Planning for successful MLOps

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Planning for successful MLOps

Table of Contents

Planning for successful MLOps

Bruno Klein, Amazon Web Services (AWS)

December 2021 (document history)

Deploying machine learning (ML) solutions in production introduces many challenges that don't arise in standard software development projects. ML solutions are more complex and trickier to get right in the first place. They also exist in usually volatile environments, where the data distribution deviates significantly over time for a variety of expected and unexpected reasons.

These issues are further aggravated by the fact that many ML practitioners don't come from a software engineering background, so they might not be familiar with the best practices of this industry, such as writing testable code, modularizing components, and using version control effectively. These challenges create technical debt, and solutions become more complex and difficult to maintain over time, powered by a compounding effect, for ML teams.

This guide enumerates ML operations (MLOps) best practices that help mitigate these challenges in ML projects and workloads.

Because MLOps is a cross-cutting concern, these issues affect not only deployment and monitoring processes, but the whole model lifecycle. In this guide, MLOps best practices are organized into four major areas:

- Data
- Training
- Deployment
- Monitoring

Targeted business outcomes

Deploying ML models in production is a task that requires continuous effort and a dedicated team to maintain these resources throughout their lifetime (in some cases, even years). ML models can unlock considerable value out of business data, but they have high costs. To minimize costs, enterprises should follow good practices in software development and data science. They should be aware of the nuances of ML systems, such as data drift, which makes models perform unexpectedly after a while. By being aware of these concerns, enterprises can meet their business goals safely and with agility in the short term and long term.

There are several kinds of ML models, and the industries they target have varying types of ML tasks and business problems, so you need to consider a different set of concerns for each model and industry. The practices laid out in this guide are not specific to a model or business, but apply to a broad set of models and industries to improve deployment times, generate higher productivity, and build stronger governance and security.

Putting models into production is a multi-disciplinary task that requires data scientists, machine learning engineers, data engineers, and software engineers. When you build your ML team, we recommend that you target these skills and backgrounds.

Data

DevOps is a software engineering practice that deals with the operationalization of software. Common elements of DevOps are version-controlled code, continuous integration and continuous delivery (CI/CD) pipelines, unit testing, and reproducible code builds and deployment, which all involve code. ML models are a product of code and data, so data has to meet the same standards as code. MLOps must address data-related questions such as how to maintain data quality, how to identify edge cases in data, how to secure data, and how to make data more maintainable.

Topics

Labeling

Provide clear labeling instructions

A dataset might include ambiguous samples that result in inconsistent labeling across the entire dataset. For example, consider the task of labeling images that contain a dog. Some samples might contain only a glimpse of the animal. Should those be marked with a positive or negative label? This type of problem might be solved by providing clear and objective instructions to labelers.

Use majority voting

Now consider the issue of labeling a speech-to-text dataset that contains noisy audio with words that are phonetically similar or identical to others, such as *know* and *go*, *shoe* and *two*, *cry* and *high*, or *right* and *write*. In this case, labelers might label these samples inconsistently.

To maintain a high degree of correctness in labeling, a common approach is to use majority voting, in which the same data sample is given to multiple workers and their results are aggregated. This method and its more sophisticated variations are described in the blog post Use the wisdom of crowds with Amazon SageMaker Ground Truth to annotate data more accurately on the AWS Machine Learning blog.

Splits and data leakage

Data leakage happens when your model gets data during inference—the moment the model is in production and receiving prediction requests—that it shouldn't have access to, such as data samples that were used for training, or information that will not be available when the model is deployed in production.

If your model is inadvertently tested on training data, data leakage might cause overfitting. Overfitting means that your model doesn't generalize well to unseen data. This section provides best practices to avoid data leakage and overfitting.

Split your data into at least three sets

One common source of data leakage is dividing (splitting) your data improperly during training. For example, the data scientist might have knowingly or unknowingly trained the model on the data that was used for testing. In such situations, you might observe very high success metrics that are caused by overfitting. To solve this issue, you should split the data into at least three sets: `training`, `validation`, and `testing`.

By splitting your data in this way, you can use the `validation` set to choose and tune the parameters you use to control the learning process (*hyperparameters*). When you have achieved a desired result or reached a plateau of improvement, perform evaluation on the `testing` set. The performance metrics for the `testing` set should be similar to the metrics for the other sets. This indicates there is no distribution mismatch between the sets, and your model is expected to generalize well in production.

Use a stratified split algorithm

When you split your data into `training`, `validation`, and `testing` for small datasets, or when you work with highly imbalanced data, make sure to use a stratified split algorithm. Stratification guarantees that each split contains approximately the same number or distribution of classes for each split. The scikit-learn ML library already implements stratification, and so does Apache Spark.

For sample size, make sure that the validation and testing sets have enough data for evaluation, so you can reach statistically significant conclusions. For example, a common split size for relatively small datasets (fewer than 1 million samples) is 70%, 15%, and 15%, for `training`, `validation`, and `testing`. For very large datasets (more than 1 million samples), you might use 90%, 5%, and 5%, to maximize the available training data.

In some use cases, it's useful to split the data into additional sets, because the production data might have experienced radical, sudden changes in distribution during the period in which it was being collected. For example, consider a data collection process for building a demand forecasting model for grocery store items. If the data science team collected the `training` data during 2019 and the `testing` data from January 2020 through March 2020, a model would probably score well on the `testing` set. However, when the model is deployed in production, the consumer pattern for certain items would have already changed significantly because of the COVID-19 pandemic, and the model would generate poor results. In this scenario, it would make sense to add another set (for example, `recent_testing`) as an additional safeguard for model approval. This addition could prevent you from approving a model for production that would instantly perform poorly because of distribution mismatch.

In some instances, you might want to create additional `validation` or `testing` sets that include specific types of samples, such as data associated with minority populations. These data samples are important to get right but might not be well represented in the overall dataset. These data subsets are called *slices*.

Consider the case of an ML model for credit analysis that was trained on data for an entire country, and was balanced to equally account for the entire domain of the target variable. Additionally, consider that this model might have a `City` feature. If the bank that uses this model expands its business into a specific city, it might be interested in how the model performs for that region. So, an approval pipeline should not only assess the model's quality based on the test data for the entire country, but should evaluate test data for a given city slice as well.

When data scientists work on a new model, they can easily assess the model's capabilities and account for edge cases by integrating under-represented slices in the validation stage of the model.

Consider duplicate samples when doing random splits

Another, less common, source of leakage is in datasets that might contain too many duplicate samples. In this case, even if you split the data into subsets, different subsets might have samples in common. Depending on the number of duplicates, overfitting might be mistaken for generalization.

Consider features that might not be available when receiving inferences in production

Data leakage also happens when models are trained with features that are not available in production, at the instant the inferences are invoked. Because models are often built based on historical data, this data might be enriched with additional columns or values that were not present at some point in time. Consider the case of a credit approval model that has a feature that tracks how many loans a customer has made with the bank in the past six months. There is a risk of data leakage if this model is deployed and used for credit approval for a new customer who doesn't have a six-month history with the bank.

Amazon SageMaker Feature Store helps solve this problem. You can test your models more accurately with the use of time travel queries, which you can use to view data at specific points in time.

Feature store

Using SageMaker Feature Store increases team productivity, because it decouples component boundaries (for example, storage versus usage). It also provides feature reusability across different data science teams within your organization.

Use time travel queries

Time travel capabilities in Feature Store help reproduce model builds and support stronger governance practices. This can be useful when an organization wants to assess data lineage, similar to how version control tools such as Git assess code. Time travel queries also help organizations provide accurate data for compliance checks. For more information, see Understanding the key capabilities of Amazon SageMaker Feature Store on the AWS Machine Learning blog.

Use IAM roles

Feature Store also helps improve security without affecting team productivity and innovation. You can use AWS Identity and Access Management (IAM) roles to give or restrict granular access to specific features for specific users or groups.

