C3 → Optimize ML Models and Deploy Human-in-loop Pipelines

Created	@February 24, 2022 9:22 AM
Tag	Code Trainning
Description	

⚠️ Third Course of Practical Data Science Specialization

Index

W1 → Advanced model training, tuning and evaluation

Advanced Model Training and Tuning

Model tuning is a part of model training process. The goal of model training process is to fit the model to the underlying data patterns in your training data and learn the best possible parameters for your model.

Once you have validated your choices of algorithm, code, and dataset to solve your machine learning use cases, you can leverage automatic model tuning to fine tune your hyperparameters to find the best performing values.

Automatic Model Tuning

Grid Search

tests every combination of the matrix that you provide containing parameters names and values in dict form usually.

+

- its good when there is small nº of parameters e/or small range of

-

- really time consuming on bigger combinations

values

- explores all combinations

- doesn’t scale well

Random Search

Like Grid Search , you need to send a matrix of parameters and values ranges. instead of evaluating all combinations it selects a random combination of parameters

+

- faster method

-

- might miss the best performing model

Bayesian Optimization

hyperparameter tuning is treated as a regression problem. The hyperparameter values are learned by trying to minimize the loss function of a surrogate model. Here, the algorithm starts with random values for the hyperparameters and continuously narrows down the search space by using the results from the previous searches.

+

- more efficient in fiding the best hyperparameters

-

- requires sequential execution
- might get stuck on local minimal

Hyperband

Hyperband is based on bandit approach. Bandit approaches typically use a combination of exploitation and exploration to find the best possible hyperparameters. The strength of the bandit approaches is that dynamic pull between exploitation and exploration. When applied to the hyperparameter tuning problem space, this is how the bandit-based hyperband algorithm works. You start with the larger space of random hyperparameter set and then you explore a random subset of these hyperparameters for a few iterations. After the first few iterations, you discard the worst performing half of the hyperparameter sets. In the subsequent few iterations, you continue to explore the best performing hyperparameters from the previous iteration. You continue this process until the set time is elapsed or you remain with just one possible candidate.

+	-
• spends time more efficiently	• it might discard good candidates very ealry on

SageMaker hyperparameter tunning (HPT)

It automatically searchs the best parameters by running multiple training jobs on your dataset using the hyperparameter range values that you specify.

you can use:

- random search method
- Bayesian method
- your own code / implementation

A warm start is particularly useful if you want to change the hyperparameter tuning ranges from the previous job, or if you want to add new hyperparameters to explore.

there are 2 types:

- **transfer_learning** : the new hyperparameter tuning job uses an updated training data and also can use a different version of the training algorithm (ex when you have more data now)
- **identical_data_and_algorithm**: the new hyperparameter tuning job uses the same input data and the training data and the training algorithm as the parent tuning job. You have a chance to update the hyperparameter tuning ranges and the maximum number of training jobs

Best Practices - SM HyperParameter Tuning

- select a small number of parameters
- select a small range of values for each parameter
- enable warm start: when you enable warm start, the hyperparameter tuning job uses results from previously completed jobs to speed up the optimization process and save you the tuning cost
- enable early stop: he individual training jobs that are launched by the tuning job are dominated early in case the objective metric is not continuously improving.

This early stopping of the individual training jobs leads to earlier completion of the hyperparameter tuning job and reduce costs

- use small number od concurrent training jobs: On one hand, if you use a larger number of concurrent jobs, the tuning process will be completed faster. But in fact, the hyperparameter tuning process is able to find best possible results only by depending on the previously completed training jobs

ML model training challenges

Checkpointing

Problem: If these long-running training jobs stop for any reason such as a power failure, or oils fault, or any other unforeseen error, then you'll have to start the training job from the very beginning. This leads to lost productivity.

ML checkpoint allows you to:

- save state of MlL models during training
- checkpoints: shapshots of the model
  - model achitecture
  - model weights
  - training configs
  - optimizer
- fequency and number of checkpoints: If you have a high frequency of checkpointing and saving several different files each time, then you are quickly using up the storage. However, this high frequency and high number of checkpoints you're processing, this state will allow you to resume your training jobs without losing any training state information. On the other hand, if the frequency and the number of checkpoints you're saving each time is low, you are definitely saving on the storage space, but there is a possibility that some of the training state has been lost when the training job is stopped.

