Machine Learning Lens

AWS Well-Architected Framework

Machine Learning Lens: AWS Well-Architected Framework

Table of Contents

Abstract 1
Abstract 1
Introduction 2
Definitions 3
Machine Learning Stack 3
AI Services 3
ML Services 3
ML Frameworks and Infrastructure 4
Combining Levels 4
Phases of ML Workloads 4
Business Goal Identification 5
ML Problem Framing 5
Data Collection 6
Data Preparation 6
Data Visualization and Analytics 7
Feature Engineering 8
Model Training 9
Model Evaluation and Business Evaluation 10
General Design Principles 12
Scenarios 13
Build Intelligent Applications using AWS AI Services 13
Reference Architecture 14
Adding Sophistication 15
Using AI services with your Data 16
Use Managed ML Services to Build Custom ML Models 16
Reference Architecture 17
Managed ETL Services for Data Processing 18
Reference Architecture 18
Machine Learning on Edge and on Multiple Platforms 19
Reference Architecture 20
Model Deployment Approaches 21
Standard Deployment 22
Blue/Green Deployments 22
Canary Deployment 24
A/B Testing 24
The Pillars of the Well-Architected Framework 26
Operational Excellence Pillar 26
Design Principles 26
Best Practices 27
Resources 33
Security Pillar 33
Design Principles 33
Best Practices 34
Resources 39
Reliability Pillar 39
Design Principles 39
Best Practices 39
Resources 43
Performance Efficiency Pillar 43
Design Principles 43
Best Practices 44
Resources 46
Cost Optimization Pillar 46
Design Principles 46

Best Practices 47
Resources 51
Conclusion 52
Contributors 53
Further Reading 54
Document Revisions 55
Notices 56

Machine Learning Lens - AWS Well-Architected Framework

Publication date: **April 2020** (*Document Revisions* (p. 55))

Abstract

This document describes the *Machine Learning Lens* for the AWS Well-Architected Framework. The document includes common machine learning (ML) scenarios and identifies key elements to ensure that your workloads are architected according to best practices.

Introduction

The AWS Well-Architected Framework helps you understand the pros and cons of decisions you make while building systems on AWS. Using the Framework, allows you to learn architectural best practices for designing and operating reliable, secure, efficient, and cost-effective systems in the cloud. It provides a way for you to consistently measure your architectures against best practices and identify areas for improvement. We believe that having well-architected systems greatly increases the likelihood of business success.

In the *Machine Learning Lens,* we focus on how to design, deploy, and architect your *machine learning workloads* in the AWS Cloud. This lens adds to the best practices included in the Well-Architected Framework. For brevity, we only include details in this lens that are specific to machine learning (ML) workloads. When designing ML workloads, you should use applicable best practices and questions from the AWS Well-Architected Framework whitepaper.

This lens is intended for those in a technology role, such as chief technology officers (CTOs), architects, developers, and operations team members. After reading this paper, you will understand the best practices and strategies to use when you design and operate ML workloads on AWS.

Definitions

The Machine Learning Lens is based on five pillars: operational excellence, security, reliability, performance efficiency, and cost optimization. AWS provides multiple core components for ML workloads that enable you to design robust architectures for your ML applications.

There are two areas that you should evaluate when you build a machine learning workload.

Topics

- Machine Learning Stack (p. 3)
- Phases of ML Workloads (p. 4)

Machine Learning Stack

When you build an ML-based workload in AWS, you can choose from different levels of abstraction to balance speed to market with level of customization and ML skill level:

- Artificial Intelligence (AI) Services
- ML Services
- ML Frameworks and Infrastructure

Artificial Intelligence (AI) Services

The AI Services level provides fully managed services that enable you to quickly add ML capabilities to your workloads using API calls. This gives you the ability to build powerful, intelligent applications with capabilities such as computer vision, speech, natural language, chatbots, predictions, and recommendations. Services at this level are based on pre-trained or automatically trained machine learning and deep learning models, so that you don't need ML knowledge to use them.

AWS provides many AI services that you can integrate with your applications through API calls. For example, you can use Amazon Translate to translate or localize text content, Amazon Polly for text-to-speech conversion, and Amazon Lex for building conversational chat bots.

ML Services

The ML Services level provides managed services and resources for machine learning to developers, data scientists, and researchers. These types of services enable you to label data, build, train, deploy, and operate custom ML models without having to worry about the underlying infrastructure needs. The undifferentiated heavy lifting of infrastructure management is managed by the cloud vendor, so that your data science teams can focus on what they do best.

In AWS, Amazon SageMaker enables developers and data scientists to quickly and easily build, train, and deploy ML models at any scale. For example, Amazon SageMaker Ground Truth helps you build highly accurate ML training datasets quickly and Amazon SageMaker Neo enables developers to train ML models once, and then run them anywhere in the cloud or at the edge.

ML Frameworks and Infrastructure

The ML Frameworks and Infrastructure level is intended for expert machine learning practitioners. These people are comfortable with designing their own tools and workflows to build, train, tune, and deploy models, and are accustomed to working at the framework and infrastructure level.

In AWS, you can use open source ML frameworks, such as TensorFlow, PyTorch, and Apache MXNet. The Deep Learning AMI and Deep Learning Containers in this level have multiple ML frameworks preinstalled that are optimized for performance. This optimization means that they are always ready to be launched on the powerful, ML-optimized compute infrastructure, such as Amazon EC2 P3 and P3dn instances, that provides a boost of speed and efficiency to machine learning workloads.

Combining Levels

Workloads often use services from multiple levels of the ML stack. Depending on the business use case, services and infrastructure from the different levels can be combined to satisfy multiple requirements and achieve multiple business goals. For example, you can use AI services for sentiment analysis of customer reviews on your retail website, and use managed ML services to build a custom model using your own data to predict future sales.

Phases of ML Workloads

Building and operating a typical ML workload is an iterative process, and consists of multiple phases. We identify these phases loosely based on the open standard process model for Cross Industry Standard Process Data Mining (CRISP-DM) as a general guideline. CRISP-DM is used as a baseline because it's a proven tool in the industry and is application neutral, which makes it an easy-to-apply methodology that is applicable to a wide variety of ML pipelines and workloads.

The end-to-end machine learning process includes the following phases:

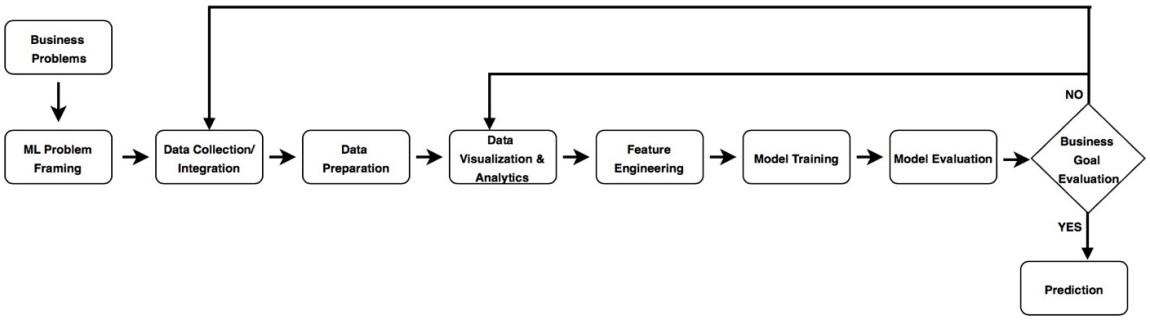

Figure 1 – End-to-End Machine Learning Process

Topics

- Business Goal Identification (p. 5)
- ML Problem Framing (p. 5)
- Data Collection (p. 6)
- Data Preparation (p. 6)
- Data Visualization and Analytics (p. 7)
- Feature Engineering (p. 8)
- Model Training (p. 9)

- Model Evaluation and Business Evaluation (p. 10)

Business Goal Identification

Business Goal Identification is the most important phase. An organization considering ML should have a clear idea of the problem to be solved, and the business value to be gained by solving that problem using ML. You must be able to measure business value against specific business objectives and success criteria. While this holds true for any technical solution, this step is particularly challenging when considering ML solutions because ML is a disruptive technology.

After you determine your criteria for success, evaluate your organization's ability to realistically execute toward that target. The target should be achievable and provide a clear path to production.

You will want to validate that ML is the appropriate approach to deliver your business goal. Evaluate all of the options that you have available for achieving the goal, how accurate the resulting outcomes would be, and the cost and scalability of each approach when deciding your approach.

For an ML-based approach to be successful, having an abundance of relevant, high-quality data that is applicable to the algorithm that you are trying to train is essential. Carefully evaluate the availability of the data to make sure that the correct data sources are available and accessible. For example, you need training data to train and benchmark your ML model, but you also need data from the business to evaluate the value of an ML solution.

Apply these best practices:

- Understand business requirements
- Form a business question
- Determine a project's ML feasibility and data requirements
- Evaluate the cost of data acquisition, training, inference, and wrong predictions
- Review proven or published work in similar domains, if available
- Determine key performance metrics, including acceptable errors
- Define the machine learning task based on the business question
- Identify critical, must have features

ML Problem Framing

In this phase, the business problem is framed as a machine learning problem: what is observed and what should be predicted (known as a label or target variable). Determining what to predict and how performance and error metrics need to be optimized is a key step in ML.

For example, imagine a scenario where a manufacturing company wants to identify which products will maximize profits. Reaching this business goal partially depends on determining the right number of products to produce. In this scenario, you want to predict the future sales of the product, based on past and current sales. Predicting future sales becomes the problem to solve, and using ML is one approach that can be used to solve it.

Apply these best practices:

- Define criteria for a successful outcome of the project
- Establish an observable and quantifiable performance metric for the project, such as accuracy, prediction latency, or minimizing inventory value
- Formulate the ML question in terms of inputs, desired outputs, and the performance metric to be optimized

- Evaluate whether ML is a feasible and appropriate approach
- Create a data sourcing and data annotation objective, and a strategy to achieve it
- Start with a simple model that is easy to interpret, and which makes debugging more manageable

Data Collection

In ML workloads, the data (inputs and corresponding desired output) serves three important functions:

- Defining the goal of the system: the output representation and the relationship of each output to each input, by means of input/output pairs
- Training the algorithm that will associate inputs to outputs
- Measuring the performance of the trained model, and evaluating whether the performance target was met

The first step is to identify what data is needed for your ML model, and evaluate the various means available for collecting that data to train your model.

As organizations collect and analyze increasingly large amounts of data, traditional on-premises solutions for data storage, data management, and analytics can no longer keep pace. A cloud-based data lake is a centralized repository that allows you to store all your structured and unstructured data regardless of scale. You can store your data as-is, without first having to structure the data, and run different types of analytics—from dashboards and visualizations to big data processing, real-time analytics, and ML—to guide you to better decisions.

AWS provides you with a number of ways to ingest data in bulk from static resources, or from new, dynamically generated sources, such as websites, mobile apps, and internet-connected devices. For example, you can build a highly scalable data lake using Amazon Simple Storage Service (Amazon S3). To easily set up your data lake, you can use AWS Lake Formation.

To ingest data, you can use AWS Direct Connect to privately connect your data center directly to an AWS Region. You can physically transfer petabytes of data in batches using AWS Snowball, or, if you have exabytes of data, by using AWS Snowmobile. You can integrate your existing on-premises storage using AWS Storage Gateway, or add cloud capabilities using AWS Snowball Edge. You can also use Amazon Kinesis Data Firehose to collect and ingest multiple streaming-data sources.

Apply these best practices:

- Detail the various sources and steps needed to extract data
- Confirm the availability of the data, both quantity and quality
- Comprehensively understand your data before preparing it for downstream consumption
- Define data governance: who owns the data, who has access, the appropriate usage of the data, and the ability to access and delete specific pieces of data on demand
- Track data lineage, so that the location and data source is tracked and known during further processing
- Use managed AWS services for your data collection and integration
- Use a centralized approach to store your data, such as a data lake

Data Preparation

ML models are only as good as the data that is used to train them. After the data is collected, the integration, annotation, preparation, and processing of that data is critical. An essential characteristic of suitable training data is that it's provided in a way that is optimized for learning and generalization.

Data preparation should start with a small, statistically valid sample, and iteratively be improved with different data preparation strategies, while continuously maintaining data integrity.

AWS provides several services that you can use to annotate your data, extract, transfer, and load (ETL) data at scale.

Amazon SageMaker is a fully managed service that encompasses the entire ML workflow to label and prepare your data, choose an algorithm, train it, tune and optimize it for deployment, and make predictions.

Amazon SageMaker Ground Truth offers easy access to public and private human labelers and provides built-in workflows and user interfaces for common labeling tasks. It uses a machine learning model to automatically label raw data to produce high-quality training datasets quickly at a fraction of the cost of manual labeling. Data is only routed to humans if the active learning model cannot confidently label it. The service provides dynamic custom workflows, job chaining, and job tracking to save time on subsequent ML labeling jobs by using the output from previous labeling jobs as the input to new labeling jobs.

AWS Glue is a fully managed extract, transform, and load (ETL) service that can be used to automate the ETL pipeline. AWS Glue automatically discovers and profiles your data with the Glue Data Catalog, recommends and generates ETL code to transform your source data into target schemas, and runs the ETL jobs on a fully managed, scale-out Apache Spark environment to load your data to its destination. It also enables you to set up, orchestrate, and monitor complex data flows.

Amazon EMR provides a managed Hadoop framework that makes it easy and fast to process vast amounts of data across dynamically scalable Amazon EC2 instances. You can also run other popular distributed frameworks in EMR, such as Apache Spark, HBase, Presto, and Flink, and interact with data in other AWS data stores, such as Amazon S3 and Amazon DynamoDB.

Data preparation applies not only to the training data used for building a machine learning model, but also to the new business data that is used to make inferences against the model after the model is deployed. Typically, the same sequence of data processing steps that you apply to the training data is also applied to the inference requests.

Amazon SageMaker Inference Pipeline deploys pipelines so that you can pass raw input data and execute pre-processing, predictions, and post-processing on both real-time and batch inference requests. Inference pipelines enable you to reuse existing data processing functionality.

Apply these best practices:

- Start with a small, statistically valid set of sample data for data preparation
- Iteratively experiment with different data preparation strategies
- Implement a feedback loop during the data cleaning process that provides alerts for anomalies through the data preparation steps
- Enforce data integrity continuously
- Take advantage of managed ETL services

Data Visualization and Analytics

A key aspect to understanding your data is to identify patterns. These patterns are often not evident when you are only looking at data in tables. The correct visualization tool can help you quickly gain a deeper understanding of your data. Before creating any chart or graph, you must decide what you want to show. For example, charts can convey information such as key performance indicators (KPI), relationships, comparisons, distributions, or compositions.

AWS provides several services that you can use to visualize and analyze data at scale.

Amazon SageMaker provides a hosted Jupyter notebook environment that you can use to visualize and analyze data. Project Jupyter is an open-source web application that allows you to create visualizations and narrative text, as well as perform data cleaning, data transformation, numerical simulation, statistical modeling, and data visualization.

Amazon Athena is a fully managed interactive query service that you can use to query data in Amazon S3 using ANSI SQL operators and functions. Amazon Athena is serverless and can scale seamlessly to meet your querying demands.

Amazon Kinesis Data Analytics provides real-time analytic capabilities by analyzing streaming data to gain actionable insights. The service scales automatically to match the volume and throughput of your incoming data.

Amazon QuickSight is a cloud-powered business intelligence (BI) service that provides dashboards and visualizations. The service automatically scales to support hundreds of users and offers secure sharing and collaboration for storyboarding. Additionally, the service has built-in ML capabilities that provide out-of-the-box anomaly detection, forecasting, and what-if analysis.

Apply these best practices:

- Profile your data (categorical vs. ordinal vs. quantitative visualization)
- Choose the correct tool or combination of tools for your use case (such as data size, data complexity, and real-time vs. batch)
- Monitor your data analysis pipeline
- Validate assumptions about your data

Feature Engineering

After exploring and gaining understanding of your data through visualizations and analytics, it's time for feature engineering. Every unique attribute of the data is considered a feature. For example, when designing a solution for a problem of predicting customer churn, you start with the customer data that has been collected over time. The customer data captures features (also known as attributes), such as customer location, age, income level, and recent purchases.

Feature engineering is a process to select and transform variables when creating a predictive model using machine learning or statistical modeling. Feature engineering typically includes feature creation, feature transformation, feature extraction, and feature selection.

- **Feature creation** identifies the features in the dataset that are relevant to the problem at hand.
- **Feature transformation** manages replacing missing features or features that are not valid. Some techniques include forming Cartesian products of features, non-linear transformations (such as binning numeric variables into categories), and creating domain-specific features.
- **Feature extraction** is the process of creating new features from existing features, typically with the goal of reducing the dimensionality of the features.
- **Feature selection** is the filtering of irrelevant or redundant features from your dataset. This is usually done by observing variance or correlation thresholds to determine which features to remove.

Amazon SageMaker provides a Jupyter notebook environment with Spark and scikit-learn preprocessors that you can use to engineer features and transform the data. From Amazon SageMaker, you can also run feature extraction and transformation jobs using ETL services, such as AWS Glue or Amazon EMR. In addition, you can use Amazon SageMaker Inference Pipeline to reuse existing data processing functionality.