For example, the following policy restricts access to a sensitive feature in Feature Store.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Deny",
            "Action": "*",
            "Resource": "arn:aws:s3:::us-east-2-12345678910-features/12345678910/
sagemaker/us-east-2/offline-store/doctor-appointments"
        }
    ]
}
```

For more information about data security and encryption using Feature Store, see Security and access control in the SageMaker documentation.

Use unit testing

When data scientists create models based on some data, they often make assumptions about the distribution of the data, or they perform a thorough analysis to fully understand the data properties. When these models are deployed, they eventually become stale. When the dataset becomes outdated, data scientists, ML engineers, and (in some cases) automated systems retrain the model with new data that is fetched from an online or offline store.

However, the distribution of this new data might have changed, which could affect the current algorithm's performance. An automated way to check for these types of issues is to borrow the concept of *unit testing* from software engineering. Common things to test for include the percentage of missing values, the cardinality of categorical variables, and whether real valued columns adhere to some expected distribution by using a framework such as hypothesis test statistics (*t*-test). You might also want to validate the data schema, to make sure it hasn’t changed and won’t generate invalid input features silently.

Unit testing requires understanding the data and its domain so you can plan the exact assertions to perform as part of the ML project. For more information, see Testing data quality at scale with PyDeequ on the AWS Big Data blog.

Training

MLOps is concerned with the operationalization of the ML lifecycle. Therefore, it must facilitate the work of data scientists and data engineers toward creating pragmatic models that achieve business needs and work well in the long term, without incurring technical debt.

Follow the best practices in this section to help address model training challenges.

Topics

Create a baseline model

When practitioners face a business problem with an ML solution, typically their first inclination is to use the state-of-the-art algorithm. This practice is risky, because it's likely that the state-of-the-art algorithm hasn't been time-tested. Moreover, the state-of-the-art algorithm is often more complex and not well understood, so it might result in only marginal improvements over simpler, alternative models. A better practice is to create a baseline model that is relatively quick to validate and deploy, and can earn the trust of the project stakeholders.

When you create a baseline, we recommend that you evaluate its metric performance whenever possible. Compare the baseline model's performance with other automated or manual systems to guarantee its success and to make sure that the model implementation or project can be delivered in the medium and long term.

The baseline model should be further validated with ML engineers to confirm that the model can deliver the non-functional requirements that have been established for the project, such as inference time, how often the data is expected to shift distribution, if the model can be easily retrained in these cases, and how it will be deployed, which will affect the cost of the solution. Get multi-disciplinary viewpoints on these questions to increase the chance that you're developing a successful and long-running model.

Data scientists might be inclined to add as many features as possible to a baseline model. Although this increases the capability of a model to predict the desired outcome, some of these features might generate only incremental metric improvements. Many features, especially those that are highly correlated, might be redundant. Adding too many features increases costs, because it requires more compute resources and tuning. Too many features also affects day-to-day operations for the model, because data drift becomes more likely or happens faster.

Consider a model in which two input features are highly correlated, but only one feature has causality. For example, a model that predicts whether a loan will default might have input features such as customer age and income, which might be highly correlated, but only income should be used to give or deny a loan. A model that was trained on these two features might be relying on the feature that doesn't have causality, such as age, to generate the prediction output. If, after going to production, the model receives inference requests for customers older or younger than the average age included in the training set, it might start to perform poorly.

Furthermore, each individual feature could potentially experience distribution shift while in production and cause the model to behave unexpectedly. For these reasons, the more features a model has, the more fragile it is with respect to drift and staleness.

Data scientists should use correlation measures and Shapley values to gauge which features add sufficient value to the prediction and should be kept. Having such complex models increases the chance of a feedback loop, in which the model changes the environment that it was modeled for. An example is a recommendation system in which consumer behavior might change because of a model's recommendations. Feedback loops that act across models are less common. For example, consider a recommendation system that recommends movies, and another system that recommends books. If both models target the same set of consumers, they would affect each other.

For each model that you develop, consider which factors might contribute to these dynamics, so you know which metrics to monitor in production.

Use a data-centric approach and error analysis

If you use a simple model, your ML team can focus on improving the data itself, and taking a data-centric approach instead of a model-centric approach. If your project uses unstructured data, such as images, text, audio, and other formats that can be assessed by humans (compared with structured data, which might be more difficult to map to a label efficiently), a good practice to get better model performance is to perform error analysis.

Error analysis involves evaluating a model on a validation set and checking for the most common errors. This helps identify potential groups of similar data samples that the model might be having difficulties getting right. To perform error analysis, you can list inferences that had higher prediction errors, or rank errors in which a sample from one class was predicted as being from another class, for example.

Architect your model for fast iteration

When data scientists follow best practices, they can experiment with a new algorithm or mix and match different features easily and quickly during proof of concept or even retraining. This experimentation contributes to success in production. A good practice is to build upon the baseline model, employing slightly more complex algorithms and adding new features iteratively while monitoring performance on the training and validation set to compare actual behavior with expected behavior. This training framework can provide an optimal balance in prediction power and help keep models as simple as possible with a smaller technical debt footprint.

For fast iteration, data scientists must swap different model implementations in order to determine the best model to use for particular data. If you have a large team, a short deadline, and other project management-related logistics, fast iteration can be difficult without a method in place.

In software engineering, the Liskov substitution principle is a mechanism for architecting interactions among software components. This principle states that you should be able to replace one implementation of an interface with another implementation without breaking the client application or the implementation. When you write training code for your ML system, you can employ this principle to establish boundaries and encapsulate the code, so you can replace the algorithm easily, and try out new algorithms more effectively.

For example, in the following code, you can add new experiments by just adding a new class implementation.

```
from abc import ABC, abstractmethod

from pandas import DataFrame


class ExperimentRunner(object):

    def __init__(self, *experiments):
        self.experiments = experiments
    def run(self, df: DataFrame) -> None:
        for experiment in self.experiments:
            result = experiment.run(df)
            print(f'Experiment "{experiment.name}" gave result {result}')


class Experiment(ABC):

    @abstractmethod
    def run(self, df: DataFrame) -> float:
        pass

    @property
    @abstractmethod
    def name(self) -> str:
        pass


class Experiment1(Experiment):

    def run(self, df: DataFrame) -> float:
        print('performing experiment 1')
        return 0

    def name(self) -> str:
        return 'experiment 1'


class Experiment2(Experiment):

    def run(self, df: DataFrame) -> float:
        print('performing experiment 2')
        return 0

    def name(self) -> str:
        return 'experiment 2'


class Experiment3(Experiment):

    def run(self, df: DataFrame) -> float:
        print('performing experiment 3')
        return 0
    def name(self) -> str:
        return 'experiment 3'