Amazon SM Manage Spot

allows you to save training costs: of Spot Instances that offer speed and unused capacity to users at discount prices. SageMaker Managed Spot uses these Spot

Instances for hyperparameter tuning and training and leverages machine learning checkpointing to resume training jobs easily

they can be closed in 2 minutes. Manage Spot automatically backs up the checkpoints in the S3 bucket.

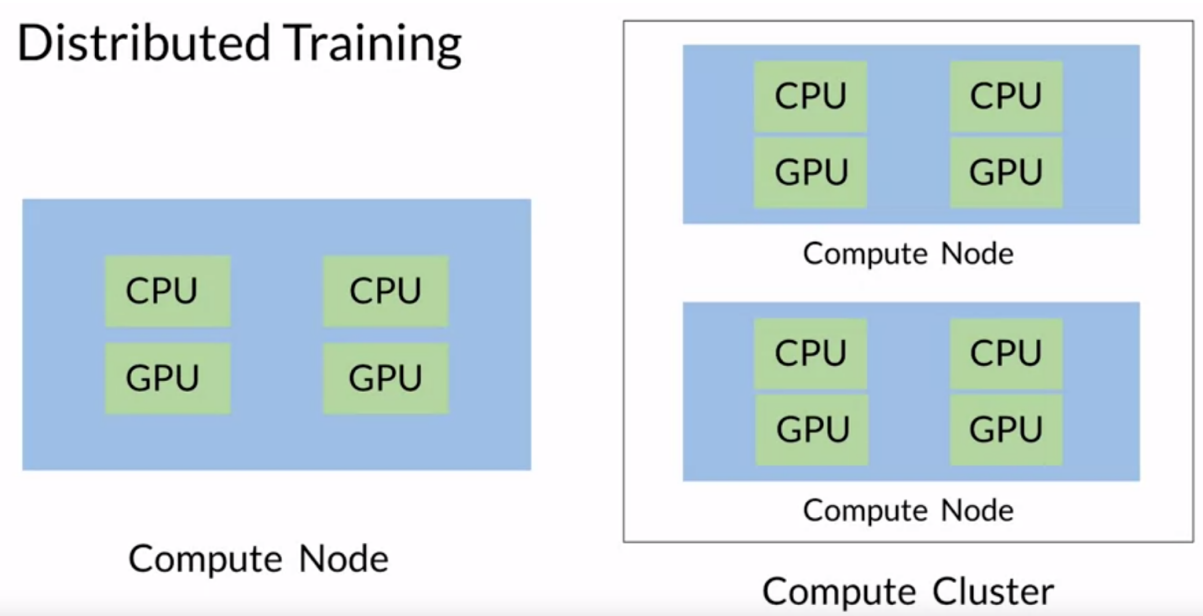
to take advantage of this product it’s needs that the script periodically saves the checkpoints and is also able to resume from the checkpoints.

Distributed Training strategies

Problem: increase training data volume & increased model size and complexity this will lead to long training jobs

In Distributed Training

The training load is split across multiple CPUs and GPUs, also called as devices within a single Compute Node. Or the node can be distributed across multiple compute nodes or compute instances that form a compute cluster

