for demand forecasting. The team rates the quality of the predictions using the MAPE, which is a measure of prediction accuracy of statistical forecasting methods and a loss function for regression problems in machine learning. In this project, the team considered a MAPE of <= 45% significant.<br><br>The scoring CD pipeline is applicable for the batch inference scenario, where the same model orchestrator that's used for model validation triggers the published scoring pipeline. | S4, S4 |

| | | | |
|---|---|---|---|
| | Model validation CD pipeline | The goal of the model validation pipeline is to validate the end-to-end model training and scoring steps on the machine learning environment with actual data. Any trained model that's produced will be added to the model registry and tagged, to await promotion after validation is completed. For batch prediction, promotion can be the publishing of a scoring pipeline that uses this version of the model. For real-time scoring, the model can be tagged to indicate that it has been promoted. | |
| | data sanity pipeline | | S3 |
| | data ingestion pipeline | | S3 |
| | Data Pipeline | It's important for clients to frequently capture additional retraining data to keep their models up to date. If they don't already have a data pipeline, creating one will be an important part of the overall solution. Using a solution such as Datasets in Azure Machine Learning can be useful for versioning data to help with traceability of model | S4 |
| | batch scoring pipeline | | S4 |
| | real-time scoring pipeline | | S4 |
| | Experimentation (pipeline) | • Continuously experiment with new implementations to produce the model: To harness the latest ideas and advances in technology, you need to try out new implementations such as feature engineering, model architecture, and hyperparameters. For example, if you use computer vision in face detection, face patterns are fixed, but better new techniques can improve the detection accuracy. | S4, S17 |
| | ML Platform Container Image CI/CD Pipeline | | S7 |
| Implementation Approach | AI services (SaaS) | They are a fully managed set of services that enable you to quickly add ML capabilities to your workloads using API calls. Examples of these AWS services are Amazon Rekognition and Amazon Comprehend.<br><br>Use a built-in Amazon SageMaker algorithm or framework. With this option, a training dataset is the only input developers and data scientists have to provide when training their models. On the other hand, the trained model artefacts are the only input they need to deploy the models. This is the least flexible of the options available and a good fit for scenarios where off-the-shelf solutions meet your need. | S12, S7 |

| | | | |
|---|---|---|---|
| | ML Services (Platform services) | AWS provides managed services and resources (Amazon SageMaker suite, for example) to enable you to label your data and build, train, deploy, and operate your ML models.<br><br>Use pre-built ML platform service container images.<br><br>For this option, you need to provide two inputs to train your models, and they are your training scripts and datasets. Likewise, the inputs for deploying the trained models are your serving scripts and the trained model artefacts.<br><br>Customize/Adapt pre-built ML platform service container images<br><br>This is for more advanced use cases. You are responsible for developing and maintaining those container images. Therefore, you may want to consider implementing a CI/CD pipeline to automate the building, testing, and publishing of the customized Amazon SageMaker container images and then integrate the pipeline with your MLOps solution. | S12, S7 |
| | Complexity vs transparency | An indicator of excellent MLOps infrastructure is whether it makes complex (cloud) infrastructure abstract so that even people without much knowledge of the infrastructure can easily deploy models. However, it should also not be such a ‘black-box’, that when errors occur there is no way to tell what is going wrong and why. Insight into the internal processes of the infrastructure is necessary to debug the problem. The right balance between transparency and complexity is therefore vital. If you depend on an external service provider for your ML infrastructure, make sure to check that there’s a logging system in place, that gives an indication of what is happening internally. | S12 |
| Data platform features | data catalog<br>data storage<br>retention policies<br>data security<br>data capture<br>reporting integration<br>data discovery | There is no ML without data. Before everything else, ML teams need access to historical and/or online data from multiple sources, and they must catalog and organize the data in a way that allows for simple and fast analysis (for example, by storing data in columnar data structures, such as Parquet).<br>In most cases, the raw data cannot be used as-is for machine learning algorithms for various reasons such as:<br>1. The data is low quality (missing fields, null values, etc.) and requires cleaning and imputing.<br>2. The data needs to be converted to numerical or categorical values which | S11, S19, S33, S36, S38, S50, S53 |