if __name__ == '__main__':
    runner = ExperimentRunner(*[
        Experiment1(),
        Experiment2(),
        Experiment3()
    ])
    df = ...
    runner.run(df)
```

Track your ML experiments

When you work with a large number of experiments, it's important to gauge whether the improvements you observe are a product of implemented changes or chance. You can use Amazon SageMaker Experiments to easily create experiments and associate metadata with them for tracking, comparison, and evaluation.

Reducing the randomness of the model build process is useful for debugging, troubleshooting, and improving governance, because you can predict the output model inference with more certainty, given the same code and data.

It is often not possible to make a training code fully reproducible, because of random weight initialization, parallel compute synchronicity, inner GPU complexities, and similar non-deterministic factors. However, properly setting random seeds, to make sure that each training run starts from the same point and behaves similarly, significantly improves outcome predictability.

Troubleshoot training jobs

In some cases, it might be difficult for data scientists to fit even a very simple baseline model. In this case, they might decide that they need an algorithm that can better fit complex functions. A good test is to use the baseline of a very small part of the dataset (for example, around 10 samples) to make sure that the algorithm overfits on this sample. This helps rule out data or code issues.

Another helpful tool for debugging complex scenarios is Amazon SageMaker Debugger, which can capture issues related to algorithmic correctness and infrastructure, such as optimal compute usage.

Deployment

In software engineering, putting code in production requires due diligence, because code might behave unexpectedly, unforeseen user behavior might break software, and unexpected edge cases can be found. Software engineers and DevOps engineers usually employ unit tests and rollback strategies to mitigate these risks. With ML, putting models in production requires even more planning, because the real environment is expected to drift, and on many occasions, models are validated on metrics that are proxies for the real business metrics they are trying to improve.

Follow the best practices in this section to help address these challenges.

Topics

Automate the deployment cycle

The training and deployment process should be entirely automated to prevent human error and to ensure that build checks are run consistently. Users should not have write access permissions to the production environment.

Amazon SageMaker Pipelines and AWS CodePipeline help create CI/CD pipelines for ML projects. One of the advantages of using a CI/CD pipeline is that all code that is used to ingest data, train a model, and perform monitoring can be version controlled. Git and AWS CodeCommit are version control tools that you can use. Sometimes you have to retrain a model by using the same algorithm and hyperparameters, but different data. The only way to verify that you're using the correct version of the algorithm is to use source control and tags. You can use the default project templates provided by SageMaker as a starting point for your MLOps practice.

When you create CI/CD pipelines to deploy your model, make sure to tag your build artifacts with a build identifier, code version or commit, and data version. This practice helps you troubleshoot any deployment issues. Tagging is also sometimes required for models that make predictions in highly regulated fields. The ability to work backward and identify the exact data, code, build, checks, and approvals associated with an ML model can help improve governance significantly.

Part of the job of the CI/CD pipeline is to perform tests on what it is building. Although data unit tests are expected to happen before the data is ingested by a feature store, the pipeline is still responsible for performing tests on the input and output of a given model and for checking key metrics. One example of such a check is to validate a new model on a fixed validation set and to confirm that its performance is similar to the previous model by using an established threshold. If performance is significantly lower than expected, the build should fail and the model should not go into production.

The extensive use of CI/CD pipelines also supports pull requests, which help prevent human error. When you use pull requests, every code change must be reviewed and approved by at least one other team member before it can go to production. Pull requests are also useful for identifying code that doesn't adhere to business rules and for spreading knowledge within the team.

For more information about performing code reviews with AWS CodeCommit, see the blog post Using CodeCommit Pull Requests to request code reviews and discuss code on the AWS DevOps blog.

Choose a deployment strategy

MLOps deployment strategies include blue/green, canary, shadow, and A/B testing.

Blue/green

Blue/green deployments are very common in software development. In this mode, two systems are kept running during development: blue is the old environment (in this case, the model that is being replaced) and green is the newly released model that is going to production. Changes can easily be rolled back with minimum downtime, because the old system is kept alive. For more in-depth information about blue/green deployments in the context of SageMaker, see the blog post Safely deploying and monitoring Amazon SageMaker endpoints with AWS CodePipeline and AWS CodeDeploy on the AWS Machine Learning blog.

Canary

Canary deployments are similar to blue/green deployments in that both keep two models running together. However, in canary deployments, the new model is rolled out to users incrementally, until all traffic eventually shifts over to the new model. As in blue/green deployments, risk is mitigated because the new (and potentially faulty) model is closely monitored during the initial rollout, and can be rolled back in case of issues. In SageMaker, you can specify initial traffic distribution by using the InitialVariantWeight API.

Shadow

You can use shadow deployments to safely bring a model to production. In this mode, the new model works alongside an older model or business process, and performs inferences without influencing any decisions. This mode can be useful as a final check or higher fidelity experiment before you promote the model to production.

Shadow mode is useful when you don't need any user inference feedback. You can assess the quality of predictions by performing error analysis and comparing the new model with the old model, and you can monitor the output distribution to verify that it is as expected. To see how to do shadow deployment with SageMaker, see the blog post Deploy shadow ML models in Amazon SageMaker on the AWS Machine Learning blog.

A/B testing

When ML practitioners develop models in their environments, the metrics that they optimize for are often proxies to the business metrics that really matter. This makes it difficult to tell for certain if a new model will actually improve business outcomes, such as revenue and clickthrough rate, and reduce the number of user complaints.

Consider the case of an e-commerce website in which the business goal is to sell as many products as possible. The review team knows that sales and customer satisfaction correlate directly with informative and accurate reviews. A team member might propose a new review ranking algorithm to improve sales. By using A/B testing, they could roll the old and new algorithms out to different but similar user groups, and monitor the results to see whether users who received predictions from the newer model are more likely to make purchases.

A/B testing also helps gauge the business impact of model staleness and drift. Teams can put new models in production with some recurrence, perform A/B testing with each model, and create an age versus performance chart. This would help the team understand the data drift volatility in their production data.

For more information about how to perform A/B testing with SageMaker, see the blog post A/B Testing ML models in production using Amazon SageMaker on the AWS Machine Learning blog.

Consider your inference requirements

With SageMaker, you can choose the underlying infrastructure to deploy your model in different ways. These inference invocation capabilities support different use cases and cost profiles. Your options include real-time inference, asynchronous inference, and batch transform, as discussed in the following sections.

Real-time inference

Real-time inference is ideal for inference workloads where you have real-time, interactive, low-latency requirements. You can deploy your model to SageMaker hosting services and get an endpoint that can be used for inference. These endpoints are fully managed, support automatic scaling (see Automatically scale Amazon SageMaker models), and can be deployed in multiple Availability Zones.

If you have a deep learning model built with Apache MXNet, PyTorch, or TensorFlow, you can also use Amazon SageMaker Elastic Inference (EI). With EI, you can attach fractional GPUs to any SageMaker instance to accelerate inference. You can select the client instance to run your application and attach an EI accelerator to use the correct amount of GPU acceleration for your inference needs.

Another option is to use multi-model endpoints, which provide a scalable and cost-effective solution to deploying large numbers of models. These endpoints use a shared serving container that is enabled to host multiple models. Multi-model endpoints reduce hosting costs by improving endpoint utilization compared with using single-model endpoints. They also reduce deployment overhead, because SageMaker manages loading models in memory and scaling them based on traffic patterns.

For additional best practices for deploying ML models in SageMaker, see Deployment best practices in the SageMaker documentation.

Asynchronous inference

Amazon SageMaker Asynchronous Inference is a capability in SageMaker that queues incoming requests and processes them asynchronously. This option is ideal for requests with large payload sizes up to 1 GB, long processing times, and near real-time latency requirements. Asynchronous inference enables you to save on costs by automatically scaling the instance count to zero when there are no requests to process, so you pay only when your endpoint is processing requests.

Batch transform

Use batch transform when you want to do the following:

- Preprocess datasets to remove noise or bias that interferes with training or inference from your dataset.
- Get inferences from large datasets.
- Run inference when you don't need a persistent endpoint.
- Associate input records with inferences to assist the interpretation of results.

Monitoring

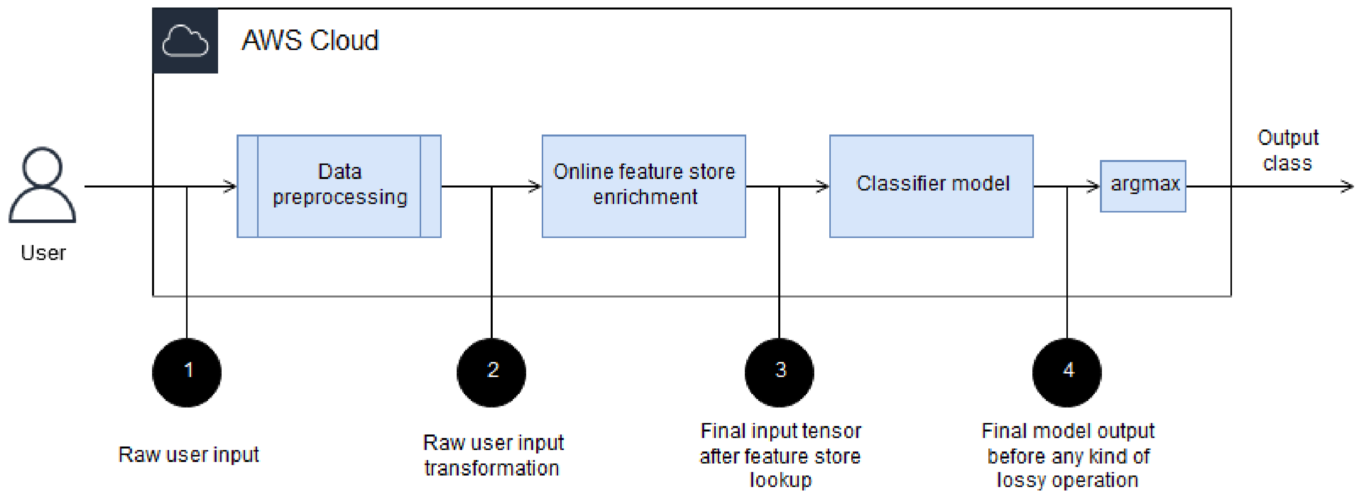
When models are already in production and delivering business value, run continuous checks to identify when models must be retrained or taken action upon.

Your monitoring team should behave proactively, not reactively, to better understand the data behavior of the environment, and to identify the frequency, rate, and abruptness of data drifts. The team should identify new edge cases in the data that might be underrepresented in the training set, validation set, and other edge case slices. They should store quality of service (QoS) metrics, use alarms to immediately take action when an issue arises, and define a strategy to ingest and amend current datasets. These practices start by logging requests and responses for the model, to provide a reference for troubleshooting or additional insights.

Ideally, data transformations should be logged in a few key stages during processing:

- Before any kind of preprocessing
- After any kind of feature store enrichment
- After all main stages of a model
- Before any kind of lossy function on the model output, such as `argmax`

The following diagram illustrates these stages.


AWS Cloud
User
Data preprocessing
Online feature store enrichment
Classifier model
argmax
Output class
1
Raw user input
2
Raw user input transformation
3
Final input tensor after feature store lookup
4
Final model output before any kind of lossy operation

You can use SageMaker Model Monitor to automatically capture input and output data and store it in Amazon Simple Storage Service (Amazon S3). You can implement other types of intermediate logging by adding logs to a custom serving container.

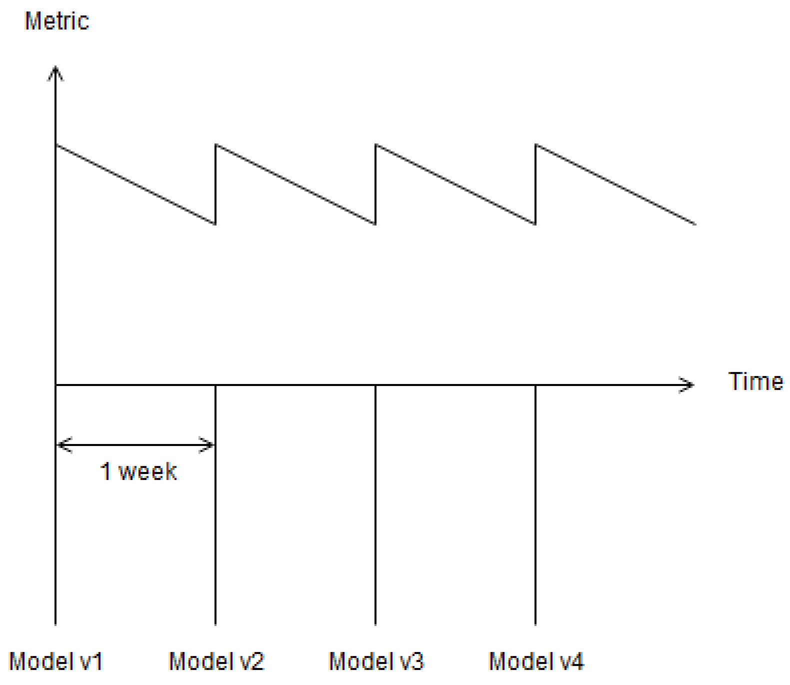
After you log the data from models, you can monitor distribution drift. In some instances, you can get ground truth (data that's correctly labeled) soon after inference. A common example of this is a model that predicts the most relevant ads to display to a user. As soon as the user has left the page you can determine whether they clicked the ad. If the user has clicked the ad, you might log that information. In this simple example, you can easily quantify how successful your model is by using a metric, such as accuracy or F1, that can be measured both in training and in deployment. For more information about these scenarios in which you have labeled data, see Monitor model quality in the SageMaker documentation. However, these simple scenarios are infrequent, because models are often designed to optimize mathematically convenient metrics that are only proxy to actual business outcomes. In such cases, the best practice is to monitor the business outcome when a model is deployed in production.