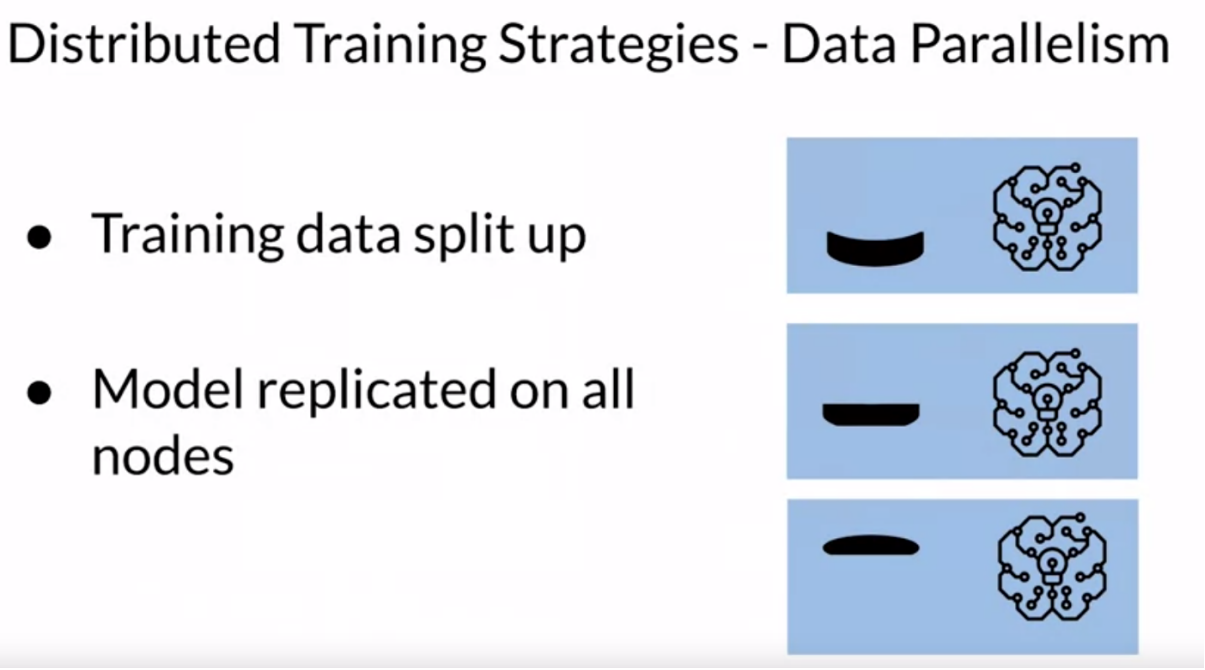


there are two distributed training strategies:

- data parellelism

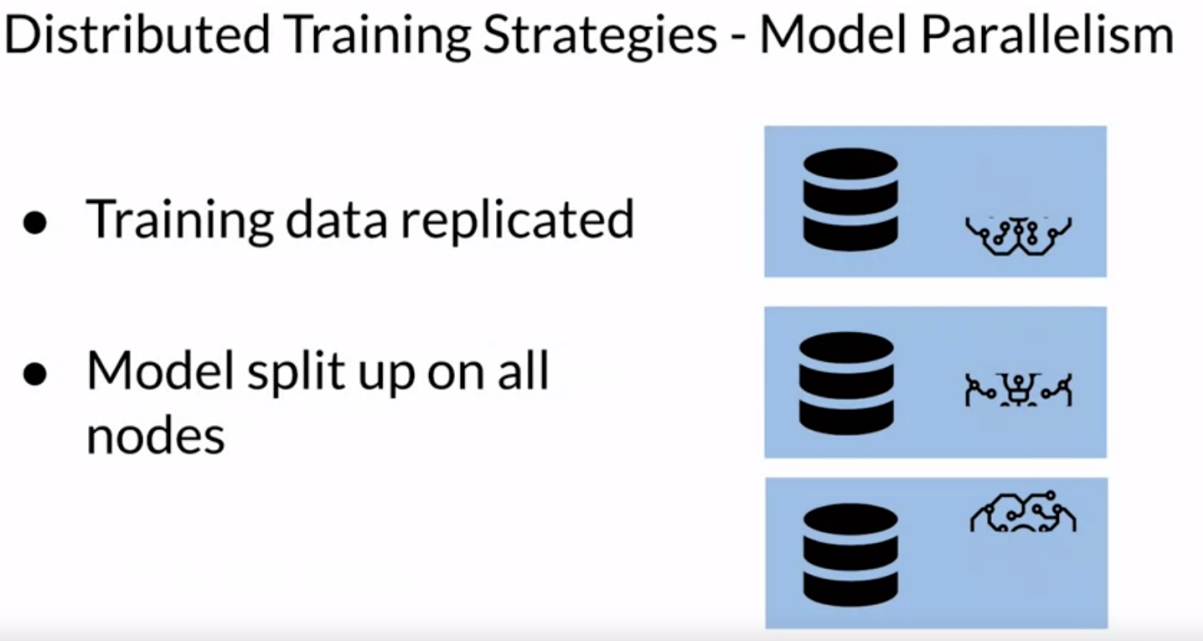
Data Parallelism

With data parallelism the training data is split up across the multiple nodes that are involved in the training cluster. The underlying algorithm or the neural network is replicated on each individual nodes of the cluster. Now, batches of data are retrained on all nodes using the algorithm and the final model as a result of a combination of results from each individual node.