Amazon SageMaker Processing enables running analytics jobs for feature engineering (and model evaluation) at scale in a fully managed environment with all the security and compliance features

provided by Amazon SageMaker. With Amazon SageMaker Processing, you have the flexibility of using the built-in data processing containers or bringing you own containers and submitting custom jobs to run on managed infrastructure. Once submitted, Amazon SageMaker launches the compute instances, processes the input data, analyzes it, and then releases the resources upon completion.

Apply these best practices:

- Use domain experts to help evaluate the feasibility and importance of features
- Remove redundant and irrelevant features (to reduce the noise in the data and reduce correlations)
- Start with features that generalize across contexts
- Iterate as you build your model (new features, feature combinations, and new tuning objectives)

Model Training

In this phase, you select a machine learning algorithm that is appropriate for your problem and then train the ML model. As part of that training, you provide the algorithm with the training data to learn from and set the model parameters to optimize the training process.

Typically, a training algorithm computes several metrics, such as training error and prediction accuracy. These metrics help determine whether the model is learning well and will generalize well for making predictions on unseen data. Metrics reported by the algorithm depend on the business problem and on the ML technique that you used. For example, a classification algorithm can be measured by a *confusion matrix* that captures true or false positives and true or false negatives, while a regression algorithm can be measured by root mean square error (RMSE).

Settings that can be tuned to control the behavior of the ML algorithm and the resulting model architecture are referred to as *hyperparameters*. The number and type of hyperparameters in ML algorithms are specific to each model. Some examples of commonly used hyperparameters are: Learning Rate, Number of Epochs, Hidden Layers, Hidden Units, and Activation Functions. Hyperparameter tuning, or optimization, is the process of choosing the optimal model architecture.

Amazon SageMaker provides several popular built-in algorithms that can be trained with the training data that you prepared and stored in Amazon S3. You can also bring your own custom algorithms to be trained on Amazon SageMaker. The custom algorithm should be containerized using Amazon ECS and Amazon ECR.

After you select the algorithm, you can start training on Amazon SageMaker with an API call. You can choose to train on a single instance or on a distributed cluster of instances. The infrastructure management that is needed for the training process is managed by Amazon SageMaker, which removes the burden of undifferentiated heavy lifting.

Amazon SageMaker also enables automatic model tuning through hyperparameter tuning jobs. After it's configured, a hyperparameter tuning job finds the best version of the model by running many training jobs on your dataset using the algorithm and the hyperparameter ranges that you specify. It then selects the hyperparameter values that result in the best performing model, as measured by the metrics selected. You can use Amazon SageMaker automatic model tuning with built-in algorithms, custom algorithms, and Amazon SageMaker pre-built containers for ML frameworks.

Amazon SageMaker Debugger provides visibility into the ML training process by monitoring, recording, and analyzing data that captures the state of a training job at periodic intervals. It also provides the ability to perform interactive exploration of data captured during training and an alerting capability for errors detected during training. For example, it can automatically detect and alert you to commonly occurring errors, such as gradient values getting too large or too small.

Amazon SageMaker Autopilot simplifies the ML training process by handling the data preprocessing, algorithm selection, and hyperparameter tuning automatically. It allows you to build classification

and regression models simply by providing training data in tabular format. This capability explores multiple ML solutions with different combinations of data preprocessors, algorithms, and algorithm parameter settings, to find the most accurate model. Amazon SageMaker Autopilot selects the best algorithm from the list of high-performing algorithms that it natively supports. It also automatically tries different parameter settings on those algorithms to get the best model quality. You can then directly deploy the best model to production, or evaluate multiple candidates to trade off metrics like accuracy, latency, and model size.

AWS Deep Learning AMI and AWS Deep Learning Containers enable you to use several open source ML frameworks for training on your infrastructure. AWS Deep Learning AMI has popular deep learning frameworks and interfaces preinstalled, such as TensorFlow, PyTorch, Apache MXNet, Chainer, Gluon, Horovod, and Keras. The AMI or container can be launched on powerful infrastructure that has been optimized for ML performance.

Amazon EMR has distributed cluster capabilities and is also an option for running training jobs on the data that is either stored locally on the cluster or in Amazon S3.

Apply these best practices:

- Generate a model testing plan before you train your model
- Have a clear understanding of the type of algorithm that you need to train
- Make sure that the training data is representative of your business problem
- Use managed services for your training deployments
- Apply incremental training or transfer learning strategies
- Stop training jobs early when the results, as measured by the objective metric, are not improving significantly to avoid overfitting and reduce cost
- Closely monitor your training metrics, because model performance may degrade over time
- Take advantage of managed services for automatic model tuning

Model Evaluation and Business Evaluation

After the model has been trained, evaluate it to determine if its performance and accuracy will enable you to achieve your business goals. You might want to generate multiple models using different methods and evaluate the effectiveness of each model. For example, you could apply different business rules for each model, and then apply various measures to determine each model's suitability. You also might evaluate whether your model needs to be more sensitive than specific, or more specific than sensitive. For multiclass models, evaluate error rates for each class separately.

You can evaluate your model using historical data (offline evaluation) or live data (online evaluation). In offline evaluation, the trained model is evaluated with a portion of the dataset that has been set aside as a *holdout set*. This holdout data is never used for model training or validation—it's only used to evaluate errors in the final model. The holdout data annotations need to have high accuracy for the evaluation to make sense. Allocate additional resources to verify the accuracy of the holdout data.

AWS services that are used for model training also have a role in this phase. Model validation can be performed using Amazon SageMaker, AWS Deep Learning AMI, or Amazon EMR.

Based on the evaluation results, you might fine-tune the data, the algorithm, or both. When you fine-tune the data, you apply the concepts of data cleansing, preparation, and feature engineering.

Apply these best practices:

- Have a clear understanding of how you measure success
- Evaluate the model metrics against the business expectations for the project
- Plan and execute Production Deployment (Model Deployment and Model Inference)

After a model is trained, tuned, and tested, you can deploy the model into production, and make inferences (predictions) against the model.

Amazon SageMaker offers a broad variety of options for deployment and inference, and is the recommended AWS service for hosting your production ML models.

As with model training, you can host models on Amazon SageMaker using an API call. You can choose to host your model on a single instance or across multiple instances. The same API enables you to configure automatic scaling, so that you can serve the varying inference demands on your ML model. The infrastructure management needed for hosting your models is completely managed by Amazon SageMaker—removing the burden of undifferentiated heavy lifting.

Amazon SageMaker Inference Pipelines enable you to deploy inference pipelines so that you can pass raw input data, execute pre-processing, predictions, and complete post-processing on real-time and batch inference requests. Inference pipelines can be composed of any ML framework, built-in algorithm, or custom containers that you can use on Amazon SageMaker. You can build feature data processing and feature engineering pipelines with a suite of feature transformers available in the SparkML and Scikit-learn framework containers, and deploy these as part of the inference pipelines to reuse data processing code and simplify management of your ML processes. These inference pipelines are fully managed and can combine preprocessing, predictions, and post-processing as part of a data science process.

Amazon SageMaker Model Monitor continuously monitors ML models in production. After an ML model is deployed in production, the real-world data might start to differ from the data that was used to train the model, leading to deviations in model quality, and eventually less accurate models. Model Monitor detects deviations, such as data drift, that can degrade model performance over time, and alerts you to take remedial actions.

Amazon SageMaker Neo enables ML models to be trained once and then run anywhere in the cloud and at the edge. Amazon SageMaker Neo consists of a compiler and a runtime. The compilation API reads models exported from various frameworks, converts them into framework-agnostic representations, and generates optimized binary code. The runtime for each target platform then loads and executes the compiled model.

Amazon Elastic Inference enables you to attach low-cost, GPU-powered acceleration to Amazon EC2 and Amazon SageMaker instances to reduce the cost of running deep learning inferences. Standalone GPU instances are designed for model training and are typically oversized for inference. Even though training jobs batch process hundreds of data samples in parallel, most inference happens on a single input in real time and consumes only a small amount of GPU compute. Amazon Elastic Inference solves this problem by allowing you to attach the appropriate amount of GPU-powered inference acceleration to any Amazon EC2 or Amazon SageMaker instance type, with no code changes.

While Elastic Inference is natively supported for a few deep learning frameworks, such as TensforFlow and Apache MXNet, you can also use it with other deep learning frameworks by using Open Neural Network Exchange (ONNX) to export your model and importing it into MXNet.

Apply these best practices:

- Monitor model performance in production and compare to business expectations
- Monitor differences between model performance during training and in production
- When changes in model performance are detected, retrain the model. For example, sales expectations and subsequent predictions may change due to new competition
- Use batch transform as an alternative to hosting services if you want to get inferences on entire datasets
- Take advantage of production variants to test variations of a new model with A/B testing

General Design Principles

The Well-Architected Framework identifies a set of general design principles to facilitate good design in the cloud for machine learning workloads:

- **Enable agility through the availability of high data quality datasets**

  Data science workloads require access to live data or batch data for all phases in a delivery pipeline. Implement mechanisms to enable access to data with data validation and quality controls.
- **Start simple and evolve through experiments**

  By starting with a small set of features, you can avoid the mistake of starting with a complex model and losing track of feature impact. Choose a simple model, and perform a series of experiments throughout the process.
- **Decouple model training and evaluation from model hosting**

  Select resources that best align with specific phases in the data science lifecycle by separating model training, model evaluation, and model hosting resources.
- **Detect data drift**

  To manage data drift over time, continuously measure the accuracy of inference after the model is in production. The data used in ML generally comes from multiple sources, and the shape and meaning of that data can change as upstream systems and processes change. Have mechanisms in place that detect those changes, so that you can take appropriate action.
- **Automate training and evaluation pipeline**

  Automation allows you to trigger automatic model training and the creation of model artifacts, which can then be consistently deployed to multiple endpoint environments. Applying automation to trigger model retraining activities decreases manual effort, reduces human error, and supports continuous improvement of model performance.
- **Prefer higher abstractions to accelerate outcomes**

  When selecting the appropriate AI/ML service, you should evaluate higher-level services for appropriateness first, then as a mechanism to quickly meet your business objectives, remove undifferentiated heavy lifting, and reduce development costs.

Scenarios

The following are a few common scenarios that influence the design and architecture of your machine learning workloads on AWS. Each scenario includes the common drivers for the design, and a reference architecture to show you how to implement each scenario.

Topics

- Build Intelligent Applications using AWS AI Services (p. 13)
- Use Managed ML Services to Build Custom ML Models (p. 16)
- Managed ETL Services for Data Processing (p. 18)
- Machine Learning on Edge and on Multiple Platforms (p. 19)
- Model Deployment Approaches (p. 21)

Build Intelligent Applications using AWS AI Services

The AWS AI services layer in the machine learning stack is a good choice for organizations that want to add AI capabilities to existing or new applications with minimal development effort and quick turnaround time. Services in this layer provide fully managed and ready-to-use computer vision, speech, natural language, and chatbot capabilities.

When developers use these services, they don't have to manage the data preparation, data analysis, model training, and evaluation phases of the ML process. Instead, these capabilities can be integrated into applications using a simple API call.

Amazon Comprehend is a natural language processing (NLP) service that uses ML to help you uncover insights and relationships in your unstructured text data. First, the service identifies the language of the text. Then, it extracts key phrases, places, people, brands, and events. It analyzes text using tokenization and parts of speech, and because the service understands how positive or negative the text is, can automatically organize a collection of text files by topic. You also can use the AutoML capabilities in Amazon Comprehend to build a custom set of entities or text classification models that are tailored uniquely to your organization's needs.

Amazon Lex is a service for building conversational interfaces into any application using voice and text. Amazon Lex provides the advanced deep learning functionalities of automatic speech recognition (ASR) for converting speech to text, and natural language understanding (NLU) to recognize the intent of the text. These capabilities enable you to build applications with highly engaging user experiences and lifelike conversational interactions.

Amazon Polly is a service that turns text into lifelike speech, enabling you to create applications that talk, and build entirely new categories of speech-enabled products. Amazon Polly is a *text-to-speech* service that uses advanced deep learning technologies to synthesize speech that sounds like a human voice.

Amazon Rekognition makes it easy to add image and video analysis to your applications. You provide an image or video to the Amazon Rekognition API, and the service can identify the objects, people, text, scenes, and activities, as well as detect any inappropriate content. Amazon Rekognition also provides highly accurate facial analysis and facial recognition on images and video that you provide. You can detect, analyze, and compare faces for a wide variety of user verification, people counting, and public safety use cases.

Amazon Transcribe is an automatic speech recognition (ASR) service that makes it easy for you to add speech-to-text capability to your applications. Using the Amazon Transcribe API, you can analyze audio files stored in Amazon S3 and have the service return a text file of the transcribed speech. You also can send a live audio stream to Amazon Transcribe and receive a stream of transcripts in real time.

Amazon Translate is a neural machine translation service that delivers fast, high-quality, and affordable language translation. Neural machine translation is a form of language translation automation that uses deep learning models to deliver more accurate and more natural sounding translation than traditional statistical and rule-based translation algorithms. Amazon Translate allows you to localize content—such as websites and applications—for international users, and to easily translate large volumes of text efficiently.

Responses from the AWS AI services also include a *confidence score* that is a representation of how confident the AI service is about a specific result. Because all ML systems are probabilistic by nature, you can think of the confidence score as a measure of how much trust the systems place in their results. When using the AI services, make sure to set an appropriate threshold that is acceptable for your specific use case. For multi-class models, use a per-class threshold, set based on class error rates. For example, using Amazon Rekognition to gauge crowd interest at an event might require a lower confidence score threshold, but using the same service for analyzing medical images might need a higher threshold. Domain-specific use cases with impactful outcomes—such as medical image analysis—might also need a second-level validation by a medical expert.

Because the AI services are serverless and pay-as-you-go, you can grow the service with your business, and keep your costs low during entry phases and non-peak times. The serverless nature of the AI services makes them ideal candidates for event-driven architectures using AWS Lambda. With AWS Lambda, you can run code for virtually any type of application or backend service with zero administration. You pay only for the compute time you consume—there is no charge when your code is not running.

One example use case is capturing and analyzing customer demographic data in a retail store, with the business goal of improving customer experience and engagement. As you capture and process images of faces, you must implement safeguards to protect that data and apply appropriate confidence levels before using that data. The reference architecture in Figure 2 shows an implementation using Amazon Rekognition for facial analysis, Amazon Athena for analyzing the facial attribute data, and Amazon QuickSight for visualization of the analysis.

The use of facial analysis technology must comply with all laws, including laws that protect civil rights. AWS customers are responsible for following all applicable laws in how they use the technology. The AWS Acceptable Use Policy (AUP) prohibits customers from using any AWS service, including Amazon Rekognition, to violate the law, and customers who violate our AUP will not be able to use our services.

Reference Architecture

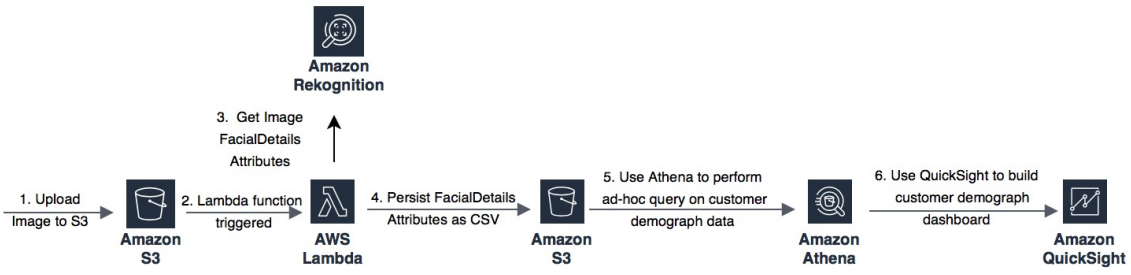

Figure 2 – Customer demographics analysis solution

This reference architecture includes these high-level processes:

- Create an Amazon S3 bucket to temporarily store images and enable encryption on the bucket to protect the images. Restrict access to the S3 bucket using AWS IAM: providing write-only permissions

for the upload process (no public read), and read-only permissions for the AWS Lambda function. Enable data event logging for your S3 bucket to a separate S3 bucket using CloudTrail, so that you can collect logs of all activity related to the bucket.

- Customer images captured in the retail store are uploaded to the **Amazon S3** bucket, so you need to establish **lifecycle policies** to make sure that an image is automatically deleted after processing.
- Each image uploaded to Amazon S3 triggers an AWS Lambda function, and you can use facial analysis to understand demographic data such as age, gender, and sentiment. The Lambda function invokes the Amazon Rekognition service to extract facial attributes from the images, which reflect the customer age, gender, and sentiment, such as happy, calm, or angry. Information about the inference is also included in the attributes, along with the confidence levels.
- The demographic data is stored in .csv format in a second Amazon S3 bucket. Encrypt the .csv file using a unique, securely stored key. A service such as AWS Key Management Service (AWS KMS) can be used to manage and store these keys.
- Amazon Athena reads and loads the demographic data from the .csv files for queries. Amazon Athena supports encrypted data for both the source data and query results, for example, using Amazon S3 with AWS KMS. To make sure that confidence levels are used appropriately, use Views in Amazon Athena to restrict searches to only those with a sufficient degree of confidence for your use case.
- Build customer insight dashboards in Amazon QuickSight. Use AWS IAM to restrict access to the Amazon QuickSight dashboard and Amazon Athena queries to appropriate personnel, with all access logged securely.