| | | | |
|---|---|---|---|
| | | can be processed by algorithms.<br>3. The data is unstructured in text, json, image, or audio formats, and needs to be converted to tabular or vector formats.<br>4. The data needs to be grouped or aggregated to make it meaningful.<br>5. The data is encoded or requires joins with reference information.<br>The ML process starts with manual exploratory data analysis and feature engineering on small data extractions. In order to bring accurate models into production, ML teams must work on larger datasets and automate the process of collecting and preparing the data.<br>Furthermore, batch collection and preparation methodologies such as ETL, SQL queries, and batch analytics don't work well for operational or real-time pipelines. As a result, ML teams often build separate data pipelines which use stream processing, NoSQL, and containerized micro-services.<br>80% of data today is unstructured, so an essential part of building operational data pipelines is to convert unstructured textual, audio and visual data into machine learning- or deep learning-friendly data organization. | |
| | Data lineage | Data lineage is a concern for jobs such as cleaning ML data or training ML models, as well as ETL operations within data platforms. Delta lake, for example, comes at this from a data platform perspective (though can be used for ML) and Pachyderm comes at this from an MLOps perspective (though can be used for ETL) | S11 |
| Data Exploration (EDA) | | Structeyd vs Unstructured data validation clean up visualization<br><br>Iteratively explore, share, and prep data for the machine learning lifecycle by creating reproducible, editable, and shareable datasets, tables, and visualizations.<br><br>What programming language to use for analysis? | S2, S7, S19 |

| | | | |
|---|---|---|---|
| Data Preparation | | Iteratively transform, aggregate, and de-duplicate data to create refined features. Most importantly, make the features visible and shareable across data teams, leveraging a feature store. | S2, S7 |
| Feature Engineering | Transform, Binning, Temporal, Text, Image Feature Selection | Transform, Binning, Temporal, Text, Image Feature Selection | S2, S4, S7 |
| | features computed (workflows) during the training and prediction phases | How are features computed (workflows) during the training and prediction phases? | S9 |
| | APIs for feature engineering | Do we design APIs for feature engineering? | S9 |
| ML common features | | | S2, S31 |
| Model Exploration | | | S2 |
| Model Evaluation | | | S2 |
| Model Selection | | | S2 |
| Model FineTuning | | | |

| | | | |
|---|---|---|---|
| | | | |
| Model Validation/Evaluation | Cross Validation<br>Model Reporting<br>A/B testing | | S2, S4,S33 |
| Data Platform Fetaures | | Data ingestion. Any ML pipeline starts with data ingestion — in other words, acquiring new data from external repositories or feature stores, where data is saved as reusable “features”, designed for specific business cases. This step splits data into separate training and validation sets or combines different data streams into one, all-inclusive dataset.<br><br>Data validation. The goal of this step is to make sure that the ingested data meets all requirements. If anomalies are spotted, the pipeline can be automatically stopped until data engineers fix the problem. It also informs if your data changes over time, highlighting differences between training sets and live data your model uses in production.<br><br>Data preparation. Here, raw data is cleansed and gets the right quality and format so your model can consume it. At this step data scientists may intervene to combine raw data with domain knowledge and build new features, using capabilities of DataRobot, Featuretools, or other solutions for feature engineering. | S2, S10, S31 |
| Data Transformation | condition the data into a standard format (Data conditioning) | Data used with these models comes from many private and public sources. Because the original data is disorganized, it's impossible for the machine learning model to consume it in its raw state. The data scientists must condition the data into a standard format for machine learning model consumption.<br><br>Much of the pilot field test focused on conditioning the raw data so that the machine learning model could process it. In an MLOps system, the team should automate this process, and track the outputs.<br><br>Data Transformation<br>Rarely will we train our machine learning model directly on raw data. Instead, we generate features from the raw data. In the context of machine learning, | S2, S4, S23, S27 |

| | | | |
|---|---|---|---|
| | | a feature is one or more processed data attributes that an algorithm uses to make predictions. This could be a temperature value, for example, but in the case of deep learning applications also highly abstract features in images. To extract features from raw data, we will apply various transformations. We will typically define these transformations in code, although some ETL tools also allow us to define them using a graphical interface. Our transformations will either be run as batch jobs on larger sets of data, or we will define them as long-running streaming applications that continuously transform data.<br>We also need to split our dataset into training and testing datasets. To train a machine learning model, we need a set of training data. To test how our model performs with previously unknown data, we need another structurally identical set of test data. Therefore, we split our original transformed data set into two data sets. These must not overlap, meaning that the same data point does not occur twice in them. A common split here is to use 70 percent of the dataset for training and 30 percent for testing the model.<br>The exact split of the data sets depends on the context. For time-series data, sequential slices from the series should be chosen, while for image processing, random images from the data set should be chosen since they have no sequential relation to each other.<br>For non-sequential data, the individual data points can also be placed in a (pseudo-)random order. We also want to perform this process in a reproducible and automated manner rather than manually. A pleasantly usable tool for management and coordination here is Apache Airflow [10]. Here, according to the “pipeline as code” principle, one can define various pipelines in the form of a data flow graph, connect a wide variety of systems, and thus perform the desired transformations. | |
| Dataflow management | to manage your workflows and scheduled tasks. | | S8, S30, S33 |
| Roles and Responsibilities | Business Analyst | The role of the business analyst is to define the use case and business requirements for the AI/ML project. Business analysts are also responsible for tracking the ROI during the lifetime of the AI/ML project and conveying | 37 |