Model Parallelism

In model parallelism, the underlying algorithm or the neural network in this case, is split across the multiple nodes. Batches of data are send to all of the nodes again so that each batch of the data can be processed by the entire neural network. The results are once again combined for a final model



Both are implemented via a parameter in an estimator:

```
from sagemaker.pytorch import PyTorch
estimator = PyTorch(
    entry_point='train.py',
    role=sagemaker.get_execution_role(),
    framework_version='1.6.0',
    py_version='py3',
    instance_count=3,
    instance_type='ml.p3.16xlarge',
    distribution={'smdistributed':{'modelparallel':{enabled': True}}}
)
estimator.fit()
```

Model Parallel Distribution Strategy

Choose a distribution Strategy:

If the train model can fit on a single node's memory, then use data parallelism. In the situations where the model cannot fit on a single node's memory, you have some experimentation to do to see if you can reduce the model size to fit on that single node. All of these experimentations will include an effort to resize the model. Some of the things that you can try to resize your model include tuning the hyperparameters. Tuning the hyperparameters, such as the number of neural network layers in your neural network, as well as tuning the optimizer to use will

have a considerable effect on the final model size. Another thing you can try is reduce the batch size. Try to incrementally decrease the batch size to see if the final end model can fit in a single node's memory. Additionally, you can also try to reduce the model input size. If for example, your model is taking a text input, then consider embedding the text with a low dimensional embedded in vector. Or if your model is taking image as an input, try to reduce the image resolution to reduce the model input. After trying these various experimentation, go back and check if the final model fits on a single node's memory. And if it does use data parallelism on a single node. Now, even after these experiments if the model is too big to fit on a single node memory, then choose to implement model parallelism.

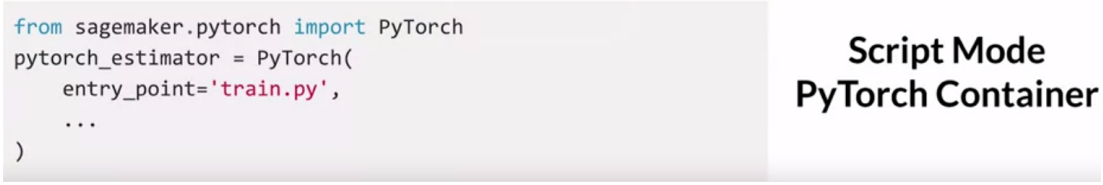
Custom Algorithms with Amazon SM

“bring your own container”

Options for estimators:

- ***built-in :*** you use the estimated object and to the estimated object, you're passing in the image URI. The image URI is pointing to a container that comes to the implementation of the built-in algorithm as well as the training and inference logic.

- ***bring your own script:*** you're using a SageMaker provider container such as a PyTorch container, but you are providing your own training script to be used during training with that particular container (training.py)



- ***bring your own container:*** you need to do 4 steps:
  - code

- containerize
- register the container with ECR
- create estimator

Codifying the logic

the code should include the logic for the algorithm that you want to implement, as well as the training logic and the inference logic

Containerize

creating a Docker container using the docker build command

```
algorithm_name=tf-custom-container-test
docker build -t ${algorithm_name} .
```

Register with ECR

First, you will create a repository to hold all of your algorithm logic as a container and into that repository, you push the container from the previous step using the docker push command. Once the push command is successful, you have successfully registered your container with Amazon ECR. This registered containers can be accessed within image URI that you can use to finally create an estimator

```
aws ecr create-repository --repository-name "${algorithm_name}" > /dev/null
fullname="${account}.dkr.ecr.${region}.amazonaws.com/${algorithm_name}:latest"
docker push ${fullname}
```

Create Estimator

nce you have that image URL, you simply create an estimator object by passing in that URI. After this point, using estimator is very similar to how you would use an estimator object with a built-in algorithm, for example

```
byoc_image_uri = '{}.dkr.ecr.{}.{}/{}'.format(account_id, region, uri_suffix,
ecr_repository + tag)
estimator = Estimator ( image_name=byoc_image_uri, ....  )
```