In this example, the object of interest is an image and Amazon Rekognition is used to analyze the image. Safeguards are put in place to protect facial images, to automatically delete the images, and use and record the confidence levels used for inference. The correct use of confidence levels is enforced by filtering out low confidence results. This architecture can be used to analyze a variety of object types, such as text or audio, using the appropriate AI service. For example, an audio file can be transcribed using Amazon Transcribe, and unstructured text can be analyzed using Amazon Comprehend.

Adding Sophistication

Though the individual AI services are pre-trained to handle a specific task—such as image analysis or transcription—the serverless nature of these services enables you to build sophisticated solutions by orchestrating multiple services using AWS Step Functions. One example is the AWS Media Analysis Solution, which helps customers to easily analyze, understand, and build a searchable catalog of existing media files. The following reference architecture uses multiple AI services—Amazon Rekognition, Amazon Transcribe, Amazon Comprehend—to analyze and extract metadata from media. The extracted metadata is indexed and persisted in Amazon Elasticsearch Service to make the entire media content searchable.

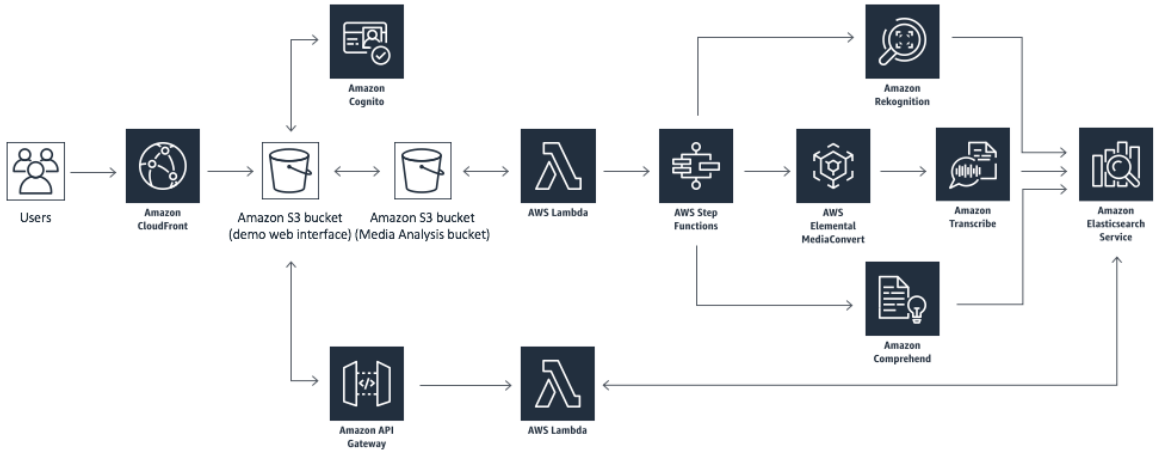

Figure 3 – Media Analysis Solution Reference Architecture

This reference architecture includes these high-level processes:

- Deploy a web interface in an Amazon S3 bucket, which enables you to immediately start analyzing small media files with a simple web interface. Use Amazon CloudFront to restrict access to the contents of the Amazon S3 bucket.
- Uploaded media files flow through an Amazon API Gateway RESTful API, an AWS Lambda function that processes API requests, and an Amazon Cognito user pool enables secure interaction with the media files.
- The AWS Step Functions state machine orchestrates the media analysis processes. A second Lambda function executes the analysis and metadata extraction using managed AI services, such as Amazon Rekognition, Amazon Transcribe, and Amazon Comprehend.
- When an MP4 video file is uploaded, AWS Elemental MediaConvert extracts audio for analysis by Amazon Transcribe and Amazon Comprehend.
- Metadata results are stored in an S3 bucket and indexed in an Amazon Elasticsearch Service (Amazon ES) cluster.

The AI services, which solve specific use cases such as image analysis, language translation, transcription, enable you to build powerful, intelligent capabilities without requiring extensive knowledge of machine learning and deep learning. The result is fast experimentation and evaluation against your business goals, which leads to reduced time to market. In this example, the impact of an error is low, so you can use lower confidence levels for any of your ML approaches.

Using AI services with your Data

While the AI services discussed previously are based on pre-trained models, AWS also offers AI services that return ML models trained with your data.

Amazon Personalize is a fully managed service that allows you to create private, customized personalization recommendations for your applications, based on the user-item interaction data you provide. Whether it's a timely video recommendation inside an application or a personalized notification email delivered just at the right moment, personalized experiences, based on your data, deliver more relevant experiences for customers often with much higher business returns.

Amazon Forecast is a fully managed service that generates highly accurate forecasts based on the historical data you provide. The service uses deep learning to learn from multiple datasets and is able to automatically try different algorithms for the best fit to your data. It can be used for a variety of use cases, such as estimating product demand, cloud computing usage, financial planning, or resource planning in a supply chain management system.

Use Managed ML Services to Build Custom ML Models

You can use a managed services approach to build and deploy ML models, using data that you own, to make predictive and prescriptive models for business value. When you use managed ML services, your development and data science teams are responsible for managing the data preparation, data analysis, model training, model evaluation, and model hosting phases of the end-to-end ML process.

Amazon SageMaker is a fully managed service that encompasses the entire ML workflow to label and prepare your data, choose an algorithm, train the model, tune and optimize it for deployment, make

predictions, and take action. To enable developers and data scientists to build an ML model without the burden of undifferentiated infrastructure management, Amazon SageMaker provides these capabilities:

- **Collect and prepare training data**

  Label your data using Amazon SageMaker Ground Truth, and leverage multiple, pre-built notebooks for many common ML problems.
- **Machine learning algorithm support**

  Choose from multiple, built-in, high-performing algorithms, bring your own algorithm, or explore AWS Marketplace for algorithms that fit your use case.
- **Model training**

  Train the ML model with your own data using an API call that sets up, manages, and terminates a high-performance training cluster. Configure the training cluster to use a single instance, or choose multiple instances to support distributed training. Amazon SageMaker Debugger provides real-time insight into the training process by automating the capture and analysis of data from training runs.
- **Model optimization**

  Train your model once on Amazon SageMaker and then optimize it for other ML frameworks using Amazon SageMaker Neo.
- **Deploy your model in production**

  Deploy your trained models on the auto scaling-capable infrastructure of your choice using an API call.
- **Monitor deployed models**

  Continuously monitor ML models in production to detect deviations such as data drift that can degrade model performance and automate remediation actions.

AWS Lambda is a tool that supports event driven architecture and connects multiple phases of the ML process together, from data ingestion to making predictions.

Reference Architecture

Automation of an end-to-end ML process using Amazon SageMaker, Amazon Kinesis Data Streams, Amazon S3, and AWS Lambda is described in the following reference architecture (Figure 4) for the *Predictive Data Science with Amazon SageMaker and a Data Lake on AWS* solution.

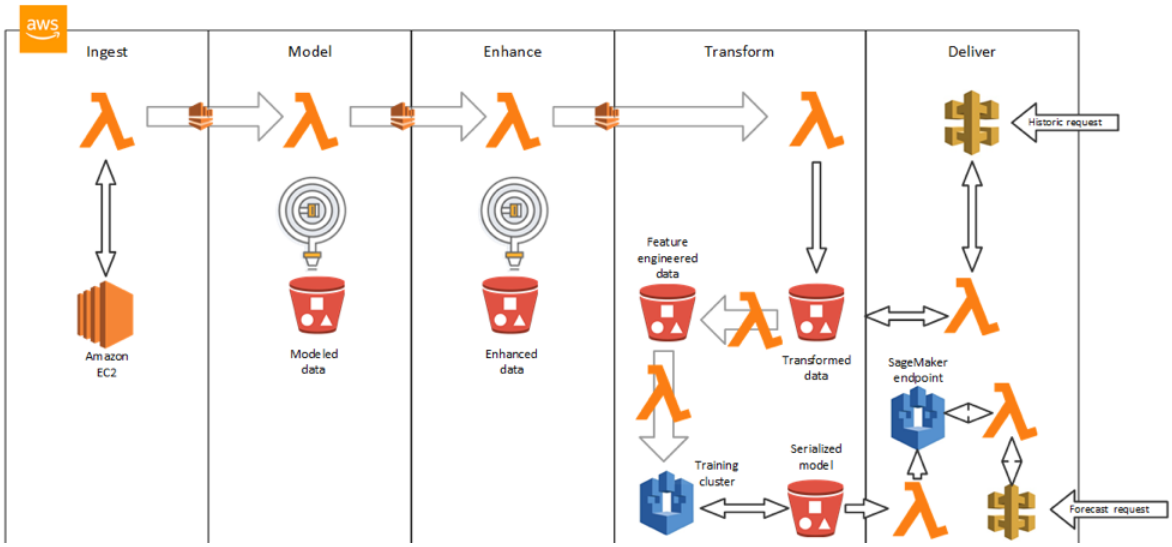

Figure 4 – Predictive Data Science with Amazon SageMaker and a Data Lake on AWS

This reference architecture includes these high-level elements:

- Amazon S3 is used as a data lake that holds the raw, modeled, enhanced, and transformed data.
- Amazon Kinesis Data Streams enable real-time processing of new data across the Ingest, Model, Enhance, and Transform stages.
- Data transformation code is hosted on AWS Lambda to prepare the raw data for consumption and ML model training, and to transform data input and output.
- AWS Lambda automates the Amazon SageMaker API calls to build, manage, and create REST endpoints for new models, based on a schedule, or triggered by data changes in the data lake.

This architecture provides automated, continuous training and improvement of the ML models that use customer data, without the undifferentiated heavy lifting of infrastructure management.

Data transformation code is hosted on AWS Lambda. The data transformation code also can be executed on an Amazon SageMaker notebook instance. However, these options might not be the correct choice in all situations, especially for large-scale data transformations.

Managed ETL Services for Data Processing

Data processing activities, such as cleansing, discovery, and feature engineering at scale, are well suited for tools like Apache Spark, which provide SQL support for data discovery, among other useful utilities. On AWS, Amazon EMR facilitates the management of Spark clusters, and enables capabilities like elastic scaling while minimizing costs through Spot Instance pricing.

Amazon SageMaker notebooks enable connectivity to an external Amazon EMR cluster, enabling data processing on the elastically scalable cluster using Apache Spark. You can then train models and deploy them using the Amazon SageMaker training and deployment APIs.

For example, imagine a business use case of targeted marketing to consumers based on a deep understanding of consumer behavior. Amazon Pinpoint is a managed service that can send targeted messages to consumers through multiple engagement channels, such as emails, text, and SMS. Some examples of targeted campaigns include promotional alerts and customer retention campaigns, and transactional messages, such as order confirmations and password reset messages. However, identifying the correct customers, or customer segments, to send the message to is a critical component. You can use ML to predict future purchasing behavior based on historical consumer purchase patterns. You can then use predicted purchasing behavior to deliver targeted campaigns through Amazon Pinpoint.

Reference Architecture

This reference architecture shows how you can use Amazon EMR, Apache Spark, and Amazon SageMaker for the different phases of ML, and Amazon Pinpoint to send targeted marketing messages.

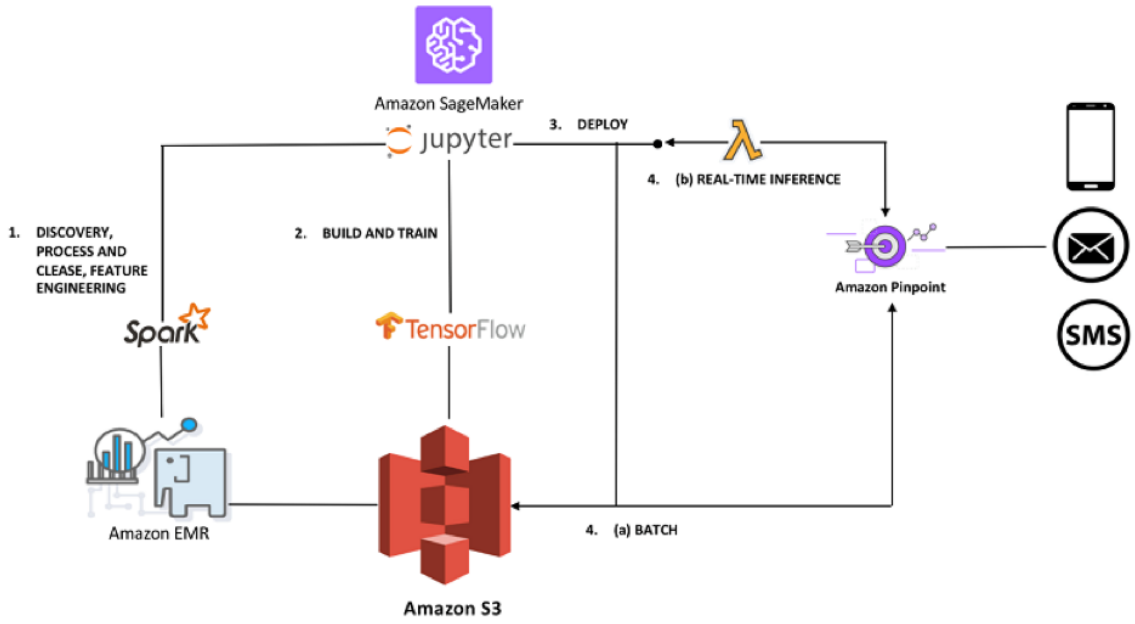

Figure 5 – Amazon Pinpoint Campaign Drive by ML on Amazon SageMaker

This reference architecture includes these high-level elements:

- Use Amazon S3 as a data lake that holds large data volumes.
- Configure an Amazon SageMaker notebook to run against an external Amazon EMR cluster. Data cleansing, processing, discovery, and feature engineering are done using Apache Spark on the EMR cluster. Transformed data is stored in Amazon S3.
- Use Amazon SageMaker to train a custom model using the transformed data and leveraging the distributed training capability.
- Use Amazon SageMaker to create an Auto Scaling API endpoint for the trained model.
- Use the API endpoint to make batch and real-time inferences.
- Process predictions in batches and catalog them in the data lake. The marketing team can then import the data into Amazon Pinpoint to launch a campaign.

Machine Learning on Edge and on Multiple Platforms

Training your ML models requires the powerful compute infrastructure available in the cloud. However, making inferences against these models typically requires far less computational power. In some cases, such as with edge devices, inferencing needs to occur even when there is limited or no connectivity to the cloud. Mining fields are an example of this type of use case. To make sure that an edge device can respond quickly to local events, it's critical that you can get inference results with low latency.

AWS IoT Greengrass enables machine learning on edge devices. AWS IoT Greengrass makes it easy to perform ML inference locally on devices, using models that are created, trained, and optimized in the cloud. ML models built using Amazon SageMaker, AWS Deep Learning AMI, or AWS Deep Learning Containers and persisted in Amazon S3 are deployed on the edge devices.

Figure 6 shows the interaction between AWS IoT Greengrass and the ML model training in the AWS Cloud.

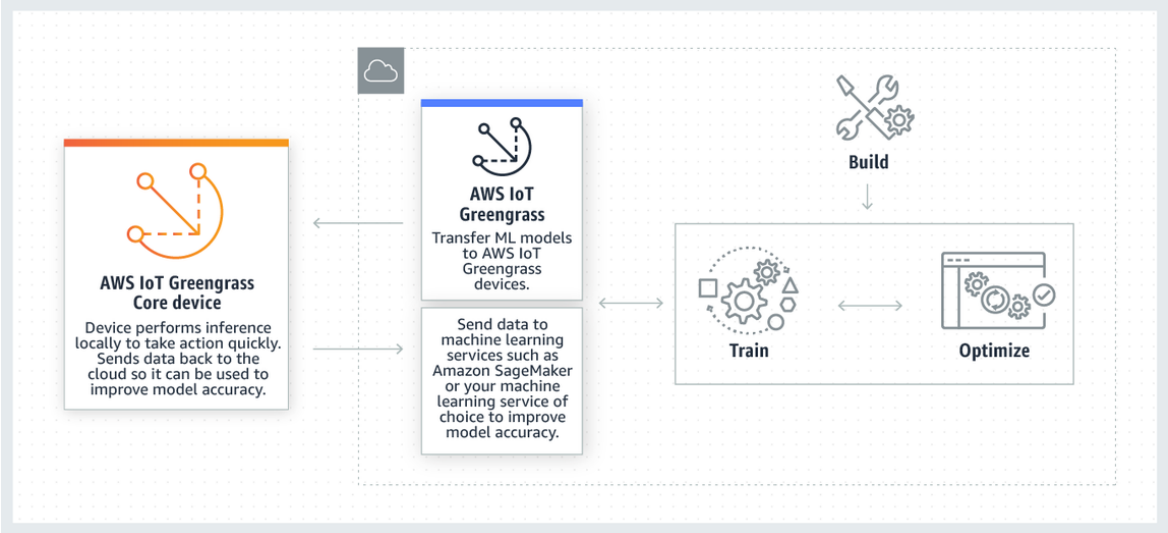

Figure 6 – AWS IoT Greengrass and the ML Model in the Cloud

Performing inference locally on connected devices running AWS IoT Greengrass reduces latency and cost. Instead of sending all device data to the cloud to perform ML inference and make a prediction, you can run inference directly on the device. As predictions are made on these edge devices, you can capture the results and analyze them to detect outliers. Analyzed data can then be sent back to Amazon SageMaker in the cloud, where it can be reclassified and tagged to improve the ML model.