| | | | |
|---|---|---|---|
| | | results to leadership in terms of value achieved and dollars captured. The business analyst must ensure the machine learning capability supports the overall solution. | |
| | Data Engineer | The role of the data engineer is to take in raw data and curate (clean, prepare and organize) it for the data scientist to consume. Data engineers therefore need the ability to produce and reproduce data pipelines on demand. They need services that will be able to automate the tasks of pulling in raw data and curating it. Note the data engineers have to be able to monitor the raw and curated data to make certain it is of high quality. And it is imperative that they detect and fix any issues with the data before the data scientist receives and starts work on it. Additionally, the data engineer should be the point of contact for data security and access control for audit purposes and ensuring that no unknown data sources are used.<br><br>Optimize the retrieval and use of data to power ML models.<br><br>Data Engineers might be building pipelines to make data accessible.<br><br>For Data Engineers: Cloudera Data Engineering offers an integrated, purpose-built experience for data engineers with these capabilities:<br><br>Containerized, managed Spark service with multiple version deployments and autoscaling compute, governed and secured with Cloudera SDX.<br><br>Apache Airflow orchestration with open preferred tooling to easily orchestrate complex data pipelines and manage/schedule dependencies.<br><br>Visual troubleshooting that enables real-time visual performance profiling and complete observability/alerting capabilities to resolve issues quickly.<br><br>Simplified job management and APIs in diverse languages to automate the pipelines for any service. | 37, 24, 43, 63 |
| | Data Scientist | The role of a data scientist is to analyze a data science problem and through repeatable experiments produce an analysis using algorithms and models along with a variety of programming languages (such as Python or R) and their associated libraries. Common tools include using PyCharm, Anaconda or JupyterLab Integrated Development | 37, 24, 43, 63 |

| | | | |
|---|---|---|---|
| | | Environments (IDEs) with Jupyter notebooks for experimentation, and having a reproducible and shareable work environment to deliver their solutions on. Data scientists should not be overly concerned with underlying infrastructure, code repositories, and model deployment and delivery methods. Instead, their focus should be on experimenting and generating high quality models for the use case.<br><br>Build models that address the business question or needs brought by subject matter experts.<br>Deliver operationalizable models so that they can be properly used in the production environment and with production data.<br>Assess model quality (of both original and tests) in tandem with subject matter experts to ensure they answer initial business questions or needs.<br><br>Data Scientists are worried about building and improving the ML model.<br><br>For Data Scientists: Cloudera Data Science Workbench offers a collaborative development environment, flexible resource allocations, notebook interface, and personalized repetitive use cases with Applied Research documents created by Cloudera's Fast Forward Labs. | |
| | Application Developers / SE | The focus of the application developer is to run the model serving microservice and possibly other applications that use the model serving microservice in a production environment. They spend a great deal of effort in building and maintaining the serving pipelines which automate the model rollout process into production. This step is crucial as it takes the model from the data scientist and developer workspace to a code repository and then ensures a secure and smooth delivery (usually via containerization) into the production environment. In some cases, application developers may not have the machine learning know-how to streamline the delivery of a model and can get help from the data scientist or ML Engineer.<br><br>Integrate ML models in the company's applications and systems.<br>Ensure that ML models work seamlessly with other non-ML based applications.<br><br>Then Machine Learning Engineers or developers will | 37, 24, 43 |

| | | | |
|---|---|---|---|
| | | have to worry about how to integrate that model and release it to production. | |
| | ML Engineer | Machine learning engineers are experts at operationalizing machine learning models: taking models out of experiments and making them run in a production environment. They work very closely with application developers to streamline the delivery of the model via the serving pipeline and also manage the monitoring of those models. In some cases, ML Engineers may step in as the data scientist to experiment and generate models; however, their focus is less on experimentation and more on the delivery of models into an application. | 37 |
| | ML Architect | Ensure a scalable and flexible environment for ML model pipelines, from design to development and monitoring. Introduce new technologies when appropriate that improve ML model performance in production | 24 |
| | IT Operations<br><br>DevOps Engineer ? | IT operations is concerned with the overall usability of the platform. They are also responsible for infrastructure monitoring (that is, compute, networking and storage), platform maintenance (for example, Kubernetes), platform security, and hardware acceleration (for example, Nvidia GPUs). It is vital that the other roles work with IT Operations to ensure that anything nonstandard in the environment is vetted before use. | 37 |
| | SRE | Conduct and build operational systems and test for security, performance, availability. Continuous Integration/Continuous Delivery pipeline management. | 24 |
| | Business Users | For Business Users: Cloudera Data Visualization enables the creation, publication, and sharing of interactive data visualizations to accelerate production ML workflows from raw data to business impact. With this product:<br><br>The data science teams can communicate and collaborate effectively across the data lifecycle; while experimenting, training, and deploying models across the business.<br><br>The business teams can leverage explainable AI, acquire broader data literacy, and make informed business decisions. | S51 |

| | | | |
|---|---|---|---|
| Maturity Model | | | |