Consider the case of a review ranking model. If the defined business outcome of the ML model is to display the most relevant and useful reviews at the top of the webpage, you can measure the success of the model by adding a button such as "Was this helpful?" for each review. Measuring the clickthrough rate of this button could be a business outcome measure that helps you measure how well your model is doing in production.

To monitor the drift of the input or output labels in SageMaker, you can use the data quality capabilities of SageMaker Model Monitor, which monitor both the input and output. You can also implement your own logic for SageMaker Model Monitor by building a custom container.

Monitoring the data that a model receives both in development time and in runtime is critical. Engineers should monitor the data not only for schema changes but also for distribution mismatches. Detecting schema changes is easier and can be implemented by a set of rules, but distribution mismatch is often trickier, especially because it requires you to define a threshold to quantify when to raise an alarm. In cases where the monitored distribution is known, often the easiest way is to monitor the distribution's parameters. In the case of a normal distribution, that would be the mean and standard deviation. Other key metrics, such as the percentage of missing values, maximum values, and minimum values, are also useful.

You can also create ongoing monitoring jobs that sample training data and inference data and compare their distributions. You can create these jobs for both model input and model output, and plot the data against time to visualize any sudden or gradual drift. This is illustrated in the following chart.



To better understand the drift profile of the data, such as how often the data distribution significantly changes, at what rate, or how sudden, we recommend that you continuously deploy new model versions and monitor their performance. For example, if your team deploys a new model every week and observes that the model performance significantly improves every time, they can determine that they should deliver new models in less than a week at the minimum.

Next steps and resources

This guide walks you through on a few considerations when planning the lifecycle of the machine learning models you want to bring to production. It discusses challenges and best practices in four areas—data, training, deployment, and monitoring—and includes additional relevant resources.

AWS provides the Well-Architected Framework, which helps cloud architects build secure, high-performing, resilient, and efficient infrastructures for a variety of applications, workloads, and technology domains. For additional reading, see the Machine Learning Lens offered by AWS Well-Architected.

Resources

Amazon SageMaker documentation

- Amazon SageMaker Feature Store
- Feature Store security and access control
- Shapley values
- Amazon SageMaker Debugger
- Amazon SageMaker Pipelines
- Amazon SageMaker default project templates
- SageMaker real-time inference
- Automatically scale Amazon SageMaker models
- Amazon SageMaker asynchronous inference
- SageMaker Model Monitor

AWS developer tools

- AWS CodePipeline
- AWS CodeCommit

AWS blog posts

- Understanding the key capabilities of Amazon SageMaker Feature Store

- Testing data quality at scale with PyDeequ
- Amazon SageMaker Experiments
- Using AWS CodeCommit Pull Requests to request code reviews and discuss code
- Safely deploying and monitoring Amazon SageMaker endpoints with CodePipeline and AWS CodeDeploy
- Deploy shadow ML models in Amazon SageMaker
- A/B Testing ML models in production using Amazon SageMaker

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Initial publication	—	December 20, 2021

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Numbers

7 Rs

Seven common migration strategies for moving applications to the cloud. These strategies build upon the 5 Rs that Gartner identified in 2011 and consist of the following:

- Refactor/re-architect – Move an application and modify its architecture by taking full advantage of cloud-native features to improve agility, performance, and scalability. This typically involves porting the operating system and database. Example: Migrate your on-premises Oracle database to the Amazon Aurora PostgreSQL-Compatible Edition.
- Replatform (lift and reshape) – Move an application to the cloud, and introduce some level of optimization to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Amazon Relational Database Service (Amazon RDS) for Oracle in the AWS Cloud.
- Repurchase (drop and shop) – Switch to a different product, typically by moving from a traditional license to a SaaS model. Example: Migrate your customer relationship management (CRM) system to Salesforce.com.
- Rehost (lift and shift) – Move an application to the cloud without making any changes to take advantage of cloud capabilities. Example: Migrate your on-premises Oracle database to Oracle on an EC2 instance in the AWS Cloud.
- Relocate (hypervisor-level lift and shift) – Move infrastructure to the cloud without purchasing new hardware, rewriting applications, or modifying your existing operations. You migrate servers from an on-premises platform to a cloud service for the same platform. Example: Migrate a Microsoft Hyper-V application to AWS.
- Retain (revisit) – Keep applications in your source environment. These might include applications that require major refactoring, and you want to postpone that work until a later time, and legacy applications that you want to retain, because there's no business justification for migrating them.

- Retire – Decommission or remove applications that are no longer needed in your source environment.

A

ABAC

See attribute-based access control.

abstracted services

See managed services.

ACID

See atomicity, consistency, isolation, durability.

active-active migration

A database migration method in which the source and target databases are kept in sync (by using a bidirectional replication tool or dual write operations), and both databases handle transactions from connecting applications during migration. This method supports migration in small, controlled batches instead of requiring a one-time cutover. It's more flexible but requires more work than active-passive migration.

active-passive migration

A database migration method in which in which the source and target databases are kept in sync, but only the source database handles transactions from connecting applications while data is replicated to the target database. The target database doesn't accept any transactions during migration.

aggregate function

A SQL function that operates on a group of rows and calculates a single return value for the group. Examples of aggregate functions include `SUM` and `MAX`.

AI

See artificial intelligence.

AIOps

See artificial intelligence operations.

anonymization

The process of permanently deleting personal information in a dataset. Anonymization can help protect personal privacy. Anonymized data is no longer considered to be personal data.

anti-pattern

A frequently used solution for a recurring issue where the solution is counter-productive, ineffective, or less effective than an alternative.

application control

A security approach that allows the use of only approved applications in order to help protect a system from malware.

application portfolio

A collection of detailed information about each application used by an organization, including the cost to build and maintain the application, and its business value. This information is key to the portfolio discovery and analysis process and helps identify and prioritize the applications to be migrated, modernized, and optimized.

artificial intelligence (AI)

The field of computer science that is dedicated to using computing technologies to perform cognitive functions that are typically associated with humans, such as learning, solving problems, and recognizing patterns. For more information, see What is Artificial Intelligence?

artificial intelligence operations (AIOps)

The process of using machine learning techniques to solve operational problems, reduce operational incidents and human intervention, and increase service quality. For more information about how AIOps is used in the AWS migration strategy, see the operations integration guide.

asymmetric encryption

An encryption algorithm that uses a pair of keys, a public key for encryption and a private key for decryption. You can share the public key because it isn't used for decryption, but access to the private key should be highly restricted.

atomicity, consistency, isolation, durability (ACID)

A set of software properties that guarantee the data validity and operational reliability of a database, even in the case of errors, power failures, or other problems.

attribute-based access control (ABAC)

The practice of creating fine-grained permissions based on user attributes, such as department, job role, and team name. For more information, see ABAC for AWS in the AWS Identity and Access Management (IAM) documentation.

authoritative data source

A location where you store the primary version of data, which is considered to be the most reliable source of information. You can copy data from the authoritative data source to other locations for the purposes of processing or modifying the data, such as anonymizing, redacting, or pseudonymizing it.

Availability Zone

A distinct location within an AWS Region that is insulated from failures in other Availability Zones and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.

AWS Cloud Adoption Framework (AWS CAF)

A framework of guidelines and best practices from AWS to help organizations develop an efficient and effective plan to move successfully to the cloud. AWS CAF organizes guidance into six focus areas called perspectives: business, people, governance, platform, security, and operations. The business, people, and governance perspectives focus on business skills and processes; the platform, security, and operations perspectives focus on technical skills and processes. For example, the people perspective targets stakeholders who handle human resources (HR), staffing functions, and people management. For this perspective, AWS CAF provides guidance for people development, training, and communications to help ready the organization for successful cloud adoption. For more information, see the AWS CAF website and the AWS CAF whitepaper.

AWS Workload Qualification Framework (AWS WQF)

A tool that evaluates database migration workloads, recommends migration strategies, and provides work estimates. AWS WQF is included with AWS Schema Conversion Tool (AWS SCT). It analyzes database schemas and code objects, application code, dependencies, and performance characteristics, and provides assessment reports.

B

bad bot

A bot that is intended to disrupt or cause harm to individuals or organizations.

BCP

See business continuity planning.

behavior graph

A unified, interactive view of resource behavior and interactions over time. You can use a behavior graph with Amazon Detective to examine failed logon attempts, suspicious API calls, and similar actions. For more information, see Data in a behavior graph in the Detective documentation.

big-endian system

A system that stores the most significant byte first. See also endianness.

binary classification

A process that predicts a binary outcome (one of two possible classes). For example, your ML model might need to predict problems such as “Is this email spam or not spam?" or "Is this product a book or a car?"

bloom filter

A probabilistic, memory-efficient data structure that is used to test whether an element is a member of a set.

blue/green deployment

A deployment strategy where you create two separate but identical environments. You run the current application version in one environment (blue) and the new application version in the other environment (green). This strategy helps you quickly roll back with minimal impact.

bot

A software application that runs automated tasks over the internet and simulates human activity or interaction. Some bots are useful or beneficial, such as web crawlers that index information on the internet. Some other bots, known as *bad bots*, are intended to disrupt or cause harm to individuals or organizations.

botnet

Networks of bots that are infected by malware and are under the control of a single party, known as a *bot herder* or *bot operator*. Botnets are the best-known mechanism to scale bots and their impact.

branch

A contained area of a code repository. The first branch created in a repository is the *main branch*. You can create a new branch from an existing branch, and you can then develop features or fix bugs in the new branch. A branch you create to build a feature is commonly referred to as a *feature branch*. When the feature is ready for release, you merge the feature branch back into the main branch. For more information, see About branches (GitHub documentation).

break-glass access

In exceptional circumstances and through an approved process, a quick means for a user to gain access to an AWS account that they don't typically have permissions to access. For more information, see the Implement break-glass procedures indicator in the AWS Well-Architected guidance.

brownfield strategy