You can use ML models that are built, trained, and optimized in the cloud and run their inference locally on devices. For example, you can build a predictive model in Amazon SageMaker for scene detection analysis, optimize it to run on any camera, and then deploy it to predict suspicious activity and send an alert. Data gathered from the inference running on AWS IoT Greengrass can be sent back to Amazon SageMaker, where it can be tagged and used to continuously improve the quality of the ML models.

Reference Architecture

A reference architecture for a Bird Species Identification on the Edge use case is shown in Figure 7. In this architecture, an object detection model is trained on Amazon SageMaker and deployed to an edge device. Custom object detection has become an important enabler for a wide range of industries and use cases—such as finding tumors in MRIs, identifying diseased crops, and monitoring railway platforms. The edge device used for this use case is AWS DeepLens, which is a deep-learning-enabled video camera.

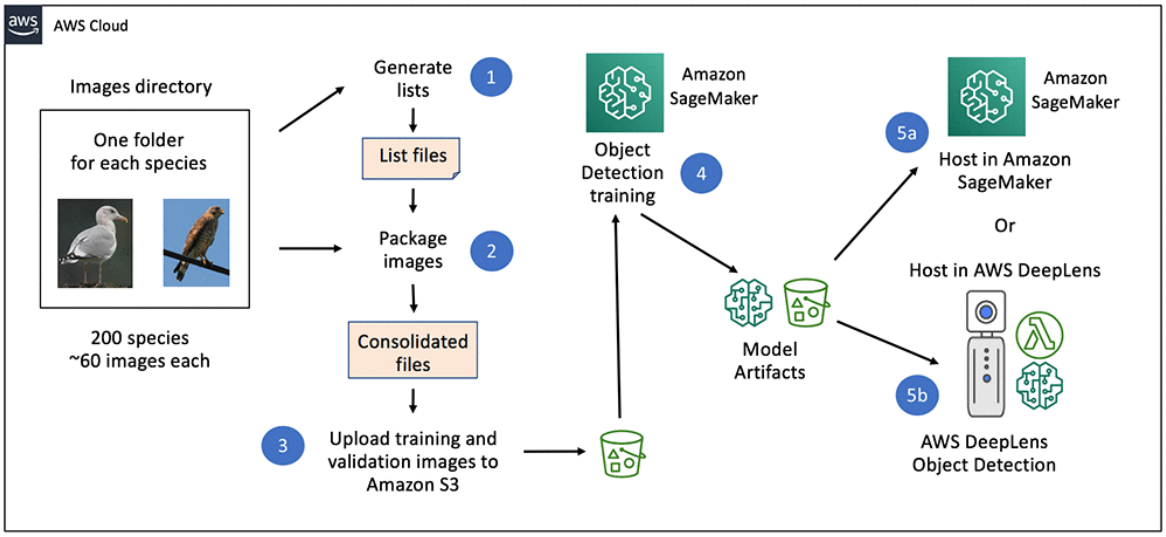

Figure 7 – Bird Species Identification on the Edge Architecture

This reference architecture includes these elements:

- Gather, understand, and prepare the bird image dataset
- Train the object detection model using the Amazon SageMaker built-in algorithm
- Host the model using an Amazon SageMaker endpoint
- Deploy the model to the edge on AWS DeepLens:
  - Convert the model artifacts before you deploy to AWS DeepLens
  - Optimize the model from your AWS Lambda function on AWS DeepLens
- Perform model inference and bird species identification on AWS DeepLens

AWS DeepLens is one of the edge devices used in the previous architecture. Though you could deploy the ML model at the edge and in the cloud to multiple hardware platforms, such as Intel or NVIDIA, that is not always practical because the ML model is tightly coupled to the framework that you used to train it, such as MXNet, Tensor, or PyTorch. If you want to deploy an ML model to a platform other than the specific platform that you trained it for, you must first optimize the model. As the number of ML frameworks and platforms increase, the effort required to optimize the models for additional platforms increases and might become prohibitively time consuming.

Amazon SageMaker Neo includes two components to address this problem: a compiler and a runtime. The compiler converts models to an efficient, common format, which is executed on the device by a compact runtime that uses less than one-hundredth of the resources that a generic framework traditionally consumes. The Amazon SageMaker Neo runtime is optimized for the underlying hardware, and uses specific instruction sets that help speed up ML inference. Models are optimized with less than one tenth of the memory footprint so that they can run on resource constrained devices, such as home security cameras and actuators.

Model Deployment Approaches

A trained ML model should be hosted in a way that consumers can easily invoke and get predictions from it. Consumers of the ML models could be either external or internal to your organization. The consumers of your ML model typically do not understand the ML process and merely want a simple API that can give them predictions in real time, or in batch mode.

Amazon SageMaker provides model hosting services for model deployment, and provides an HTTPS endpoint where the ML model is available to provide inferences.

Deploying a model using Amazon SageMaker hosting services is a three-step process:

1. Create a model in Amazon SageMaker.

   Apply best practices to ensure that the model meets business requirements before proceeding.
2. Create an endpoint configuration for an HTTPS endpoint.

   Specify the name of one or more models in production variants, and the ML compute instances that you want Amazon SageMaker to launch to host each production variant. The endpoint configuration lets you attach several models to the same endpoint, with different weights and instance configurations (production variants). You can update the configuration at any time during the life of the endpoint.
3. Create an HTTPS endpoint.

   Amazon SageMaker launches the ML compute instances and deploys the model, or models, as specified in the endpoint configuration details, and provides an HTTPS endpoint. Consumers of the model can then use the endpoint to make inferences.

The Amazon SageMaker model endpoint configuration *production variants* capability allows multiple ML models to be hosted on different infrastructures, with each processing either a subset or all of the inference requests. You can leverage *production variants* to minimize deployment risks.

For all variants, include a model version in the model endpoint response. When issues arise with model inferences, or in cases where model explainability is required, knowing the specific model version can help you track changes back to their source.

Standard Deployment

In a standard model deployment, the Amazon SageMaker endpoint is configured with a single production variant. The production variant configuration specifies the type and count of the instance to host the model. All inference traffic is processed by the single model hosted on the endpoint.

The following is a sample production variant configuration for a standard deployment.

```
ProductionVariants=[{
  'InstanceType':'ml.m4.xlarge',
  'InitialInstanceCount':1,
  'ModelName':model_name,
  'VariantName':'AllTraffic'
}])
```

Blue/Green Deployments

The blue/green deployment technique provides two identical production environments. You can use this technique when you need to deploy a new version of the model to production.

As shown in Figure 8, this technique requires two identical environments:

- A live production environment (blue) that runs **version n**,
- An exact copy of this environment (green) that runs **version n+1**.

While the blue environment (version n) is processing the live traffic, you test the next release (version n +1) on the green environment with synthetic traffic. Tests should include verifying that the new model

is meeting both technical and business metrics. If all tests of version n+1 in the green environment are a success, then the live traffic is switched to the green environment. You then validate the metrics again in the green environment, this time with live traffic. If you find any issues in this testing, you switch the traffic back to blue environment. If no issues are found for a period of time, you can remove the blue environment.

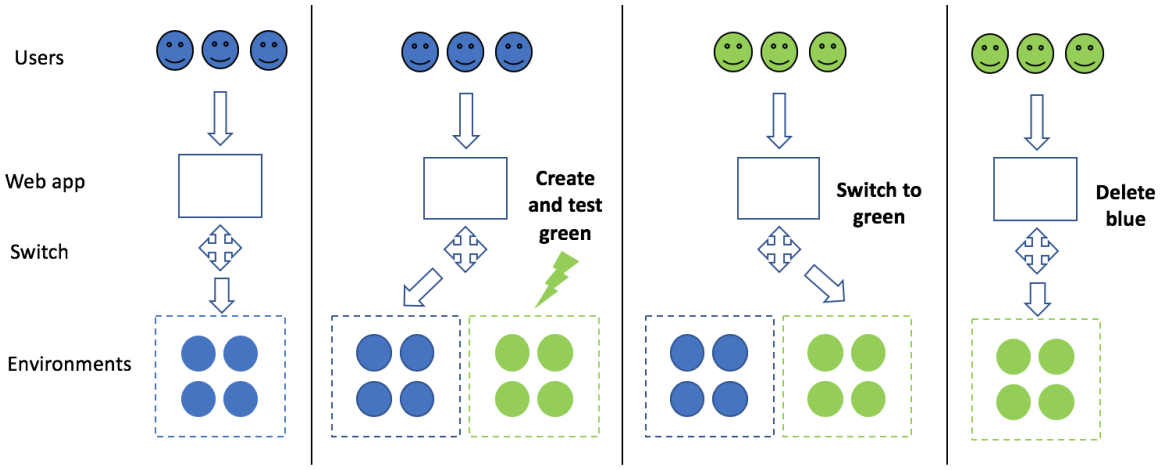

Figure 8 – Blue/Green Deployment Technique

Implementing a blue/green deployment on Amazon SageMaker includes these steps:

1. Create a new endpoint configuration, using the same production variants for the existing live model and for the new model.
2. Update the existing live endpoint with the new endpoint configuration. Amazon SageMaker creates the required infrastructure for the new production variant and updates the weights without any downtime.
3. Switch traffic to the new model through an API call.
4. Create a new endpoint configuration with only the new production variant and apply it to the endpoint.

Amazon SageMaker terminates the infrastructure for the previous production variant.

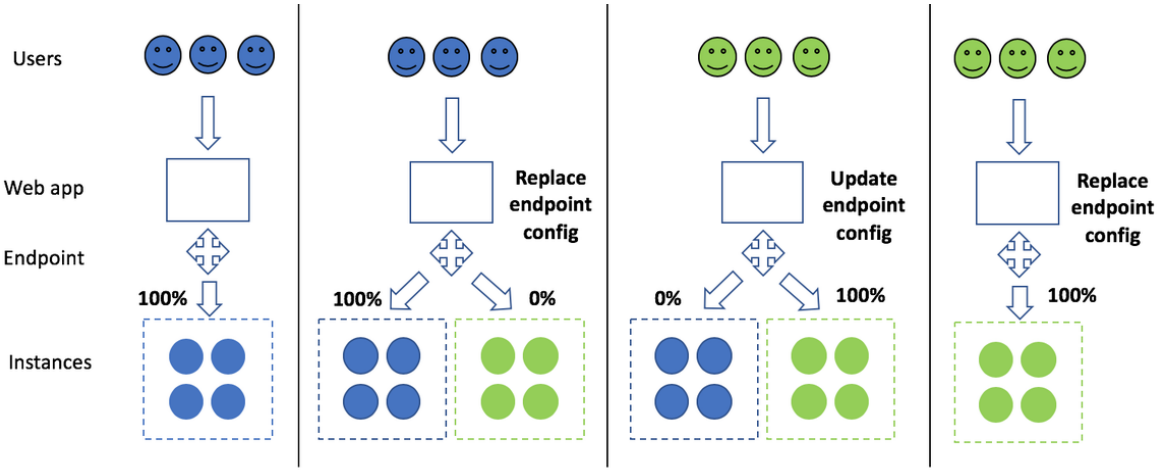

Figure 9 – Blue/Green Model Deployment with Amazon SageMaker Production Variants

Canary Deployment

With a canary deployment, you can validate a new release with minimal risk by first deploying to a small group of your users. Other users continue to use the previous version until you're satisfied with the new release. Then, you can gradually roll the new release out to all users.

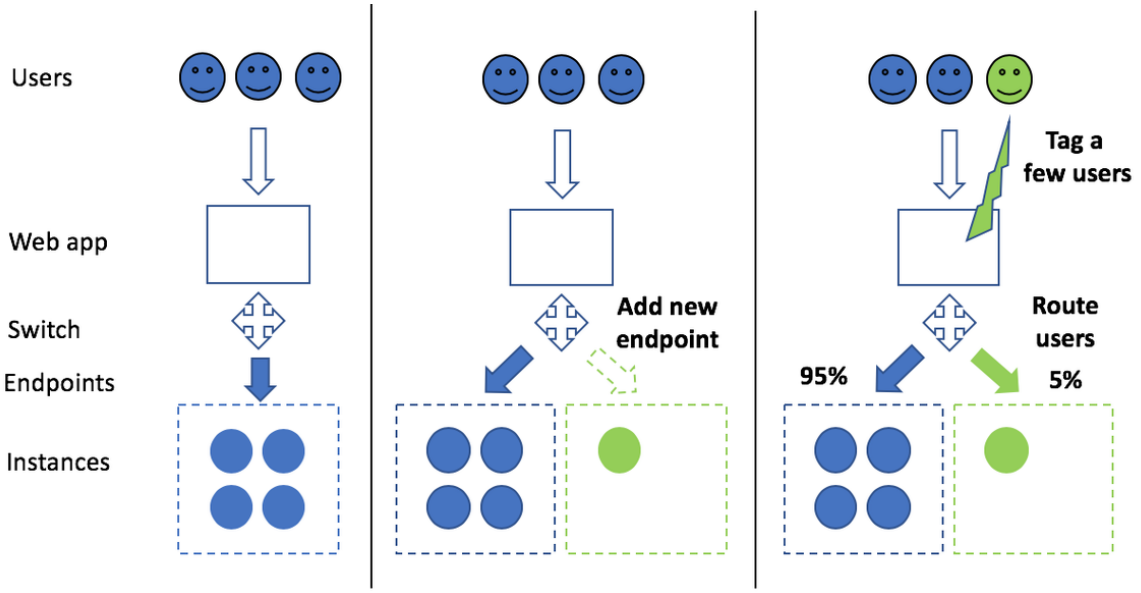

Figure 10 – Canary Deployment with Amazon SageMaker Production Variants: Initial Rollout

After you have confirmed that the new model performs as expected, you can gradually roll it out to all users, scaling endpoints up and down accordingly.

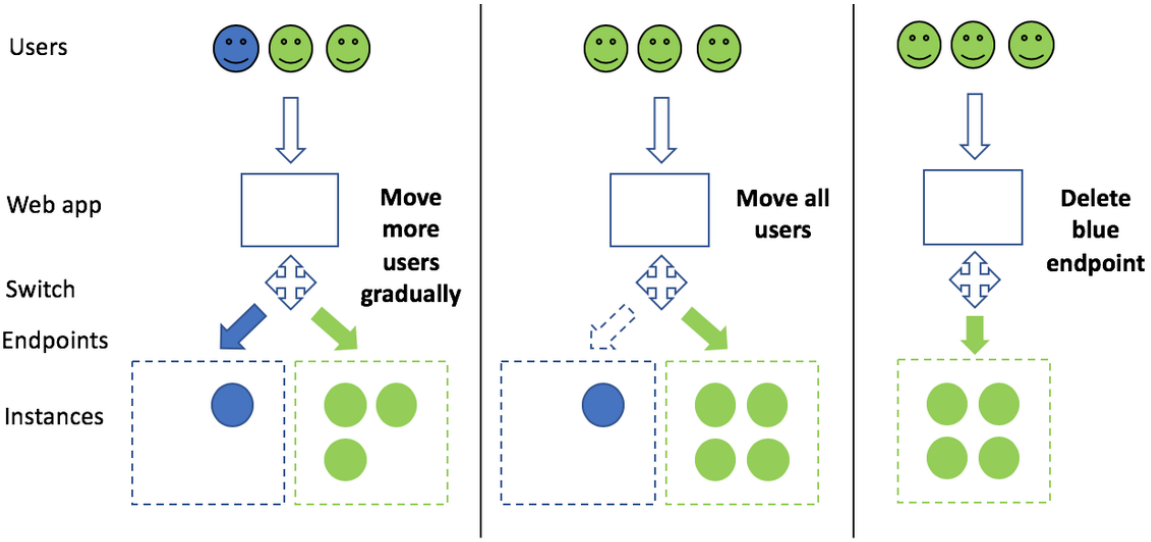

Figure 11 – Canary Deployment with Amazon SageMaker Production Variants: Complete Rollout

A/B Testing

A/B testing is a technique that you can use to compare the performance of different versions of the same feature, while monitoring a high-level metric, such as the click-through rate or conversion rate. In this

context, this means making inferences using different models for different users, and then analyzing the results. The different models are built using the same algorithm (the built-in Amazon SageMaker algorithm or your custom algorithm), but using two different hyperparameter settings.

A/B testing is similar to canary testing, but has larger user groups and a longer time scale, typically days or even weeks. For this type of testing, Amazon SageMaker endpoint configuration uses two production variants: one for model A, and one for model B. To begin, configure the settings for both models to balance traffic between the models equally (50/50) and make sure that both models have identical instance configurations. After you have monitored the performance of both models with the initial setting of equal weights, you can either gradually change the traffic weights to put the models out of balance (60/40, 80/20, etc.), or you can change the weights in a single step, continuing until a single model is processing all of the live traffic.

The following is a sample production variant configuration for A/B testing.

```
ProductionVariants=[
  {
    'InstanceType':'ml.m4.xlarge',
    'InitialInstanceCount':1,
    'ModelName':'model_name_a',
    'VariantName':'Model-A',
    'InitialVariantWeight':1
  },
  {
    'InstanceType':'ml.m4.xlarge',
    'InitialInstanceCount':1,
    'ModelName':'model_name_b',
    'VariantName':'Model-B',
    'InitialVariantWeight':1
  }
])
```

The Pillars of the Well-Architected Framework

Each of the following pillars is important to help you achieve a well-architected machine learning workload solution. For each pillar, we only discuss details that are specific to the Machine Learning Lens, including definitions, best practices, questions, considerations, and the key AWS services that are specific to ML workloads.

When designing your ML workloads, make sure to also use applicable best practices and questions from the AWS Well-Architected Framework whitepaper.

Topics

- Operational Excellence Pillar (p. 26)
- Security Pillar (p. 33)
- Reliability Pillar (p. 39)
- Performance Efficiency Pillar (p. 43)
- Cost Optimization Pillar (p. 46)

Operational Excellence Pillar

The operational excellence pillar includes the ability to run, monitor, and gain insights into systems to deliver business value and to continually improve supporting processes and procedures.

Topics

- Design Principles (p. 26)
- Best Practices (p. 27)
- Resources (p. 33)

Design Principles

In the cloud, there are a number of principles that can help you strengthen your ability to optimize operational aspects of your ML workloads. Having the ability to operationalize these workloads is critical in bringing ML workloads to market quickly.

AWS Operational Excellence best practices are designed to ensure ML workloads are operating efficiently in the cloud. For standard operational excellence practices that apply across all AWS workloads, refer to the Operational Excellence Pillar: AWS Well-Architected Framework whitepaper. The design principles for optimizing operational excellence for ML workloads include:

- **Establish cross functional teams**: To ensure ML workloads have a path to production, include cross-functional and domain expertise on project teams. Include all stakeholders required to develop, deploy, and support your ML workload.
- **Identify the end-to-end architecture and operational model early:** Early in the ML development lifecycle, identify the end-to-end architecture and operational model for model training and hosting. This allows for early identification of architectural and operational considerations that will be required for the development, deployment, management and integration of ML workloads.

- **Continuously monitor and measure ML workloads:** Identify and regularly collect key metrics related to the training, hosting, and predictions made against a model. This ensures that you are able to continuously monitor the health of a deployed model across key evaluation criteria, such as system metrics, model latency, or detecting data drift.
- **Establish a model retraining strategy: A deployed model's performance and effectiveness may change over time.** Identify metrics that indicate when a model version's performance and effectiveness is meeting business objectives and create alerts at thresholds that indicate that a model needs to be retrained to trigger these activities. Alerts can trigger activities such as invalidating the current model, reverting to an older model version, retraining a new model based on new ground truth data, or data science teams refining your model retraining strategy.
- **Document machine learning discovery activities and findings:** Data science discovery and exploration tasks provide background and insight into the creation and evolution of machine learning models. Document these activities in a code package that is managed and versioned in source control.
- **Version machine learning inputs and artifacts:** Versioned inputs and artifacts enable you to recreate artifacts for previous versions of your ML workload. Version inputs used to create models, including training data and training source code, in addition to model artifacts. Also version the algorithms used, feature engineering source code, hosting configurations, inference code and data, and post processing source code.
- **Automate machine learning deployment pipelines:** Minimize human touch points in ML deployment pipelines to ensure that ML models are consistently and repeatedly deployed using a pipeline that defines how models move from development to production. Identify and implement a deployment strategy that satisfies the requirements of your use case and business problem. If required, include human quality gates in your pipeline to have humans evaluate if a model is ready to deploy to a target environment.

Best Practices

There are three best practice areas for operational excellence in the cloud.

Topics

- Prepare (p. 27)
- Operate (p. 30)
- Evolve (p. 31)

Prepare

To prepare for operational excellence you have to understand your workloads and their expected behaviors. To prepare for operational readiness in ML workloads, you need to evaluate:

- Operational priorities
- Design for operations
- Operational Readiness

MLOPS 01: How have you prepared your team to operate and support a machine learning workload?

ML workloads are often different from a support perspective because the teams required to integrate with and deploy ML models may be unfamiliar with operational aspects specific to ML workloads. Best practices for ensuring ML models are effectively integrated into production environments and meet

business objectives include ensuring cross-collaboration between teams and training all resources responsible for supporting and maintaining machine learning workloads at base proficiency levels.

There are often operational requirements that must be considered for a ML workload that a data scientist may not accommodate, such as the ability to scale or to model latency. In contrast, there are also specific model behaviors that need to be captured that operations personnel may not be able to evaluate the measurement of, such as model effectiveness over time.

When considering your approach to preparing teams to integrate and operate ML workloads, key practices include:

- Provide high-level cross training between teams that will develop models and APIs, and teams that will support or have audit responsibility for your ML workload.
- Establish cross functional teams to ensure models and APIs can be effectively integrated into a production solution. This removes obstacles that can commonly block ML workloads from being deployed and integrated with a production solution.

MLOPS 02: How are you documenting model creation activities?

The ML model development lifecycle is significantly different from an application development lifecycle due in part to the amount of experimentation required before finalizing a version of a model. To add clarity for supporting and working with the model version, document the model creation process especially as it relates to assumptions made, the data pre/post processing required for the model as well as for integrating systems or applications with the model version.

Documenting this process provides transparency into the model for other stakeholders that are responsible for integrating with and supporting the model. Storing this documentation in a secure, versioned location, such as a source control repository, also captures the intellectual property related to the creation and evolution of the model.

In AWS, Amazon SageMaker Notebooks and Amazon SageMaker Studio provide managed notebook environments where data scientists can document their development process and experiments. These notebooks can be integrated with source control systems and become a standard part of the documentation created for each model deployed.

MLOPS 03: How are you tracking model lineage?

As you iteratively develop your ML models using different algorithms and hyperparameters for each algorithm, many model training experiments and model versions result. Tracking these models and tracing any given model's lineage is important not only for auditing and compliance, but also to perform root cause analysis of degrading model performance.

Additionally, synchronizing model lineage with data lineage is important because as versions of both data processing code and model versions are generated, the full data pipeline for training of each model version needs to be documented to enable debugging model errors and for compliance audits.

In AWS, Amazon SageMaker Experiments allows you to organize and track iterations of ML models. Amazon SageMaker Experiments automatically captures input parameters, configurations, and output artifacts for each model and stores them as experiments. This avoids using manual tracking or building custom tracking solutions to manage the numerous versions of input and output artifacts created and utilized for each iteration of model development. This approach allows teams to easily choose and deploy the model with the most optimal results across a number of experiments.

Example

MLOPS 04: How have you automated the development and deployment pipeline for your ML workload?

Create an operational architecture defining how your ML workload will be deployed, updated, and operated as part of its design. Incorporating common practices of Infrastructure-as-Code (IaC) and Configuration-as-Code (CaC) allows for consistency in deployments as well as the ability to reliably re-create resources across environments. Additionally, ensuring there is an automated mechanism to orchestrate the movement of an ML workload across phases and target environments in a controlled manner reduces risk when updating your workload.

Incorporate Continuous Integration and Continuous Delivery (CI/CD) practices to ML workloads (MLOps) to ensure that automation includes traceability along with quality gates. For example, CI/CD pipelines start with source and artifact version control supporting standard change management activities, as well as providing higher degrees of confidence to debugging activities. Applying the practice of source, data, and artifact version control to ML workloads improves operational debugging activities by enabling traceability back to the versions deployed. Additionally, versioning allows for the ability to rollback to a specific known working version following a failed change, or when a new model fails to provide needed functionality.

Implementing logging and monitoring throughout the CI/CD pipeline also establishes the foundation for inserting quality gates, either allowing or denying deployment into higher level environments. Best practices include standard quality gates, such as checking containers for package vulnerabilities and ensuring ML-specific quality gates are included in the pipeline. These quality gates should evaluate models using metrics identified and specific to your business use case. These could include metrics such as evaluating precision-recall, F1, or accuracy. Injecting quality gates helps ensure that a newer version of the model does not replace a current deployed model when a condition is identified that indicates an operational concern, such as a security exposure or a decrease in model performance or accuracy metrics.

In AWS, AI Services such as Amazon Polly are provided via an API endpoint. As a result, there are no unique best practices in this area as the model is already trained and deployed. Development and deployment automation related to the code and systems that interface with that API endpoint should follow standard AWS best practices. Some AWS AI services, such as Amazon Personalize, train a model based on training data that you provide. Follow the best practices described in this whitepaper for securing and protecting your data when creating or updating models in these services.

When building and training your own ML models in AWS, automating your development and deployment pipeline is achieved through a combination of AWS services and third-party integrations. Identifying the correct service or tools to use for creating an automated pipeline for model deployment relies on identifying the deployment strategy, model characteristics, and model training strategy.

Each ML workload varies based on the AWS ML services being used. However, a general guideline for creating pipelines includes using an orchestration layer, such as AWS CodePipeline, combined with logic that is responsible for executing the stages within the pipeline. Use AWS Lambda to create and execute the function-based logic due to its low operational overhead with no servers to manage. The following figure represents a reference pipeline for deployment on AWS. However, this deployment will vary depending on the factors previously discussed.

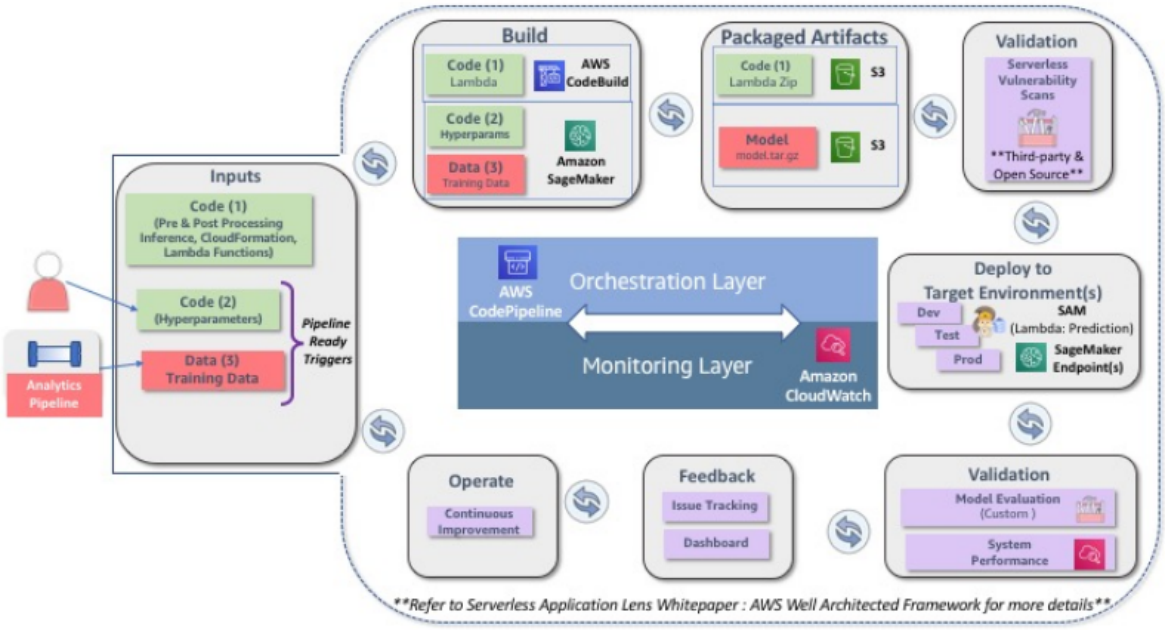

Figure 12 – Reference MLOps CI/CD Pipeline for Machine Learning on AWS

Operate

MLOPS 05: How are you monitoring and logging model hosting activities?

When hosting a model endpoint for predictions, the endpoint should have monitoring and alerts established to identify and react to any potential issues or opportunities for improvement. Model endpoints should include monitoring and alerting on metrics that measure the operational health of the underlying compute resources hosting the endpoint as well as monitoring the health of endpoint responses.

In AWS, the standard practices for managing operational health of endpoint compute resources should include those already defined in the AWS Well Architected: Operational Excellence whitepaper. Amazon SageMaker automatically monitors core system metrics and also includes capabilities to setup automatic scaling capabilities for your hosted model to dynamically adjust underlying compute supporting an endpoint based on demand. This capability ensures that your endpoint can dynamically support demand while reducing operational overhead.

In addition to monitoring compute resources in support of automatic scaling, Amazon SageMaker also outputs endpoint metrics for monitoring the usage and health of the endpoint. Amazon SageMaker Model Monitor provides the capability to monitor your ML models in production and provides alerts when data quality issues appear. Best practice includes creating a mechanism to aggregate and analyze model prediction endpoint metrics using services, such as Amazon Elasticsearch Service with built-in support for Kibana for dashboards and visualization. Additionally, having the ability to ensure traceability of hosting metrics back to versioned inputs allows for analysis of changes that could be impacting current operational performance.

Evolve

MLOPS 06: How do you know when to retrain ML models with new or updated data?

ML workloads can initially provide high value predictions but the accuracy of that same model's predictions can degrade over time. Often this is due to a concept known as drift, which can be a result of many factors that include changes to ground truth data over time. As model predictions are integrated into business decisions, this can indirectly affect the performance of existing models. For example, take a retail scenario predicting the risk associated with a particular shipment, where the training data includes past damaged shipments. As the business starts using the model to make business decisions, this indirectly affects the data as there will be fewer instances of damaged products.

There is often a need to retrain a model using new or updated data to ensure that the model is able to effectively learn and predict based on the most current data available. To be able to effectively incorporate additional data into a ML model, there must be a mechanism implemented to analyze existing model performance against defined metrics and trigger an alarm or a retraining event when model variance reaches a specific threshold, or proactively retrain the model over time based on new known data.

Best practices for taking additional data into consideration include:

- Define metrics that are indicative of model performance and accuracy
- Ensure that there is a mechanism in place to regularly capture those metrics for analysis and alert based on metric thresholds. For example, there may need to be a system that can identify, capture, or track downstream results back to specific model predictions so that metrics, such as error rates, can be calculated over time.
- Assess whether it's appropriate to retrain the model. Identify whether additional ground truth data is available or can be acquired, or whether additional data needs to be labeled. Decide on an initial strategy for retraining based on known workloads traits, such as regularly scheduled training with new data, new data as a trigger to retraining, or evaluate retraining based on metric thresholds. The strategy should evaluate tradeoffs between the amount of change, the cost of retraining, and the potential value of having a newer model in production. Set up automated retraining based on the strategy defined.

In AWS, AI Services such as Amazon Translate are automatically trained on new data so that you are able to take advantage of a model that is updated by AWS to improve model performance over time.

When using ML Services on AWS to build and train your own models, AWS provides multiple capabilities to support ongoing model retraining with new data. Store prepared data used for training in Amazon S3. The following retraining scenarios are included and should be considered based on workload characteristics:

- **Model Drift (Metric Driven Retraining):** For ML workloads that are sensitive to variation, such as when data distribution deviates significantly from the original training data or there is an increase in out-of-sample data, setting up an automated mechanism to trigger retraining of a model based on a defined metric or the presence of new prepared data. On AWS, one mechanism to identify a drift in data includes utilizing Amazon SageMaker Model Monitor to detect when the distribution of data has changed. Detection of drift is made available through AWS CloudWatch metrics which can be used to automatically trigger retraining jobs.
- **Additional Training Data:** AWS supports mechanisms for automatically triggering retraining based on new data PUT to an Amazon S3 bucket. The preferred method to initiate a controlled execution of model retraining is to set up an ML pipeline that includes an event trigger based on changes to a source Amazon S3 bucket. To detect the presence of new training data in an S3 bucket, CloudTrail

combined with CloudWatch Events allows you to trigger an AWS Lambda function or AWS Step Functions workflow to initiate retraining tasks in your training pipeline. The following figure illustrates the practice showing AWS CodePipeline with ML Services:

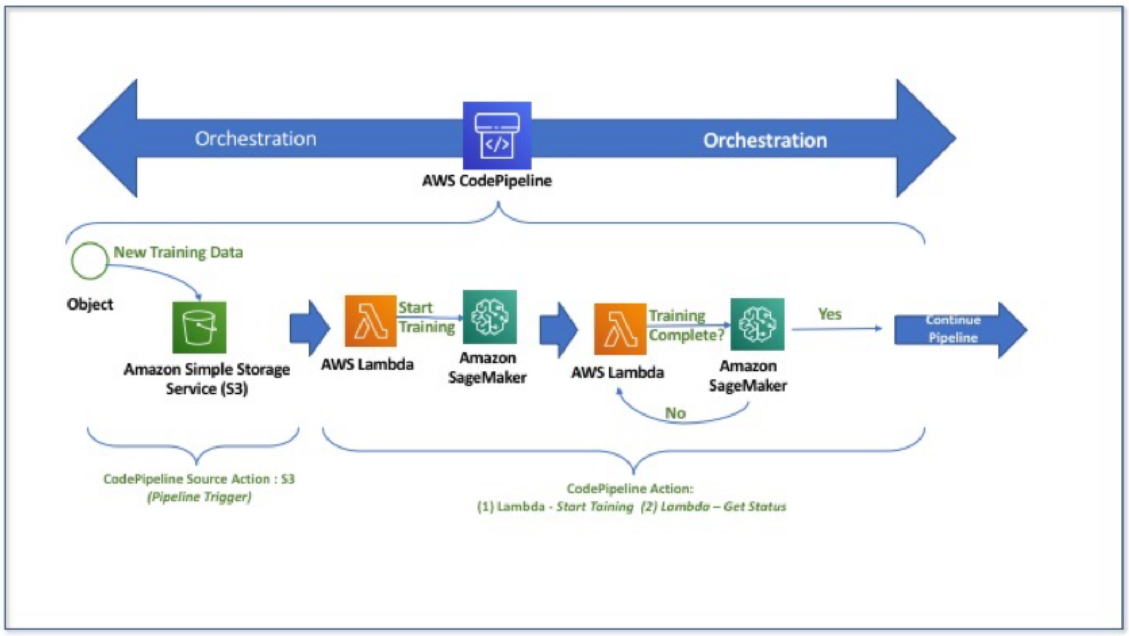

Figure 13 – Example event trigger for new training data for ML Services

Alternatively, you can also use third-party deployment orchestration tools, such as Jenkins, that integrate with AWS service APIs to automate model retraining when new data is available.