The existing infrastructure in your environment. When adopting a brownfield strategy for a system architecture, you design the architecture around the constraints of the current systems and infrastructure. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

buffer cache

The memory area where the most frequently accessed data is stored.

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.

business continuity planning (BCP)

A plan that addresses the potential impact of a disruptive event, such as a large-scale migration, on operations and enables a business to resume operations quickly.

C

CAF

See AWS Cloud Adoption Framework.

canary deployment

The slow and incremental release of a version to end users. When you are confident, you deploy the new version and replace the current version in its entirety.

CCoE

See Cloud Center of Excellence.

CDC

See change data capture.

change data capture (CDC)

The process of tracking changes to a data source, such as a database table, and recording metadata about the change. You can use CDC for various purposes, such as auditing or replicating changes in a target system to maintain synchronization.

chaos engineering

Intentionally introducing failures or disruptive events to test a system's resilience. You can use AWS Fault Injection Service (AWS FIS) to perform experiments that stress your AWS workloads and evaluate their response.

CI/CD

See continuous integration and continuous delivery.

classification

A categorization process that helps generate predictions. ML models for classification problems predict a discrete value. Discrete values are always distinct from one another. For example, a model might need to evaluate whether or not there is a car in an image.

client-side encryption

Encryption of data locally, before the target AWS service receives it.

Cloud Center of Excellence (CCoE)

A multi-disciplinary team that drives cloud adoption efforts across an organization, including developing cloud best practices, mobilizing resources, establishing migration timelines, and leading the organization through large-scale transformations. For more information, see the CCoE posts on the AWS Cloud Enterprise Strategy Blog.

cloud computing

The cloud technology that is typically used for remote data storage and IoT device management. Cloud computing is commonly connected to edge computing technology.

cloud operating model

In an IT organization, the operating model that is used to build, mature, and optimize one or more cloud environments. For more information, see Building your Cloud Operating Model.

cloud stages of adoption

The four phases that organizations typically go through when they migrate to the AWS Cloud:

- Project – Running a few cloud-related projects for proof of concept and learning purposes
- Foundation – Making foundational investments to scale your cloud adoption (e.g., creating a landing zone, defining a CCoE, establishing an operations model)
- Migration – Migrating individual applications
- Re-invention – Optimizing products and services, and innovating in the cloud

These stages were defined by Stephen Orban in the blog post The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy blog. For information about how they relate to the AWS migration strategy, see the migration readiness guide.

CMDB

See configuration management database.

code repository

A location where source code and other assets, such as documentation, samples, and scripts, are stored and updated through version control processes. Common cloud repositories include GitHub or AWS CodeCommit. Each version of the code is called a *branch*. In a microservice structure, each repository is devoted to a single piece of functionality. A single CI/CD pipeline can use multiple repositories.

cold cache

A buffer cache that is empty, not well populated, or contains stale or irrelevant data. This affects performance because the database instance must read from the main memory or disk, which is slower than reading from the buffer cache.

cold data

Data that is rarely accessed and is typically historical. When querying this kind of data, slow queries are typically acceptable. Moving this data to lower-performing and less expensive storage tiers or classes can reduce costs.

computer vision (CV)

A field of AI that uses machine learning to analyze and extract information from visual formats such as digital images and videos. For example, AWS Panorama offers devices that add CV to on-premises camera networks, and Amazon SageMaker provides image processing algorithms for CV.

configuration drift

For a workload, a configuration change from the expected state. It might cause the workload to become noncompliant, and it's typically gradual and unintentional.

configuration management database (CMDB)

A repository that stores and manages information about a database and its IT environment, including both hardware and software components and their configurations. You typically use data from a CMDB in the portfolio discovery and analysis stage of migration.

conformance pack

A collection of AWS Config rules and remediation actions that you can assemble to customize your compliance and security checks. You can deploy a conformance pack as a single entity in an AWS account and Region, or across an organization, by using a YAML template. For more information, see Conformance packs in the AWS Config documentation.

continuous integration and continuous delivery (CI/CD)

The process of automating the source, build, test, staging, and production stages of the software release process. CI/CD is commonly described as a pipeline. CI/CD can help you automate processes, improve productivity, improve code quality, and deliver faster. For more information, see Benefits of continuous delivery. CD can also stand for *continuous deployment*. For more information, see Continuous Delivery vs. Continuous Deployment.

CV

See computer vision.

D

data at rest

Data that is stationary in your network, such as data that is in storage.

data classification

A process for identifying and categorizing the data in your network based on its criticality and sensitivity. It is a critical component of any cybersecurity risk management strategy because it helps you determine the appropriate protection and retention controls for the data. Data classification is a component of the security pillar in the AWS Well-Architected Framework. For more information, see Data classification.

data drift

A meaningful variation between the production data and the data that was used to train an ML model, or a meaningful change in the input data over time. Data drift can reduce the overall quality, accuracy, and fairness in ML model predictions.

data in transit

Data that is actively moving through your network, such as between network resources.

data mesh

An architectural framework that provides distributed, decentralized data ownership with centralized management and governance.

data minimization

The principle of collecting and processing only the data that is strictly necessary. Practicing data minimization in the AWS Cloud can reduce privacy risks, costs, and your analytics carbon footprint.

data perimeter

A set of preventive guardrails in your AWS environment that help make sure that only trusted identities are accessing trusted resources from expected networks. For more information, see Building a data perimeter on AWS.

data preprocessing

To transform raw data into a format that is easily parsed by your ML model. Preprocessing data can mean removing certain columns or rows and addressing missing, inconsistent, or duplicate values.

data provenance

The process of tracking the origin and history of data throughout its lifecycle, such as how the data was generated, transmitted, and stored.

data subject

An individual whose data is being collected and processed.

data warehouse

A data management system that supports business intelligence, such as analytics. Data warehouses commonly contain large amounts of historical data, and they are typically used for queries and analysis.

database definition language (DDL)

Statements or commands for creating or modifying the structure of tables and objects in a database.

database manipulation language (DML)

Statements or commands for modifying (inserting, updating, and deleting) information in a database.

DDL

See database definition language.

deep ensemble

To combine multiple deep learning models for prediction. You can use deep ensembles to obtain a more accurate prediction or for estimating uncertainty in predictions.

deep learning

An ML subfield that uses multiple layers of artificial neural networks to identify mapping between input data and target variables of interest.

defense-in-depth

An information security approach in which a series of security mechanisms and controls are thoughtfully layered throughout a computer network to protect the confidentiality, integrity, and availability of the network and the data within. When you adopt this strategy on AWS, you add multiple controls at different layers of the AWS Organizations structure to help secure resources. For example, a defense-in-depth approach might combine multi-factor authentication, network segmentation, and encryption.

delegated administrator

In AWS Organizations, a compatible service can register an AWS member account to administer the organization's accounts and manage permissions for that service. This account is called the *delegated administrator* for that service. For more information and a list of compatible services, see Services that work with AWS Organizations in the AWS Organizations documentation.

deployment

The process of making an application, new features, or code fixes available in the target environment. Deployment involves implementing changes in a code base and then building and running that code base in the application's environments.

development environment

See environment.

detective control

A security control that is designed to detect, log, and alert after an event has occurred. These controls are a second line of defense, alerting you to security events that bypassed the preventative controls in place. For more information, see Detective controls in *Implementing security controls on AWS*.

development value stream mapping (DVSM)

A process used to identify and prioritize constraints that adversely affect speed and quality in a software development lifecycle. DVSM extends the value stream mapping process originally designed for lean manufacturing practices. It focuses on the steps and teams required to create and move value through the software development process.

digital twin

A virtual representation of a real-world system, such as a building, factory, industrial equipment, or production line. Digital twins support predictive maintenance, remote monitoring, and production optimization.

dimension table

In a star schema, a smaller table that contains data attributes about quantitative data in a fact table. Dimension table attributes are typically text fields or discrete numbers that behave like text. These attributes are commonly used for query constraining, filtering, and result set labeling.

disaster

An event that prevents a workload or system from fulfilling its business objectives in its primary deployed location. These events can be natural disasters, technical failures, or the result of human actions, such as unintentional misconfiguration or a malware attack.

disaster recovery (DR)

The strategy and process you use to minimize downtime and data loss caused by a disaster. For more information, see Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML

See database manipulation language.

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

DR

See disaster recovery.

drift detection

Tracking deviations from a baselined configuration. For example, you can use AWS CloudFormation to detect drift in system resources, or you can use AWS Control Tower to detect changes in your landing zone that might affect compliance with governance requirements.

DVSM

See development value stream mapping.

E

EDA

See exploratory data analysis.

edge computing

The technology that increases the computing power for smart devices at the edges of an IoT network. When compared with cloud computing, edge computing can reduce communication latency and improve response time.

encryption

A computing process that transforms plaintext data, which is human-readable, into ciphertext.

encryption key

A cryptographic string of randomized bits that is generated by an encryption algorithm. Keys can vary in length, and each key is designed to be unpredictable and unique.

endianness

The order in which bytes are stored in computer memory. Big-endian systems store the most significant byte first. Little-endian systems store the least significant byte first.

endpoint

See service endpoint.

endpoint service

A service that you can host in a virtual private cloud (VPC) to share with other users. You can create an endpoint service with AWS PrivateLink and grant permissions to other AWS accounts or to AWS Identity and Access Management (IAM) principals. These accounts or principals can connect to your endpoint service privately by creating interface VPC endpoints. For more information, see Create an endpoint service in the Amazon Virtual Private Cloud (Amazon VPC) documentation.

enterprise resource planning (ERP)

A system that automates and manages key business processes (such as accounting, MES, and project management) for an enterprise.