When defining your strategy for incorporating additional data into your models, ensure that the strategy supports model versioning that maintains all previous training data in its original form, or that previous versions of training data are easily reproduced. This ensures that if a model artifact is inadvertently deleted, the same model artifact can be re-created using the combined versions of all components used to create the versioned artifact.

MLOPS 07: How do you incorporate learnings between iterations of model development, model training, and model hosting?

To incorporate learnings, it's key to have an ongoing feedback mechanism in place that allows for the ability to share and communicate successful development experiments, analysis of failures, and operational activities. This facilitates the ability to continuously improve on future iterations of the ML workload.

Key considerations for learnings should include evaluation of the model across the following dimensions:

- **Business Evaluation:** To validate the success of a model against the business goal, you must ensure baseline business metrics exist as well as a mechanism to continually gather and monitor that information over time. As an example, if your business goal is to increase sales for a product by targeting specific customers for advertising campaigns, it's necessary to baseline and include an operational mechanism to continuously measure Key Performance Indicators (KPIs) of success, such as sales for a product, targeted customers, and customers purchasing the product.

- **Model Evaluation:** To validate the success of the model against the ML problem that you have framed, you must capture key metrics related to model performance through the end-to-end pipeline. This includes training metrics, such as training or validation errors, as well as ongoing metrics for a hosted model, such as prediction accuracy. The specific metrics should be chosen based on the use case and the business KPIs.
- **System Evaluation:** To validate system level resources used to support the phases of ML workloads, it's key to continuously collect and monitor system level resources such as compute, memory, and network. Requirements for ML workloads change in different phases. For example, training jobs are more memory intensive while inference jobs are more compute intensive.

In AWS, in addition to standard practices in this area, you can also utilize SageMaker notebook instances to capture data science exploration activities providing documentation and thorough explanation of the model development lifecycle. This is key not only in enabling you to successfully support a model in production but also to provide visibility and traceability of the activities of multiple data scientists and developers as models evolve. Additionally, providing centralized visibility to key operational metrics collected allows teams to continuously review and perform retrospective analysis of your operations over time.

Resources

Refer to the following resources to learn more about our best practices for operational excellence.

Documentation and Blogs

- Build end-to-end machine learning workflows with Amazon SageMaker and Apache Airflow
- Automated and continuous deployment of Amazon SageMaker models with AWS Step Functions
- Manage Amazon SageMaker with Step Functions
- Creating a pipeline with AWS CodePipeline & AWS Lambda

Whitepapers

- Operational Excellence Pillar – AWS Well-Architected Framework
- Serverless Applications Lens – AWS Well-Architected Framework

Security Pillar

The security pillar includes the ability to protect information, systems, and assets while delivering business value through risk assessments and mitigation strategies.

Topics

- Design Principles (p. 33)
- Best Practices (p. 34)
- Resources (p. 39)

Design Principles

In addition to the overall Well-Architected security design principles, there are specific design principles for ML security:

- **Restrict Access to ML systems:** The access level to ML systems should be taken into account when designing the system. Access to both the ML models and data sets used to train the model should be restricted to avoid data and model contamination. Inference endpoints should be secured such that only authorized parties can make inferences against the ML model.
- **Ensure Data Governance:** Data used for ML could be collected from multiple sources and needs to be available to various teams across your organization. Because production data is required for not only data science development activities but also for training models, ensuring teams have end-to-end access to high quality datasets requires a data governance strategy that ensures the integrity, security and availability of datasets. Implementing data lake solutions with governance and access controls ensures developers and data scientists have controlled access to quality data for use in exploration activities as well as training models. Data must also be guarded against exfiltration or mutation. Control what actions different teams in your organization can take on the data and where they can send it.
- **Enforce Data Lineage:** As data from various sources is used in different phases of the ML process, monitor and track your data's origins and transformations over time. Data lineage enables visibility and simplifies the process of tracing data processing and machine learning errors back to the root cause. Strictly control who can access the data and what they can do with it. Preventative controls, auditing, and monitoring are needed to demonstrate how data has been controlled during its lifetime.
- **Enforce Regulatory Compliance:** Regulatory concerns around ML systems include privacy considerations, such as those described in HIPAA or GDPR, where the ML system must adhere to the regulatory guidelines established in those frameworks. They can also include financial risk management concerns, such as the Federal Reserve's SR 11-7 guidelines. Unlike traditional models where the algorithms remain static, models leveraging ML/AI algorithms can evolve over time, so continued vigilance is necessary to ensure compliance with regulatory bodies.

Best Practices

There are five best practice areas for security in the cloud.

Topics

- Identity and Access Management (p. 34)
- Detective controls (p. 35)
- Infrastructure Protection (p. 35)
- Data Protection (p. 35)
- Incident Response (p. 38)

Identity and Access Management

MLSEC 01: How do you control access to your ML workload?

Typically, multiple teams are involved in building ML workloads with each team being responsible for one or more ML phases. Access to all resources used across the various phases of the ML process, including data, algorithms, hyperparameters, trained model artifacts and infrastructure, must be tightly controlled with least-privileged based access. For example, a team responsible for feature engineering may not be responsible for training or deploying the model, and therefore should not have permissions to do so. Similarly, an operations team responsible for deploying the model to production should not have permissions to access or modify training data. Some workloads may have team members with overlapping responsibilities across multiple phases of ML workloads and require appropriate permissions to perform role responsibilities.

In AWS, access to the various resources and services is controlled through AWS IAM. While Identities are used for authentication, fine grained control about who (humans) and what (processes) can access data, modify the data and algorithms, launch training jobs, and deploy models are implemented using IAM Users, Groups, Roles, and Policies.

Restrict access to a deployed model to only the intended legitimate consumers. For model consumers who are located within your AWS environment or have the means to retrieve temporary IAM credentials to access your environment, use an IAM role with least-privileged permissions to invoke the deployed model endpoint. For consumers who are external to your environment, provide access via a secure API using a combination of API Gateway and hosted model endpoints.

Detective controls

See the AWS Well-Architected Framework whitepaper for best practices in the detective controls area for security that apply to ML.

Infrastructure Protection

See the AWS Well-Architected Framework whitepaper for best practices in the infrastructure protection area for security that apply to ML.

Data Protection

MLSEC 02 : How are you protecting and monitoring access to sensitive data used in your ML workloads?

In the ML process, data is used in all phases. Early in a project, after identifying the business goals, you evaluate the accessibility and availability of various data sources, and interact with available data. Before the ML portion of a project can begin, typically a centralized data lake already exists or is built. Protect data in your data lake at rest and as it moves through the different phases of your ML process. All teams in your organization do not require access to all data. Classify the data, implement least privileged based granular access controls to various portions of the data, and continuously monitor access to the data.

In AWS, a centralized data lake is implemented using AWS Lake Formation on Amazon S3. Securing and monitoring a data lake on Amazon S3 is achieved using a combination of various services and capabilities to encrypt data in transit and at rest and monitor access including granular AWS IAM policies, S3 bucket policies, S3 access logs, Amazon CloudWatch, and AWS CloudTrail. Building Big Data Storage Solutions (Data Lakes) for Maximum Flexibility discusses using these various capabilities to build a secure data lake.

In addition to implementing access control through AWS IAM, use Amazon Macie for protecting and classifying data in Amazon S3. Amazon Macie is a fully managed security service that uses machine learning to automatically discover, classify, and protect sensitive data in AWS. The service recognizes sensitive data, such as personally identifiable information (PII) or intellectual property, and provides visibility into how this data is being accessed or moved. Amazon Macie continuously monitors data access activity for anomalies, and generates detailed alerts when it detects a risk of unauthorized access or inadvertent data leaks.

As data moves from the data lake onto compute instances, either for exploration or training, ensure that access to the destination compute instances is also tightly controlled. Once again, encrypt data in transit to and at rest on the compute infrastructure.

During the Data Preparation and Feature Engineering phases, there are multiple options for secure data exploration on AWS. Data can be explored in a managed notebook environment hosted by Amazon SageMaker or on an Amazon EMR notebook. You can also use managed services, such as Amazon Athena and AWS Glue, to explore the data without moving the data out of the data lake on Amazon S3.

A combination of the two approaches can also be used. Use a Jupyter notebook hosted on an Amazon SageMaker notebook instance to explore, visualize. and feature engineer a small subset of data and then scale up the feature engineering using a managed ETL service, such as Amazon EMR or AWS Glue.

When you use a Jupyter notebook hosted on an Amazon SageMaker notebook instance, deploy the notebook instance in an Amazon VPC, which enables you to use network level controls to limit communication to the hosted notebook. Additionally, network calls into and out of the notebook instance can be captured in VPC flow logs to enable additional visibility and control at the network level. By deploying the notebook in your VPC, you will also be able to query data sources and systems accessible from within your VPC, such as relational databases on Amazon RDS or Amazon Redshift data warehouses. Using IAM, you can further restrict access to the web-based UI of the notebook instance so that it can only be accessed from within your VPC.

To communicate with data stored in your data lake in Amazon S3 from the notebook instance within your VPC, use VPC interface endpoint connectivity. This ensures that communication between your notebook instance and Amazon S3 is conducted entirely and securely within the AWS network. Encrypt data at rest on your notebook instances by encrypting the EBS volumes attached to the Amazon SageMaker notebook instance using an AWS KMS-managed key.

The Jupyter notebook server provides web-based access to the underlying operating system, which gives developers and data scientists the ability to install additional software packages or Jupyter kernels to customize the environment. By default, a user has permissions to assume local root permissions, giving them total control of the underlying EC2 instance. This access can be restricted to remove the user's ability to assume root permissions but still give them control over their local user's environment.

In addition to restricting access to root permissions, use Lifecycle Configurations to manage Jupyter notebook instances. Lifecycle configurations are shell scripts which run as root when the notebook instance is first created or when the notebook instance is starting. They enable you to install custom tooling, packages, or monitoring. Lifecycle configurations can be changed and reused across multiple notebook instances so that you can make a change once and apply the new configuration to managed notebook instances by restarting them. This gives IT, operations, and security teams the control they need, while supporting the needs of your developers and data scientists.

When training a model, more compute power than what a single notebook instance can provide is often required. In AWS, you can use Amazon SageMaker to train models on a cluster of training instances. Amazon SageMaker provisions the underlying infrastructure used to execute your training jobs, running your algorithms against your data to produce a trained model.

Launch your cluster of training instances in your VPC, which allows you to apply network-level controls to training instances and grant access to AWS services including Amazon S3 and Amazon ECR via VPC endpoints. Restrict training jobs access to data sources not hosted on AWS services within your VPC using security groups. Also control network access beyond your VPC, using proxy servers and security groups. Encrypt data on your training node's EBS volumes using KMS-managed encryption keys to provide further protection for sensitive data during training. Use Amazon SageMaker control plane VPC endpoints to enable private communication between your VPC and the Amazon SageMaker control plane to manage and monitor training jobs.

When training your model on a cluster of instances, also accommodate information exchanged by algorithms during this process. It is common, as part of a distributed training job, for frameworks like TensorFlow to share information like coefficients. This is not your training data; instead, it is the information that the algorithms require to stay synchronized with one another. This data is not always encrypted by default. As part of a distributed training job, configure Amazon SageMaker to encrypt inter-node communication for your training job. Data transferred between these nodes is then encrypted in transit.

Along with securing the hosted Jupyter notebook environment and the training clusters, it's also crucial to secure the ML algorithm implementations. Amazon SageMaker uses container technology to train and host algorithms and models. This enables Amazon SageMaker and other ML partners to package algorithms and models as containers that you can then use as part of your ML project. In addition, you

can package any technology, language, or framework for use with Amazon SageMaker. When creating your own containers, publish them to a private container registry hosted on AWS Elastic Container Repository (ECR), and encrypt containers hosted on Amazon ECR at rest using a KMS managed key.

During training, Amazon SageMaker retrieves the container you specify from Amazon ECR and prepares it for execution on a training instance. For smaller data sets, Amazon SageMaker supports “File” mode for training, which downloads the training data from S3 bucket to the EBS volume attached to the training instance. This allows the algorithm to read its training data from the local file system, without integrating directly with Amazon S3. By using containers and copying objects from Amazon S3, Amazon SageMaker enables network isolation of your algorithms and models during training and hosting.

However, if you have large training data sets, copying to a local file system prior to starting a training job is inefficient. For this situation, use the “Pipe” mode of Amazon SageMaker, which streams data directly from Amazon S3 into the training instance. This mean your training jobs start sooner, finish quicker, and need less disk space, reducing your overall cost to train ML models on Amazon SageMaker.

Logs generated during training by Amazon SageMaker are logged to AWS CloudWatch Logs. Use an AWS KMS-managed encryption key to encrypt log data ingested by AWS CloudWatch Logs.

Data protection is important not only for the training data, but also for the production/live data that is used for inference. An example is an inference API call made to either an AWS AI service or a hosted model endpoint on Amazon. The HTTPS requests for these API calls should be signed, so that the requester identity can be verified, and request payload data is protected in transit and against potential replay attacks. When you use the AWS Command Line Interface (AWS CLI) or one of the AWS SDKs to make the API call, these tools automatically sign the requests for you with the access key that you specified when you configured the tools. However, if you write custom code to send HTTPS requests to AWS, you need to implement the functionality to sign the requests.

Additionally, AWS AI Services, such as Amazon Translate and Amazon Comprehend, have provisions to use your data for continuous development and improvement of AWS and affiliate ML and AI technologies. You may opt out of having your data used for these purposes by contacting AWS Support. After you receive confirmation that your account has been opted out and have followed any provided instructions, your content will no longer be stored or used to develop or improve the AWS AI Service or any Amazon ML/AI technologies.

MLSEC 03: How are you protecting trained ML models?

In addition to protecting data used to train an ML model, secure access to the model artifact generated by the training process. Host your model so that a consumer of the model can perform inference against it securely. Consumers of an ML model, which could be an internal or external applications or users, typically integrate with it via a simple endpoint or API that can provide predictions.

In AWS, an ML model generated at the end of the training phase is typically persisted in Amazon S3. Upload models trained within your VPC to Amazon S3 using a private VPC endpoint. This ensures that the model is transferred to Amazon S3 securely within the AWS network. When a model is trained using Amazon SageMaker, the service encrypts model artifacts and other system artifacts in transit and at rest.

Amazon SageMaker deploys and hosts a trained model on a cluster of inference compute nodes and provides an endpoint (HTTPS URL) to perform inferences against. Amazon SageMaker hosted endpoints support both real-time inferences and batch transform predictions. In both cases, the hosted endpoints enable the same VPC-based network protections, network isolation of the container hosting the model, and encryption of the inference node’s EBS volumes.

Amazon SageMaker hosted endpoints provide the added security of protecting your models and invocations using IAM. This enables you to control which IAM users, IAM roles, source VPCs, or IPs can perform inference against your model. In addition, you can use AWS PrivateLink to securely share your model as a service to other consumers.

Amazon SageMaker logs model inference activities to AWS CloudWatch Logs, similar to the logs captured as part of training. Again, ensure logs ingested by AWS CloudWatch Logs are encrypted using a KMS-managed encryption key. This provides you with a log of the activity of your models during inference and enables you to provide as much detail as you require to meet your security and auditability requirements.

Consumers of an ML model often make predictions against the model from applications external to the environment hosting the model, for example, a web application might make inferences against an internet-facing endpoint. The following diagram shows a serverless architecture for accessing a model hosted on Amazon SageMaker. In this architecture, an API Gateway is directly accessed by end users, while the AWS Lambda and the Amazon SageMaker model endpoint are operated in a protected private network.

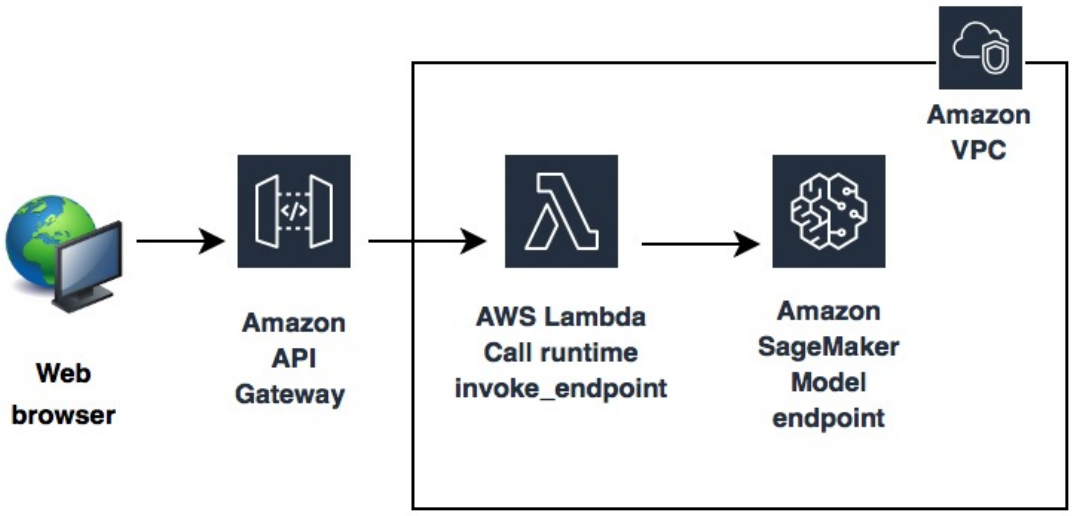

Figure 14 – Serverless Architecture for inference.