envelope encryption

The process of encrypting an encryption key with another encryption key. For more information, see Envelope encryption in the AWS Key Management Service (AWS KMS) documentation.

environment

An instance of a running application. The following are common types of environments in cloud computing:

- development environment – An instance of a running application that is available only to the core team responsible for maintaining the application. Development environments are used to test changes before promoting them to upper environments. This type of environment is sometimes referred to as a *test environment*.
- lower environments – All development environments for an application, such as those used for initial builds and tests.
- production environment – An instance of a running application that end users can access. In a CI/CD pipeline, the production environment is the last deployment environment.
- upper environments – All environments that can be accessed by users other than the core development team. This can include a production environment, preproduction environments, and environments for user acceptance testing.

epic

In agile methodologies, functional categories that help organize and prioritize your work. Epics provide a high-level description of requirements and implementation tasks. For example, AWS CAF security epics include identity and access management, detective controls, infrastructure security, data protection, and incident response. For more information about epics in the AWS migration strategy, see the program implementation guide.

ERP

See enterprise resource planning.

exploratory data analysis (EDA)

The process of analyzing a dataset to understand its main characteristics. You collect or aggregate data and then perform initial investigations to find patterns, detect anomalies, and check assumptions. EDA is performed by calculating summary statistics and creating data visualizations.

F

fact table

The central table in a star schema. It stores quantitative data about business operations. Typically, a fact table contains two types of columns: those that contain measures and those that contain a foreign key to a dimension table.

fail fast

A philosophy that uses frequent and incremental testing to reduce the development lifecycle. It is a critical part of an agile approach.

fault isolation boundary

In the AWS Cloud, a boundary such as an Availability Zone, AWS Region, control plane, or data plane that limits the effect of a failure and helps improve the resilience of workloads. For more information, see AWS Fault Isolation Boundaries.

feature branch

See branch.

features

The input data that you use to make a prediction. For example, in a manufacturing context, features could be images that are periodically captured from the manufacturing line.

feature importance

How significant a feature is for a model's predictions. This is usually expressed as a numerical score that can be calculated through various techniques, such as Shapley Additive Explanations (SHAP) and integrated gradients. For more information, see Machine learning model interpretability with :AWS.

feature transformation

To optimize data for the ML process, including enriching data with additional sources, scaling values, or extracting multiple sets of information from a single data field. This enables the ML model to benefit from the data. For example, if you break down the "2021-05-27 00:15:37" date into "2021", "May", "Thu", and "15", you can help the learning algorithm learn nuanced patterns associated with different data components.

FGAC

See fine-grained access control.

fine-grained access control (FGAC)

The use of multiple conditions to allow or deny an access request.

flash-cut migration

A database migration method that uses continuous data replication through change data capture to migrate data in the shortest time possible, instead of using a phased approach. The objective is to keep downtime to a minimum.

G

geo blocking

See geographic restrictions.

geographic restrictions (geo blocking)

In Amazon CloudFront, an option to prevent users in specific countries from accessing content distributions. You can use an allow list or block list to specify approved and banned countries. For more information, see Restricting the geographic distribution of your content in the CloudFront documentation.

Gitflow workflow

An approach in which lower and upper environments use different branches in a source code repository. The Gitflow workflow is considered legacy, and the trunk-based workflow is the modern, preferred approach.

greenfield strategy

The absence of existing infrastructure in a new environment. When adopting a greenfield strategy for a system architecture, you can select all new technologies without the restriction of compatibility with existing infrastructure, also known as brownfield. If you are expanding the existing infrastructure, you might blend brownfield and greenfield strategies.

guardrail

A high-level rule that helps govern resources, policies, and compliance across organizational units (OUs). *Preventive guardrails* enforce policies to ensure alignment to compliance standards. They are implemented by using service control policies and IAM permissions boundaries. *Detective guardrails* detect policy violations and compliance issues, and generate alerts for remediation. They are implemented by using AWS Config, AWS Security Hub, Amazon GuardDuty, AWS Trusted Advisor, Amazon Inspector, and custom AWS Lambda checks.

H

HA

See high availability.

heterogeneous database migration

Migrating your source database to a target database that uses a different database engine (for example, Oracle to Amazon Aurora). Heterogeneous migration is typically part of a re-architecting effort, and converting the schema can be a complex task. AWS provides AWS SCT that helps with schema conversions.

high availability (HA)

The ability of a workload to operate continuously, without intervention, in the event of challenges or disasters. HA systems are designed to automatically fail over, consistently deliver high-quality performance, and handle different loads and failures with minimal performance impact.

historian modernization

An approach used to modernize and upgrade operational technology (OT) systems to better serve the needs of the manufacturing industry. A *historian* is a type of database that is used to collect and store data from various sources in a factory.

homogeneous database migration

Migrating your source database to a target database that shares the same database engine (for example, Microsoft SQL Server to Amazon RDS for SQL Server). Homogeneous migration is typically part of a rehosting or replatforming effort. You can use native database utilities to migrate the schema.

hot data

Data that is frequently accessed, such as real-time data or recent translational data. This data typically requires a high-performance storage tier or class to provide fast query responses.

hotfix

An urgent fix for a critical issue in a production environment. Due to its urgency, a hotfix is usually made outside of the typical DevOps release workflow.

hypercare period

Immediately following cutover, the period of time when a migration team manages and monitors the migrated applications in the cloud in order to address any issues. Typically, this period is 1–4 days in length. At the end of the hypercare period, the migration team typically transfers responsibility for the applications to the cloud operations team.

I

IaC

See infrastructure as code.

identity-based policy

A policy attached to one or more IAM principals that defines their permissions within the AWS Cloud environment.

idle application

An application that has an average CPU and memory usage between 5 and 20 percent over a period of 90 days. In a migration project, it is common to retire these applications or retain them on premises.

IIoT

See industrial Internet of Things.

immutable infrastructure

A model that deploys new infrastructure for production workloads instead of updating, patching, or modifying the existing infrastructure. Immutable infrastructures are inherently more consistent, reliable, and predictable than mutable infrastructure. For more information, see the Deploy using immutable infrastructure best practice in the AWS Well-Architected Framework.

inbound (ingress) VPC

In an AWS multi-account architecture, a VPC that accepts, inspects, and routes network connections from outside an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

incremental migration

A cutover strategy in which you migrate your application in small parts instead of performing a single, full cutover. For example, you might move only a few microservices or users to the new system initially. After you verify that everything is working properly, you can incrementally move additional microservices or users until you can decommission your legacy system. This strategy reduces the risks associated with large migrations.

Industry 4.0

A term that was introduced by Klaus Schwab in 2016 to refer to the modernization of manufacturing processes through advances in connectivity, real-time data, automation, analytics, and AI/ML.

infrastructure

All of the resources and assets contained within an application's environment.

infrastructure as code (IaC)

The process of provisioning and managing an application's infrastructure through a set of configuration files. IaC is designed to help you centralize infrastructure management, standardize resources, and scale quickly so that new environments are repeatable, reliable, and consistent.

industrial Internet of Things (IIoT)

The use of internet-connected sensors and devices in the industrial sectors, such as manufacturing, energy, automotive, healthcare, life sciences, and agriculture. For more information, see Building an industrial Internet of Things (IIoT) digital transformation strategy.

inspection VPC

In an AWS multi-account architecture, a centralized VPC that manages inspections of network traffic between VPCs (in the same or different AWS Regions), the internet, and on-premises networks. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

Internet of Things (IoT)

The network of connected physical objects with embedded sensors or processors that communicate with other devices and systems through the internet or over a local communication network. For more information, see What is IoT?

interpretability

A characteristic of a machine learning model that describes the degree to which a human can understand how the model's predictions depend on its inputs. For more information, see Machine learning model interpretability with AWS.

IoT

See Internet of Things.

IT information library (ITIL)

A set of best practices for delivering IT services and aligning these services with business requirements. ITIL provides the foundation for ITSM.

IT service management (ITSM)

Activities associated with designing, implementing, managing, and supporting IT services for an organization. For information about integrating cloud operations with ITSM tools, see the operations integration guide.

ITIL

See IT information library.

ITSM

See IT service management.

L

label-based access control (LBAC)

An implementation of mandatory access control (MAC) where the users and the data itself are each explicitly assigned a security label value. The intersection between the user security label and data security label determines which rows and columns can be seen by the user.

landing zone

A landing zone is a well-architected, multi-account AWS environment that is scalable and secure. This is a starting point from which your organizations can quickly launch and deploy workloads and applications with confidence in their security and infrastructure environment. For more information about landing zones, see Setting up a secure and scalable multi-account AWS environment.

large migration

A migration of 300 or more servers.

LBAC

See label-based access control.

least privilege

The security best practice of granting the minimum permissions required to perform a task. For more information, see Apply least-privilege permissions in the IAM documentation.

lift and shift

See 7 Rs.

little-endian system

A system that stores the least significant byte first. See also endianness.

lower environments

See environment.

M

machine learning (ML)

A type of artificial intelligence that uses algorithms and techniques for pattern recognition and learning. ML analyzes and learns from recorded data, such as Internet of Things (IoT) data, to generate a statistical model based on patterns. For more information, see Machine Learning.

main branch

See branch.

malware

Software that is designed to compromise computer security or privacy. Malware might disrupt computer systems, leak sensitive information, or gain unauthorized access. Examples of malware include viruses, worms, ransomware, Trojan horses, spyware, and keyloggers.

managed services

AWS services for which AWS operates the infrastructure layer, the operating system, and platforms, and you access the endpoints to store and retrieve data. Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB are examples of managed services. These are also known as *abstracted services*.

manufacturing execution system (MES)

A software system for tracking, monitoring, documenting, and controlling production processes that convert raw materials to finished products on the shop floor.

MAP

See Migration Acceleration Program.

mechanism

A complete process in which you create a tool, drive adoption of the tool, and then inspect the results in order to make adjustments. A mechanism is a cycle that reinforces and improves itself as it operates. For more information, see Building mechanisms in the AWS Well-Architected Framework.

member account

All AWS accounts other than the management account that are part of an organization in AWS Organizations. An account can be a member of only one organization at a time.

MES

See manufacturing execution system.

Message Queuing Telemetry Transport (MQTT)

A lightweight, machine-to-machine (M2M) communication protocol, based on the publish/subscribe pattern, for resource-constrained IoT devices.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.

Migration Acceleration Program (MAP)

An AWS program that provides consulting support, training, and services to help organizations build a strong operational foundation for moving to the cloud, and to help offset the initial cost of migrations. MAP includes a migration methodology for executing legacy migrations in a methodical way and a set of tools to automate and accelerate common migration scenarios.

migration at scale

The process of moving the majority of the application portfolio to the cloud in waves, with more applications moved at a faster rate in each wave. This phase uses the best practices and lessons learned from the earlier phases to implement a *migration factory* of teams, tools, and processes to streamline the migration of workloads through automation and agile delivery. This is the third phase of the AWS migration strategy.

migration factory

Cross-functional teams that streamline the migration of workloads through automated, agile approaches. Migration factory teams typically include operations, business analysts and owners, migration engineers, developers, and DevOps professionals working in sprints. Between 20 and 50 percent of an enterprise application portfolio consists of repeated patterns that can be optimized by a factory approach. For more information, see the discussion of migration factories and the Cloud Migration Factory guide in this content set.

migration metadata

The information about the application and server that is needed to complete the migration. Each migration pattern requires a different set of migration metadata. Examples of migration metadata include the target subnet, security group, and AWS account.

migration pattern

A repeatable migration task that details the migration strategy, the migration destination, and the migration application or service used. Example: Rehost migration to Amazon EC2 with AWS Application Migration Service.

Migration Portfolio Assessment (MPA)

An online tool that provides information for validating the business case for migrating to the AWS Cloud. MPA provides detailed portfolio assessment (server right-sizing, pricing, TCO comparisons, migration cost analysis) as well as migration planning (application data analysis and data collection, application grouping, migration prioritization, and wave planning). The MPA tool (requires login) is available free of charge to all AWS consultants and APN Partner consultants.

Migration Readiness Assessment (MRA)

The process of gaining insights about an organization's cloud readiness status, identifying strengths and weaknesses, and building an action plan to close identified gaps, using the AWS CAF. For more information, see the migration readiness guide. MRA is the first phase of the AWS migration strategy.

migration strategy

The approach used to migrate a workload to the AWS Cloud. For more information, see the 7 Rs entry in this glossary and see Mobilize your organization to accelerate large-scale migrations.

ML

See machine learning.

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.

monolithic applications (monoliths)

Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application’s features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.

MPA

See Migration Portfolio Assessment.

MQTT

See Message Queuing Telemetry Transport.

multiclass classification

A process that helps generate predictions for multiple classes (predicting one of more than two outcomes). For example, an ML model might ask "Is this product a book, car, or phone?" or "Which product category is most interesting to this customer?"

mutable infrastructure

A model that updates and modifies the existing infrastructure for production workloads. For improved consistency, reliability, and predictability, the AWS Well-Architected Framework recommends the use of immutable infrastructure as a best practice.

O

OAC

See origin access control.

OAI

See origin access identity.

OCM

See organizational change management.

offline migration

A migration method in which the source workload is taken down during the migration process. This method involves extended downtime and is typically used for small, non-critical workloads.

OI

See operations integration.

OLA

See operational-level agreement.

online migration

A migration method in which the source workload is copied to the target system without being taken offline. Applications that are connected to the workload can continue to function during the migration. This method involves zero to minimal downtime and is typically used for critical production workloads.

OPC-UA

See Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

A machine-to-machine (M2M) communication protocol for industrial automation. OPC-UA provides an interoperability standard with data encryption, authentication, and authorization schemes.

operational-level agreement (OLA)

An agreement that clarifies what functional IT groups promise to deliver to each other, to support a service-level agreement (SLA).

operational readiness review (ORR)

A checklist of questions and associated best practices that help you understand, evaluate, prevent, or reduce the scope of incidents and possible failures. For more information, see Operational Readiness Reviews (ORR) in the AWS Well-Architected Framework.

operational technology (OT)

Hardware and software systems that work with the physical environment to control industrial operations, equipment, and infrastructure. In manufacturing, the integration of OT and information technology (IT) systems is a key focus for Industry 4.0 transformations.

operations integration (OI)

The process of modernizing operations in the cloud, which involves readiness planning, automation, and integration. For more information, see the operations integration guide.

organization trail

A trail that's created by AWS CloudTrail that logs all events for all AWS accounts in an organization in AWS Organizations. This trail is created in each AWS account that's part of the organization and tracks the activity in each account. For more information, see Creating a trail for an organization in the CloudTrail documentation.

organizational change management (OCM)

A framework for managing major, disruptive business transformations from a people, culture, and leadership perspective. OCM helps organizations prepare for, and transition to, new systems and strategies by accelerating change adoption, addressing transitional issues, and driving cultural and organizational changes. In the AWS migration strategy, this framework is called *people acceleration*, because of the speed of change required in cloud adoption projects. For more information, see the OCM guide.

origin access control (OAC)

In CloudFront, an enhanced option for restricting access to secure your Amazon Simple Storage Service (Amazon S3) content. OAC supports all S3 buckets in all AWS Regions, server-side encryption with AWS KMS (SSE-KMS), and dynamic `PUT` and `DELETE` requests to the S3 bucket.

origin access identity (OAI)

In CloudFront, an option for restricting access to secure your Amazon S3 content. When you use OAI, CloudFront creates a principal that Amazon S3 can authenticate with. Authenticated principals can access content in an S3 bucket only through a specific CloudFront distribution. See also OAC, which provides more granular and enhanced access control.

ORR

See operational readiness review.

OT

See operational technology.

outbound (egress) VPC

In an AWS multi-account architecture, a VPC that handles network connections that are initiated from within an application. The AWS Security Reference Architecture recommends setting up your Network account with inbound, outbound, and inspection VPCs to protect the two-way interface between your application and the broader internet.

P

permissions boundary

An IAM management policy that is attached to IAM principals to set the maximum permissions that the user or role can have. For more information, see Permissions boundaries in the IAM documentation.

personally identifiable information (PII)

Information that, when viewed directly or paired with other related data, can be used to reasonably infer the identity of an individual. Examples of PII include names, addresses, and contact information.

PII

See personally identifiable information.

playbook

A set of predefined steps that capture the work associated with migrations, such as delivering core operations functions in the cloud. A playbook can take the form of scripts, automated runbooks, or a summary of processes or steps required to operate your modernized environment.

PLC

See programmable logic controller.

PLM

See product lifecycle management.

policy

An object that can define permissions (see identity-based policy), specify access conditions (see resource-based policy), or define the maximum permissions for all accounts in an organization in AWS Organizations (see service control policy).

polyglot persistence

Independently choosing a microservice’s data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.

portfolio assessment

A process of discovering, analyzing, and prioritizing the application portfolio in order to plan the migration. For more information, see Evaluating migration readiness.

predicate

A query condition that returns `true` or `false`, commonly located in a `WHERE` clause.

predicate pushdown

A database query optimization technique that filters the data in the query before transfer. This reduces the amount of data that must be retrieved and processed from the relational database, and it improves query performance.

preventative control

A security control that is designed to prevent an event from occurring. These controls are a first line of defense to help prevent unauthorized access or unwanted changes to your network. For more information, see Preventative controls in *Implementing security controls on AWS*.

principal

An entity in AWS that can perform actions and access resources. This entity is typically a root user for an AWS account, an IAM role, or a user. For more information, see *Principal* in Roles terms and concepts in the IAM documentation.

Privacy by Design

An approach in system engineering that takes privacy into account throughout the whole engineering process.

private hosted zones

A container that holds information about how you want Amazon Route 53 to respond to DNS queries for a domain and its subdomains within one or more VPCs. For more information, see Working with private hosted zones in the Route 53 documentation.

proactive control

A security control designed to prevent the deployment of noncompliant resources. These controls scan resources before they are provisioned. If the resource is not compliant with the control, then it isn't provisioned. For more information, see the Controls reference guide in the AWS Control Tower documentation and see Proactive controls in *Implementing security controls on AWS*.

product lifecycle management (PLM)

The management of data and processes for a product throughout its entire lifecycle, from design, development, and launch, through growth and maturity, to decline and removal.

production environment

See environment.

programmable logic controller (PLC)

In manufacturing, a highly reliable, adaptable computer that monitors machines and automates manufacturing processes.

pseudonymization

The process of replacing personal identifiers in a dataset with placeholder values. Pseudonymization can help protect personal privacy. Pseudonymized data is still considered to be personal data.

publish/subscribe (pub/sub)

A pattern that enables asynchronous communications among microservices to improve scalability and responsiveness. For example, in a microservices-based MES, a microservice can publish event messages to a channel that other microservices can subscribe to. The system can add new microservices without changing the publishing service.

Q

query plan

A series of steps, like instructions, that are used to access the data in a SQL relational database system.

query plan regression

When a database service optimizer chooses a less optimal plan than it did before a given change to the database environment. This can be caused by changes to statistics, constraints, environment settings, query parameter bindings, and updates to the database engine.

R

RACI matrix

See responsible, accountable, consulted, informed (RACI).

ransomware

A malicious software that is designed to block access to a computer system or data until a payment is made.

RASCI matrix

See responsible, accountable, consulted, informed (RACI).

RCAC

See row and column access control.

read replica

A copy of a database that's used for read-only purposes. You can route queries to the read replica to reduce the load on your primary database.