High level steps in this architecture are:

1. A consumer application invokes the API Gateway API with request parameter values.
2. API Gateway passes the parameter values to the Lambda function. The Lambda function parses the value and sends it to the Amazon SageMaker model endpoint.
3. The model performs the prediction and returns the predicted value to AWS Lambda. The Lambda function parses the returned value and sends it back to API Gateway.
4. API Gateway responds to the client with the inference value.

For the complete use case served by this architecture, see Call an Amazon SageMaker model endpoint using Amazon API Gateway and AWS Lambda.

Incident Response

See the AWS Well-Architected Framework whitepaper for best practices in the incident response area for security that apply to ML.

Key AWS Services

Key AWS services for data security and monitoring on AWS are:

- AWS IAM
- Amazon Virtual Private Cloud (Amazon VPC) and VPC Endpoints

- Amazon SageMaker

Resources

Refer to the following resources to learn more about our best practices for security on AWS.

Whitepapers

- AWS Security Best Practices
- Building Big Data Storage Solutions (Data Lakes) for Maximum Flexibility

Documentation and Blogs

- OWASP Secure Coding Best Practices
- Authentication and Access Control for Amazon SageMaker
- Call an Amazon SageMaker model endpoint using Amazon API Gateway and AWS Lambda
- Build a serverless frontend for an Amazon SageMaker endpoint

Reliability Pillar

The reliability pillar includes the ability of a system to recover from infrastructure or service disruptions, dynamically acquire computing resources to meet demand, and mitigate disruptions such as misconfigurations or transient network issues.

Topics

- Design Principles (p. 39)
- Best Practices (p. 39)
- Resources (p. 43)

Design Principles

In the cloud, there are a number of principles that can help you strengthen the reliability of your system. For standard practices, refer to the Reliability Pillar : AWS Well-Architected Framework whitepaper. There are additionally principles designed to help increase reliability specifically for ML workloads:

- **Manage changes to model inputs through automation:** ML workloads have additional requirements to manage changes to the data that is used to train a model to be able to recreate the exact version of a model in the event of failure or human error. Managing versions and changes through automation provides for a reliable and consistent recovery method.
- **Train once and deploy across environments:** When deploying the same version of an ML model across multiple accounts or environments, the same practice of build once that is applied to application code should be applied for model training. A specific version of a model should only be trained once and the output model artifacts should be utilized to deploy across multiple environments to avoid bringing in any unexpected changes to the model across environments.

Best Practices

There are three best practice areas for reliability in the cloud.

Topics

- Foundations (p. 40)
- Change Management (p. 40)
- Failure Management (p. 42)

Foundations

There are no foundational practices unique to ML workloads that belong to this sub-section. Practices identified in the Reliability Pillar of the AWS Well-Architected Framework whitepaper should be used for ensuring foundational capabilities.

Change Management

MLREL 01: How do you manage changes to your machine learning models and prediction endpoints?

For ML workloads, it's important to create a mechanism to enable traceability of changes made to your model as well as changes to prediction endpoints. This allows for faster troubleshooting and reverting to a previous model version if a newer model is not performing as expected. A deployed version of a model should be traceable back to a specific versioned model artifact that is protected in an artifact repository with read-only access to limited resources. Retain model artifacts using a retention period defined by the business.

To reduce overhead and manual intervention, changes to a model or endpoint should be automated through a pipeline that includes integration with any change management tracking systems as required by the business. Including traceability through versioned pipeline inputs and artifacts allows you to track changes and automatically rollback after a failed change.

To deploy changes to a model, it's recommended that you use standard A/B testing strategies, where a defined portion of traffic is directed to the new model while directing the remaining traffic to the old model. In this case, the rollback includes a DNS change back to the older version. To effectively identify when a rollback or rollforward is required, metrics that evaluate model performance must be implemented to alert when rollback or rollforward actions are required. When architecting for rollback or rollforward, it's key to evaluate the following for each model:

- Where is the model artifact stored?
- Are model artifacts versioned?
- What changes are included in each version?
- For a deployed endpoint, what version of the model is deployed?

Building traceability mechanisms to trace back resources and automatically deploy your models provide for reliable rollback and recovery capabilities. Additionally, to ensure that a model can be deployed reliably to multiple environments, a train-once strategy should be used to reduce any inadvertent variability in the deployment process.

When creating models on AWS, it's recommended that you use existing service capabilities as well as implement standards ensuring that ML models can be restored to a previous version.

For AI Services on AWS, such as Amazon Transcribe, AWS performs version control on the deployed endpoints that are used for making endpoint predictions. AWS is responsible for the change management related to hosting the endpoint as a service.

There are common standards for ML Services on AWS and ML Frameworks and interfaces on AWS to provide traceability in change management as well as ensure roll forward and roll back capabilities. Store model artifacts as versioned objects in Amazon S3 to ensure model durability and availability. Ensure that container images used for model training and model hosting are stored in a durable and secure image repository, such as AWS Elastic Container Registry (ECR). Additionally, protect the integrity of model artifacts and container images by restricting access to model artifacts using IAM role-based access and implementing minimum privileges on policies applied to resources. Store all configuration used to create an artifact as code in a managed source control system, such as AWS CodeCommit.

In addition, having traceability of the artifacts enables you to roll forward or rollback to a specific version. Create and maintain a manifest of model artifact version history highlighting changes between model artifact versions deployed. This can be accomplished by storing change manifest data in a persistent store, or optimally through an automated deployment pipeline that controls end-to-end model development and deployment as described later in ***Failure Management***. Use A/B testing capabilities native to SageMaker through production variants when possible for evaluating and quickly reacting to changes across multiple models.

For ML Frameworks and Interfaces on AWS, a number of capabilities are provided to create reusable design standards to increase automation and recoverability of your workloads in the form of AWS CloudFormation templates and AWS Developer Tools. Create a deployment strategy supporting rollback and roll forward capabilities as outlined in the Reliability Pillar of the AWS Well-Architected Framework whitepaper to evaluate and quickly react to changes across multiple models.

Implementing automatic rollback and roll forward capabilities allows you to recover from a failed change, system failure, or degraded model performance. This capability requires a well-defined versioning strategy combined with a mechanism to track and revert changes when issues are detected. Ensure that all metrics critical to model evaluation are defined and collected. Collect metrics in a monitoring system, such as Amazon CloudWatch, with alarms set to trigger rollback events in the event that a model version is not performing as expected.

MLREL 02: How are changes to ML models coordinated across your workload?

To ensure that changes to an ML model are introduced with minimal or no interruption to existing workload capabilities, it's important to accommodate how dependent systems and applications will integrate when designing interfacing applications. Flexible application and API design helps abstract change from interfacing applications. In addition, having a strategy for communicating and coordinating changes to dependent systems and/or applications is essential. Follow a defined change management strategy to introduce changes and communicate them to impacted teams and enable traceability of those changes.

Manage the deployment of new model versions in the same way that application level changes are governed and controlled. A change management strategy for ML models must account for how changes are communicated and deployed to help avoid interruptions in service and degradation of model performance and accuracy. Your strategy for deploying new model versions must also include validation activities to perform prior to and after deployment into a target environment.

To ensure that change management is consistently and correctly executed, it's a best practice to execute all changes through a CI/CD pipeline with access controls that follow the principle of least privilege to enforce your deployment process. Controlling deployments through automation combined with manual or automated quality gates ensures that changes can be effectively validated with dependent systems prior to deployment.

MLREL 03: How are you scaling endpoints hosting models for predictions?

It's critical to implement capabilities that allow you to automatically scale your model endpoints. This ensures that you can reliably process predictions to meet changing workload demands. To scale your endpoints, you must include monitoring on endpoints to identify a threshold that triggers the addition or removal of resources to support current demand. Once a trigger to scale is received, a solution must be in place to scale backend resources supporting that endpoint. Perform load testing against endpoints to be able to validate their ability to scale effectively and serve predictions reliably.

In AWS, endpoint scaling and responsibility in this area is dependent on the AI/ML service being utilized. For AWS AI services including Amazon Comprehend, Amazon Polly, and Amazon Translate, the endpoints are managed and scaled automatically by AWS. For AWS ML Services, such as Amazon SageMaker, automatic scaling capabilities across Availability Zones is configurable within the service. For high availability, configure automatic horizontal scaling across multiple Availability Zones for all production variants. Once configured, it's critical to perform failure testing to ensure that your endpoint can recover from failure and support availability requirements.

For AWS ML Frameworks and Interfaces, setup automatically scaled, load balanced capabilities regardless of whether the model is being hosted on EC2 instances or using containers on ECS or EKS that are hosted on EC2 instances or AWS Fargate. Automatic scaling capabilities can also be used to self-heal EC2 instances by automatically replacing an unhealthy instance. Refer to the Reliability Pillar in the AWS Well Architected Framework for standard best practices in this area.

Failure Management

MLREL 04: How do you recover from failure or inadvertent loss of a trained ML model?

A trained ML model is a packaged artifact that must be recoverable in the event of a failure or loss. A failure or loss of resources can be caused by multiple events ranging from system failure to human error. For ML models, the following failure scenario should be considered and compared with your recovery objectives to ensure that the appropriate strategy is deployed. If a model artifact is inadvertently deleted because of human error or the underlying storage becomes unavailable, can you easily recover or re-create that artifact?

The ability to protect a model artifact from inadvertent deletion can be achieved by ensuring that the model artifact is protected by allowing only minimum required privileges to use the artifact, implementing additional mechanisms such as MFA for delete by privileged users, and storing a secondary copy of the artifact as required by your defined Disaster Recovery strategy. In parallel, implementing an artifact versioning strategy allows for recovery of the specific versioned artifact. Secondarily, the ability to re-create a specific version of a model artifact provides additional protection from failure or loss. Apply the same protection mechanisms and versioning strategy to model inputs including data and training code.

In AWS, the best practices related to model failure and recoverability vary depending on the AWS service used. AWS AI services, such as Amazon Polly, use pre-built models that are protected and managed by AWS. AWS ML services, such as Amazon SageMaker, create and store model artifacts in Amazon S3. In this case, you can take advantage of access controls provided by AWS IAM to protect model inputs and artifacts for protection of resources. Additionally, you can use a mechanism, such as Amazon S3 versioning combined with object tagging for versioning and traceability of model artifacts, for recoverability in the event of failure.

MLREL 05: How do you recover from failure or inadvertent loss of model hosting resources?

Having the ability to recover any component in an ML workload ensures that the solution is able to withstand failure or loss of a resource. A failure or loss of resources can be caused by multiple events

ranging from system failure to human error. For ML workloads, the following failure scenarios should be considered and compared with your recovery objectives to ensure the appropriate strategy is deployed. If a model endpoint is inadvertently deleted, can that endpoint be re-created to recover the endpoint to a specific version?

In AWS, the best practices related to failure management varies depending on the AWS service used. AWS AI services, such as Amazon Polly, are hosted, scaled, and managed by AWS so that you are not responsible for endpoint recovery. For AWS ML Services and AWS ML Infrastructure and Frameworks, it's a best practice to ensure that an endpoint responsible for hosting model predictions is fully recoverable to a specific version or point in time as defined by your business. The ability to recover a model endpoint requires that all components and configuration used to create that endpoint are included in a managed version control strategy to enable complete recovery pending the unavailability of any component.

As an example, re-creating an endpoint in SageMaker requires exact versions of multiple components in SageMaker including: Model Artifacts, Container Images, and Endpoint Configurations. To re-create a specific model artifact version you also need to know the versioning strategy for the training data and the algorithm used to create that model artifact. To ensure that you have the ability to re-create any component in the pipeline in the event of failure, version all dependent resources as well. In addition to versioning, it's key to ensure that all versioned artifacts are included in a manifest documenting the deployment and all versioned artifacts are protected using the principle of least privilege as outlined in the Security Pillar.

Resources

Resources

Refer to the following resources to learn more about best practices for reliability.

Whitepapers

- Reliability Pillar - AWS Well Architected Framework

Performance Efficiency Pillar

The performance efficiency pillar focuses on the efficient use of computing resources to meet requirements and how to maintain that efficiency as demand changes and technologies evolve.

Topics

- Design Principles (p. 43)
- Best Practices (p. 44)
- Resources (p. 46)

Design Principles

In the cloud, there are a number of principles that can help you strengthen the performance efficiency of your system. For standard practices, refer to the Performance Efficiency Pillar: AWS Well-Architected Framework whitepaper. There are additionally a number of principles designed to help increase performance efficiency specifically for ML workloads:

- **Optimize compute for your ML workload:** Most ML workloads are very compute-intensive, because large amounts of vector multiplications and additions need to be performed on a multitude of data and parameters. Especially in Deep Learning, there is a need to scale to chipsets that provide larger queue depth, higher Arithmetic Logic Units and Register counts, to allow for massively parallel processing. Because of that, GPUs are the preferred processor type to train a Deep Learning model. We

will discuss the details pertaining to the selection of appropriate compute resource in section MLPER 01 below.

- **Define latency and network bandwidth performance requirements for your models:** Some of your ML applications might require near-instantaneous inference results to satisfy your business requirements. Offering the lowest latency possible may require the removal of costly roundtrips to the nearest API endpoints. This reduction in latency can be achieved by running the inference directly on the device itself. This is known as Machine Learning at the Edge. A common use-case for such a requirement is predictive maintenance in factories. This form of low latency and near real-time inference at the edge allows for early indications of failure, potentially mitigating costly repairs of machinery before the failure actually happens.
- **Continuously monitor and measure system performance:** The practice of identifying and regularly collecting key metrics related to the building, training, hosting, and running predictions against a model ensures that you are able to continuously monitor the holistic success across key evaluation criteria. To validate system level resources used to support the phases of ML workloads, it's key to continuously collect and monitor system level resources such as compute, memory and network. Requirements for ML workloads change in different phases as training jobs are more memory intensive while inference jobs are more compute intensive as discussed in the previous two design principles.

Best Practices

There are four best practice areas for performance efficiency in the cloud:

- Selection (compute, storage, database, network)
- Review
- Monitoring
- Tradeoffs

Take a data-driven approach to selecting a high-performance architecture. Gather data on all aspects of the architecture, from the high-level design to the selection and configuration of resource types. By reviewing your choices on a cyclical basis, you can ensure that you are taking advantage of the continually evolving AWS services. Monitoring will ensure that you are aware of any deviance from expected performance and can take action on it. Finally, your architecture can make tradeoffs to improve performance, such as using compression or caching, or relaxing consistency requirements.

Selection

MLPER 01: How are you choosing the most appropriate instance type for training and hosting your models?

A typical machine learning pipeline is comprised of a series of steps. The process starts with the collection of training and test data, followed by feature engineering and transformation of the data collected. After the initial data preparation phase, training the ML models, as well as evaluating and tuning them, leads to the final stage of deploying, serving, and monitoring your models' performance throughout its lifecycle. Performance of your ML workload in each of these phases should be carefully considered, since the computation needs of each phase differs. For example, while you may need a power cluster of GPU instances for model training, when it comes to inference, an automatic scaling cluster of CPU instances may be sufficient to meet your performance requirement.

Data size, data type, and the selection of algorithm can have a noticeable effect on which configuration is most effective. When training the same model repeatedly, it's highly recommended to perform initial testing across a spectrum of instance types to discover configurations that are performant and cost

effective. As a general guideline, GPU instances are recommended for the majority of deep learning purposes as training new models is faster on a GPU instance than a CPU instance. You can scale sub-linearly when you have multi-GPU instances or if you use distributed training across many instances with GPUs. However, it's important to note that algorithms that train most efficiently on GPUs might not necessarily require GPUs for efficient inference.

AWS provides a selection of instance types optimized to fit different machine learning (ML) use cases. Instance types have varying combinations of CPU, GPU, FPGA, memory, storage, and networking capacity. Additionally, can you attach a GPU-powered inference accelerator to your Amazon EC2 or Amazon SageMaker instances via Amazon Elastic Inference or use Amazon EC2 instances that are powered by AWS Inferentia, a high performance ML inference chip, custom designed by AWS. This provides you with the flexibility to choose the appropriate mix of resources optimized to fit your ML use cases, whether you are training models or running inference against trained models. Each instance type includes one or more instance sizes, allowing you to scale your resources to the requirements of your target workload.

Lastly, some algorithms, such as XGBoost, implement an open-source algorithm that has been optimized for CPU computations, while on the AWS Deep Learning AMI (DLAMI) some frameworks, such as Caffe, only work with GPU support and cannot be run in CPU mode.