re-architect

See 7 Rs.

recovery point objective (RPO)

The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

recovery time objective (RTO)

The maximum acceptable delay between the interruption of service and restoration of service.

refactor

See 7 Rs.

Region

A collection of AWS resources in a geographic area. Each AWS Region is isolated and independent of the others to provide fault tolerance, stability, and resilience. For more information, see Specify which AWS Regions your account can use.

regression

An ML technique that predicts a numeric value. For example, to solve the problem of "What price will this house sell for?" an ML model could use a linear regression model to predict a house's sale price based on known facts about the house (for example, the square footage).

rehost

See 7 Rs.

release

In a deployment process, the act of promoting changes to a production environment.

relocate

See 7 Rs.

replatform

See 7 Rs.

repurchase

See 7 Rs.

resiliency

An application's ability to resist or recover from disruptions. High availability and disaster recovery are common considerations when planning for resiliency in the AWS Cloud. For more information, see AWS Cloud Resilience.

resource-based policy

A policy attached to a resource, such as an Amazon S3 bucket, an endpoint, or an encryption key. This type of policy specifies which principals are allowed access, supported actions, and any other conditions that must be met.

responsible, accountable, consulted, informed (RACI) matrix

A matrix that defines the roles and responsibilities for all parties involved in migration activities and cloud operations. The matrix name is derived from the responsibility types defined in the matrix: responsible (R), accountable (A), consulted (C), and informed (I). The support (S) type is optional. If you include support, the matrix is called a *RASCI matrix*, and if you exclude it, it's called a *RACI matrix*.

responsive control

A security control that is designed to drive remediation of adverse events or deviations from your security baseline. For more information, see Responsive controls in *Implementing security controls on AWS*.

retain

See 7 Rs.

retire

See 7 Rs.

rotation

The process of periodically updating a secret to make it more difficult for an attacker to access the credentials.

row and column access control (RCAC)

The use of basic, flexible SQL expressions that have defined access rules. RCAC consists of row permissions and column masks.

RPO

See recovery point objective.

RTO

See recovery time objective.

runbook

A set of manual or automated procedures required to perform a specific task. These are typically built to streamline repetitive operations or procedures with high error rates.

S

SAML 2.0

An open standard that many identity providers (IdPs) use. This feature enables federated single sign-on (SSO), so users can log into the AWS Management Console or call the AWS API operations without you having to create user in IAM for everyone in your organization. For more information about SAML 2.0-based federation, see About SAML 2.0-based federation in the IAM documentation.

SCADA

See supervisory control and data acquisition.

SCP

See service control policy.

secret

In AWS Secrets Manager, confidential or restricted information, such as a password or user credentials, that you store in encrypted form. It consists of the secret value and its metadata. The secret value can be binary, a single string, or multiple strings. For more information, see What's in a Secrets Manager secret? in the Secrets Manager documentation.

security control

A technical or administrative guardrail that prevents, detects, or reduces the ability of a threat actor to exploit a security vulnerability. There are four primary types of security controls: preventative, detective, responsive, and proactive.

security hardening

The process of reducing the attack surface to make it more resistant to attacks. This can include actions such as removing resources that are no longer needed, implementing the security best practice of granting least privilege, or deactivating unnecessary features in configuration files.

security information and event management (SIEM) system

Tools and services that combine security information management (SIM) and security event management (SEM) systems. A SIEM system collects, monitors, and analyzes data from servers, networks, devices, and other sources to detect threats and security breaches, and to generate alerts.

security response automation

A predefined and programmed action that is designed to automatically respond to or remediate a security event. These automations serve as detective or responsive security controls that help you implement AWS security best practices. Examples of automated response actions include modifying a VPC security group, patching an Amazon EC2 instance, or rotating credentials.

server-side encryption

Encryption of data at its destination, by the AWS service that receives it.

service control policy (SCP)

A policy that provides centralized control over permissions for all accounts in an organization in AWS Organizations. SCPs define guardrails or set limits on actions that an administrator can delegate to users or roles. You can use SCPs as allow lists or deny lists, to specify which services or actions are permitted or prohibited. For more information, see Service control policies in the AWS Organizations documentation.

service endpoint

The URL of the entry point for an AWS service. You can use the endpoint to connect programmatically to the target service. For more information, see AWS service endpoints in *AWS General Reference*.

service-level agreement (SLA)

An agreement that clarifies what an IT team promises to deliver to their customers, such as service uptime and performance.

service-level indicator (SLI)

A measurement of a performance aspect of a service, such as its error rate, availability, or throughput.

service-level objective (SLO)

A target metric that represents the health of a service, as measured by a service-level indicator.

shared responsibility model

A model describing the responsibility you share with AWS for cloud security and compliance. AWS is responsible for security *of* the cloud, whereas you are responsible for security *in* the cloud. For more information, see Shared responsibility model.

SIEM

See security information and event management system.

single point of failure (SPOF)

A failure in a single, critical component of an application that can disrupt the system.

SLA

See service-level agreement.

SLI

See service-level indicator.

SLO

See service-level objective.

split-and-seed model

A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization’s capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.

SPOF

See single point of failure.

star schema

A database organizational structure that uses one large fact table to store transactional or measured data and uses one or more smaller dimensional tables to store data attributes. This structure is designed for use in a data warehouse or for business intelligence purposes.

strangler fig pattern

An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was introduced by Martin Fowler as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

subnet

A range of IP addresses in your VPC. A subnet must reside in a single Availability Zone.

supervisory control and data acquisition (SCADA)

In manufacturing, a system that uses hardware and software to monitor physical assets and production operations.

symmetric encryption

An encryption algorithm that uses the same key to encrypt and decrypt the data.

synthetic testing

Testing a system in a way that simulates user interactions to detect potential issues or to monitor performance. You can use Amazon CloudWatch Synthetics to create these tests.

T

tags

Key-value pairs that act as metadata for organizing your AWS resources. Tags can help you manage, identify, organize, search for, and filter resources. For more information, see Tagging your AWS resources.

target variable

The value that you are trying to predict in supervised ML. This is also referred to as an *outcome variable*. For example, in a manufacturing setting the target variable could be a product defect.

task list

A tool that is used to track progress through a runbook. A task list contains an overview of the runbook and a list of general tasks to be completed. For each general task, it includes the estimated amount of time required, the owner, and the progress.

test environment

See environment.

training

To provide data for your ML model to learn from. The training data must contain the correct answer. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict). It outputs an ML model that captures these patterns. You can then use the ML model to make predictions on new data for which you don't know the target.

transit gateway

A network transit hub that you can use to interconnect your VPCs and on-premises networks. For more information, see What is a transit gateway in the AWS Transit Gateway documentation.

trunk-based workflow

An approach in which developers build and test features locally in a feature branch and then merge those changes into the main branch. The main branch is then built to the development, preproduction, and production environments, sequentially.

trusted access

Granting permissions to a service that you specify to perform tasks in your organization in AWS Organizations and in its accounts on your behalf. The trusted service creates a service-linked role in each account, when that role is needed, to perform management tasks for you. For more information, see Using AWS Organizations with other AWS services in the AWS Organizations documentation.

tuning

To change aspects of your training process to improve the ML model's accuracy. For example, you can train the ML model by generating a labeling set, adding labels, and then repeating these steps several times under different settings to optimize the model.

two-pizza team

A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development.

U

uncertainty

A concept that refers to imprecise, incomplete, or unknown information that can undermine the reliability of predictive ML models. There are two types of uncertainty: *Epistemic uncertainty* is caused by limited, incomplete data, whereas *aleatoric uncertainty* is caused by the noise and randomness inherent in the data. For more information, see the Quantifying uncertainty in deep learning systems guide.

undifferentiated tasks

Also known as *heavy lifting*, work that is necessary to create and operate an application but that doesn't provide direct value to the end user or provide competitive advantage. Examples of undifferentiated tasks include procurement, maintenance, and capacity planning.

upper environments

See environment.

V

vacuuming

A database maintenance operation that involves cleaning up after incremental updates to reclaim storage and improve performance.

version control

Processes and tools that track changes, such as changes to source code in a repository.

VPC peering

A connection between two VPCs that allows you to route traffic by using private IP addresses. For more information, see What is VPC peering in the Amazon VPC documentation.

vulnerability

A software or hardware flaw that compromises the security of the system.

W

warm cache

A buffer cache that contains current, relevant data that is frequently accessed. The database instance can read from the buffer cache, which is faster than reading from the main memory or disk.

warm data

Data that is infrequently accessed. When querying this kind of data, moderately slow queries are typically acceptable.

window function

A SQL function that performs a calculation on a group of rows that relate in some way to the current record. Window functions are useful for processing tasks, such as calculating a moving average or accessing the value of rows based on the relative position of the current row.

workload

A collection of resources and code that delivers business value, such as a customer-facing application or backend process.

workstream

Functional groups in a migration project that are responsible for a specific set of tasks. Each workstream is independent but supports the other workstreams in the project. For example, the portfolio workstream is responsible for prioritizing applications, wave planning, and collecting migration metadata. The portfolio workstream delivers these assets to the migration workstream, which then migrates the servers and applications.

WORM

See write once, read many.

WQF

See AWS Workload Qualification Framework.

write once, read many (WORM)

A storage model that writes data a single time and prevents the data from being deleted or modified. Authorized users can read the data as many times as needed, but they cannot change it. This data storage infrastructure is considered immutable.

Z

zero-day exploit

An attack, typically malware, that takes advantage of a zero-day vulnerability.

zero-day vulnerability

An unmitigated flaw or vulnerability in a production system. Threat actors can use this type of vulnerability to attack the system. Developers frequently become aware of the vulnerability as a result of the attack.

zombie application

An application that has an average CPU and memory usage below 5 percent. In a migration project, it is common to retire these applications.