Regardless of your instance selection for training and hosting, load test your instance or Amazon SageMaker endpoints to determine the peak load that your instance or endpoint can support, and the latency of requests, as concurrency increases.

MLPER 02: How do you scale your ML workload while maintaining optimal performance?

When considering how to scale your ML architecture for an increase in demand and for optimal performance, it's important to differentiate between deploying your models via the managed service experience versus deploying and managing ML models on your own.

On AWS, while the managed ML experience is provided by Amazon SageMaker, you typically use Deep Learning AMIs (DLAMI), which provides MXNet, TensorFlow, Caffe, Chainer, Theano, PyTorch, and CNTK frameworks, on EC2 instances to manage models on your own. This section discusses both use cases, while also briefly discussing a third use-case of using AWS AI services.

Amazon SageMaker manages your production compute infrastructure on your behalf to perform health checks, apply security patches, and conduct other routine maintenance, all with built-in Amazon CloudWatch monitoring and logging. Amazon SageMaker Model Monitor continuously monitors ML models in production, detects deviations such as data drift that can degrade model performance over time, and alerts you to take remedial actions

Additionally, Amazon SageMaker hosting automatically scales to the performance needed for your application using Application Auto Scaling. By using Application Auto Scaling, you can automatically adjust your inference capacity to maintain predictable performance at a low cost. In addition, you can manually change the number and type of EC2 instances without incurring downtime through modifying the endpoint configuration.

For your Deep Learning workloads, you have the option of scaling with Amazon Elastic Inference (EI), to increase throughput and decrease the latency for real-time inferences against your deep learning models. Amazon Elastic Inference enables you to attach GPU-powered inference acceleration to any Amazon EC2 instance. This feature is also available for Amazon SageMaker notebook instances and endpoints, bringing acceleration to built-in algorithms and to deep learning environments at reduced costs.

Deep learning neural networks are ideally suited to take advantage of multiple processors, distributing workloads seamlessly and efficiently across different processor types and quantities. With the wide range

of on-demand resources available through the cloud, you can deploy virtually unlimited resources to tackle deep learning models of any size. By leveraging distributed networks, deep learning in the cloud allows you to design, develop and train deep learning applications faster.

The AWS Deep Learning AMIs for Ubuntu and Amazon Linux support distributed training of TensorFlow deep learning models with near-linear scaling efficiency. The AWS Deep Learning AMIs come pre-built with an enhanced version of TensorFlow that is integrated with an optimized version of the Horovod distributed training framework. This optimization leads to high performance implementations that allow nodes to communicate directly with each other instead of going through a centralized node and average gradients using the ring-allreduce algorithm. There are many other frameworks that support distributed training, such as using Chainer or Keras, in addition to the mentioned Horovod distribution.

Using AWS AI services allows you to add intelligence to your applications through an API call to a pre-trained service, rather than developing and training your own models. Scaling an architecture that uses AWS AI services involves monitoring resource and rate limits such as API request rates. You can find detailed information on how you can manage service limits in the reliability section of the general Well-Architected Framework.

Resources

Refer to the following resources to learn more about our best practices for Performance Efficiency.

Documentation and Blogs

- Scalable multi-node training with TensorFlow
- Amazon Elastic Inference
- Load test and optimize an Amazon SageMaker endpoint using automatic scaling
- Selecting the Instance Type for DLAMI

Whitepapers

- Performance Pillar - AWS Well Architected Framework

Cost Optimization Pillar

The **cost optimization** pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your very first proof of concept to the ongoing operation of production workloads, adopting the practices in this paper will enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment.

Topics

- Design Principles (p. 46)
- Best Practices (p. 47)
- Resources (p. 51)

Design Principles

In the cloud, there are a number of principles that can help you improve the cost optimization of your system. For standard practices, refer to the AWS Well-Architected Framework whitepaper. There are

additionally a number of principles designed to help increase cost optimization specifically for ML workloads:

- **Use managed services to reduce cost of ownership:** Adopt appropriate managed services for each phase of the ML workload to take advantage of the only pay for what you use model. For example, model tuning is typically a compute and time intensive process. To avoid any unnecessary charges, use a managed service that creates a distributed training cluster, executes training jobs to tune the model, persists the resulting models, and decommissions the cluster automatically when training is complete.
- **Experiment with small datasets:** While ML workloads benefits from large high-quality training datasets, start with smaller datasets on a small compute instance (or your local system) to iterate quickly at low cost. After the experimentation period, scale up to train with the full dataset available on a distributed compute cluster. When training with full dataset opt for streaming data into the cluster instead of storing data on the cluster nodes.
- **Right size training and model hosting instances:** For model training and hosting, experiment to determine the optimal compute capacity needed. It's recommended that you start with smaller instances, scale out first, and then scale up. Also measure the difference between CPU vs GPU needs during training and hosting. While some ML models require high power GPU instance for training, inferences against the deployed model might not require the full power of a GPU.
- **Account for inference architecture based on consumption patterns:** Some models, such as ecommerce fraud detection, need to be available constantly for real-time predictions, while others, such as ecommerce forecasting models, might only need to be available periodically. In the first case, the cost of a 24 X 7 hosting model is justified, but significant cost savings can be realized in the second case by deploying the model on-demand, executing predictions, and then taking down the model.
- **Define overall ROI and opportunity cost:** Weigh the cost of adopting ML against the opportunity cost of not leaning on ML transformation. Specialized resources, such as data scientist time or model time-to-market, might be your most expensive and constrained resources. The most cost-effective hardware choice might not be the cost optimized if it constraints experimentation and development speed.

Best Practices

There are four best practice areas for cost optimization in the cloud:

Topics

- Cost-Effective Resources (p. 47)
- Matching Supply and Demand (p. 50)
- Expenditure Awareness (p. 50)
- Optimizing Over Time (p. 50)

As with the other pillars, there are tradeoffs to evaluate. For example, do you want to optimize for speed to market or for cost? In some cases, it's best to optimize for speed—going to market quickly, shipping new features, or simply meeting a deadline—rather than investing in upfront cost optimization. Design decisions are sometimes guided by haste as opposed to empirical data, as the temptation always exists to overcompensate "just in case" rather than spend time benchmarking for the most cost-optimal deployment. This often leads to drastically over-provisioned and under-optimized deployments. The following best practices provide techniques and strategic guidance for the initial and ongoing cost optimization of your deployment.

Cost-Effective Resources

MLCOST 01: How do you optimize data labeling costs?

Building an ML model requires developers and data scientists to prepare their datasets for training their ML models. Before developers can select their algorithms, build their models, and deploy them to make predictions, human annotators manually review thousands of examples and add the labels required to train the ML models. This process is time consuming and expensive.

In AWS, Amazon SageMaker Ground Truth simplifies data labeling tasks using human annotators through Amazon Mechanical Turk, third-party vendors, or their own employees. Amazon SageMaker Ground Truth learns from these human annotations in real time and applies active learning to automatically label much of the remaining dataset, reducing the need for human review. The combination of human and ML capabilities allows Amazon SageMaker Ground Truth to create highly accurate training data sets, saves time and complexity, and reduces costs when compared to solely human annotation.

MLCOST 02: How do you optimize costs during ML experimentation?

Notebooks are a popular way to explore and experiment with data in small quantities. Iterating with a small sample of the dataset locally and then scaling to train on the full dataset in a distributed manner is common in machine learning.

In AWS, Amazon SageMaker notebook instances provide a hosted Jupyter environment that can be used to explore small samples of data. Stop the notebook instances when you are not actively using them. Where practical, commit your work, stop them, and restart them when you need them again. Storage is persisted and you can use lifecycle configuration to automate package installation or repository synchronization.

While experimenting with training a model, use Amazon SageMaker notebook "local" mode to train your model on the notebook instance itself, instead of on a separate managed training cluster. You can iterate and test your work without having to wait for a new training or hosting cluster to be built each time. This saves both time and cost associated with creating a managed training cluster. Experimentation can also happen outside of a notebook, for example on a local machine. From your local machine, you can use the SageMaker SDK to train and deploy models on AWS.

When experimenting, also review AWS Marketplace for Machine Learning that offers an ever growing catalog of machine learning algorithms and models. Models from AWS Marketplace are deployed directly to Amazon SageMaker and allow you to build ML applications quickly. This saves you both the cost and time associated with model development.

Additionally, AWS Marketplace for Machine Learning enables you to sell the models you develop, thus providing an additional revenue stream to monetize in-house models. You can make your custom model available to other customers, while securing your intellectual property. Amazon SageMaker allows access to your models through secure endpoints without exposing the underlying models.

MLCOST 03: How do you select the most cost optimal resources for ML training?

When you are ready to train a ML model with the complete training data, avoid running the training jobs in the local mode on a notebook instance unless the data set is small. Instead, launch a training cluster with the one or more compute instances for distributed training. Right size the compute instances in the training cluster based on the workload.

In AWS, use the Amazon SageMaker Training API to create a cluster of managed instances. Using multiple instances in your training cluster enables distributed training which results in faster training time. All instances in the training cluster are automatically terminated when training is complete.

While a variety of instance types with different capacity configurations are available to use for training, it's important that you right size the training instances according to the ML algorithm used. Be aware

that simple models might not train faster on larger instances, because they might not be able to benefit from increased hardware parallelism. They might even train slower due to high GPU communication overhead. It's recommended that you start with smaller instances, scale out first, and then scale up. Additionally, if the ML algorithm of your choice supports checkpointing, evaluate using managed spot training with Amazon SageMaker to save cost.

In addition to choosing optimized instance types for training, selecting optimized versions of ML frameworks for faster training is important. AWS provides optimized versions of frameworks, such as TensorFlow, Chainer, Keras, and Theano, which include optimizations for high-performance training across Amazon EC2 instance families.

When dealing with large volumes of training data, Amazon SageMaker Pipe mode offers significantly better read throughput than the Amazon SageMaker File mode. While the File mode downloads data to the local Amazon EBS volume prior to starting the model training, Pipe mode streams data from Amazon S3 into Amazon SageMaker when training ML models. This means your training jobs start sooner, finish faster, and require less disk space, reducing your overall cost to train ML models on Amazon SageMaker.

Determining the correct set of hyperparameters for an ML model can be expensive. This process typically requires techniques such as grid search or random search that involve training hundreds of different models. Amazon SageMaker automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose. Hyperparameter tuning uses ML techniques that can quickly and efficiently determine the optimal set of parameters with a limited number of training jobs.

Additionally, a warm start of hyperparameter tuning jobs can accelerate the tuning process and reduce the cost for tuning models. Warm start of hyperparameter tuning jobs eliminates the need to start a tuning job from scratch. Instead, you can create a new hyperparameter tuning job based on selected parent jobs, so that training jobs conducted in those parent jobs can be reused as prior knowledge, thus reducing costs associated with model tuning.

Finally, evaluate Amazon SageMaker AutoPilot that automatically analyzes your data and builds a model, saving you time and cost. Autopilot selects the best algorithm from the list of high-performing algorithms and automatically tries different parameter settings on those algorithms to get the best model quality.

MLCOST 04: How do you optimize cost for ML Inference?

In line with model training, it's important to understand which instance type is a good fit for your workload. Start by taking into consideration latency, throughput, and cost. Once again, it's recommended that you start small, scale out first, and then scale up.

In addition to using ML compute instance automatic scaling for cost savings, measure the difference between CPU vs GPU. While deep learning ML models require high power GPU instance for training, inferences against the deep learning models do not typically need the full power of a GPU. As such, hosting these deep learning on a full fledged GPU might result in underutilization and unnecessary costs. Additionally, account for the inference architecture needed for your ML model. That is, decide whether the model can be deployed on-demand as a batch of inference requests are needed, or whether it must be available 24 X 7 for real-time predictions.

On AWS, Amazon SageMaker endpoints supports automatic scaling, enabling you to match supply and demand of resources. With automatic scaling in Amazon SageMaker, you can ensure that your model's elasticity and availability, as well as optimize the cost by selecting the right metrics to scale your inference endpoint.

While automatic scaling increases or decreases the number of instances behind an endpoint to match inference request volume, you can also increase the compute power of hosted instances by attaching a fractional GPU compute capacity to an instance. Amazon Elastic Inference allows you to attach low-cost GPU-powered acceleration to Amazon EC2 and Amazon SageMaker instances to reduce the cost of running deep learning inference.

While standalone GPU instances are a good fit for model training activities which requires processing hundreds of data samples in parallel, they are typically oversized for inference which consumes a small amount of GPU resources. Different models need different amounts of GPU, CPU, and memory resources. Selecting a GPU instance type to satisfy the requirements of the most demanding resource often results in under-utilization of the other resources and unnecessary costs.

With Amazon Elastic Inference, you can choose the Amazon EC2 instance type that is best suited to the overall CPU and memory needs of your model, and then separately configure the amount of inference acceleration that you need to use resources efficiently and to reduce the cost of running inference.

Some applications do not need online/real-time predictions and are better suited for periodic batch predictions. In this case, hosting an endpoint 24x7 is not required. These applications are a good fit for Amazon SageMaker Batch Transform. Instead of a single inference per request, batch transform generates inferences for an entire dataset. Batch transform manages all of the compute resources required to perform inferences. This includes launching instances and terminating them after the batch transform job has completed.

Sometimes performing inference for a given data requires the data to be pre-processed, post-processed, or both. This could involve chaining together inferences from multiple intermediate models before a final model can generate the desired inference. In this situation, instead of deploying the intermediate models on multiple endpoints, use Amazon SageMaker Inference Pipelines.

An *inference pipeline* is an Amazon SageMaker model composed of a linear sequence of containers that process requests for inferences on data. These inference pipelines are fully managed and can combine preprocessing, predictions, and post-processing. Pipeline model invocations are handled as a sequence of HTTPS requests. By deploying all relevant steps to the same endpoint, you can save costs and reduce inference latency.

When you have multiple models in production with each model deployed on a different endpoint, your inference costs go up in proportion with the number of models. However, if you have a large number of similar models that you can serve through a shared serving container, and don't need to access all the models at the same time, use the Amazon SageMaker multi-model endpoints capability. This enables you to deploy multiple trained models to an endpoint and serve them using a single container. You can easily invoke a specific model by specifying the target model name as a parameter in your prediction request. When there is a long tail of ML models that are infrequently accessed, using one multi-model endpoint can efficiently serve inference traffic and enable significant cost savings.

Matching Supply and Demand

See the AWS Well-Architected Framework whitepaper for best practices in the matching supply and demand area for cost optimization that apply to ML workloads.

Expenditure Awareness

See the AWS Well-Architected Framework whitepaper for best practices in the expenditure awareness area for cost optimization that apply to ML workloads.

Optimizing Over Time

See the AWS Well-Architected Framework whitepaper for best practices in the optimizing over time area for cost optimization that apply to ML workloads.

Resources

Refer to the following resources to learn more about our best practices for cost optimization.

Documentation and Blogs

- Amazon SageMaker Pricing
- Use the Amazon SageMaker local mode to train on your notebook instance
- Making the most of your Machine Learning budget on Amazon SageMaker
- Lowering total cost of ownership for machine learning and increasing productivity with Amazon SageMaker

Conclusion

Machine Learning opens up unparalleled opportunities for organizations enabling automation, efficiency, and innovation. In this paper, we revisited the five pillars of the Well-Architected Framework through the lens of ML to provide architectural best practices for building and operating reliable, secure, efficient, and cost-effective ML workloads in the AWS cloud. The best practices are discussed in the context of using AI services, managed ML services, and the ML frameworks on AWS, providing you with a choice of using the most appropriate option for your business goals. Use these best practices, composed as a set of questions, to review your existing or proposed ML workloads.

While operating ML workloads, ensure that cross functional teams are involved and automation of the end-to-end pipeline is in place. For reliability, ensure robust responses to system failures by taking advantage of self-healing capabilities of automatic scaling and keeping track of all artifacts through versioning so that a functional system can be rebuilt automatically.

ML workloads on AWS should be secured using authentication and authorization controls that tightly control who and what can access the various ML artifacts. A secure application will protect your organization's sensitive information assets and meet compliance requirements at every layer. To enable performant ML workloads you have a choice of serval instances types that are optimized to fit different ML requirements. Take a data driven approach to make a selection of CPU vs GPU instances, while keeping in mind the various combinations of CPU, GPU, FPGA, memory, storage, and networking capacity available. For cost optimization, take advantage of "only pay for what you use" and reduce unnecessary waste by sizing resources in accordance with data, training and inference demands.

The landscape of machine learning is continuing to evolve with the ecosystem of tooling and processes growing and maturing. As this occurs, we will continue to update this paper to help you ensure that your ML applications are well architected.

Contributors

Contributors to this document include:

- Sireesha Muppala, AI/ML Specialist SA, Amazon Web Services
- Shelbee Eigenbrode, AI/ML Solutions Architect, Amazon Web Services
- Christian Williams, Machine Learning Specialist SA, Amazon Web Services
- Bardia Nikpourian, Specialist TAM – AI/ML, Amazon Web Services
- Ryan King, Sr. TPM, AWS Managed Cloud, Amazon Web Services

Further Reading

For additional information, see the following:

- Managing Machine Learning Projects (AWS Whitepaper)

Document Revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

update-history-change	update-history-description	update-history-date
Initial publication (p. 55)	Machine Learning Lens first published.	April 16, 2020

Notices