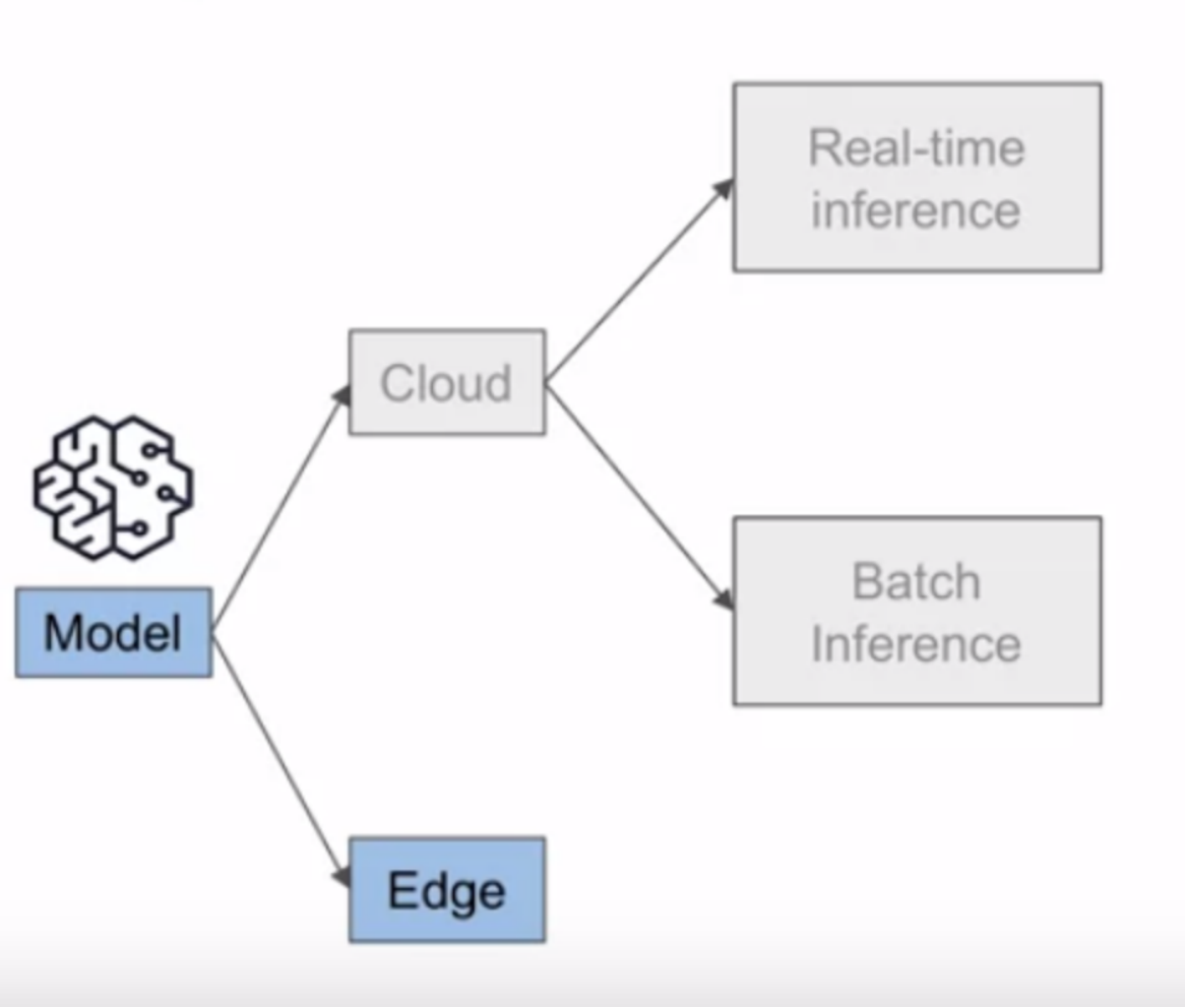
Reading Material

- Hyperband
- Bayesian Optimization
- Amazon SageMaker Automatic Model Tuning

W2 → Advanced model deployment and monitoring

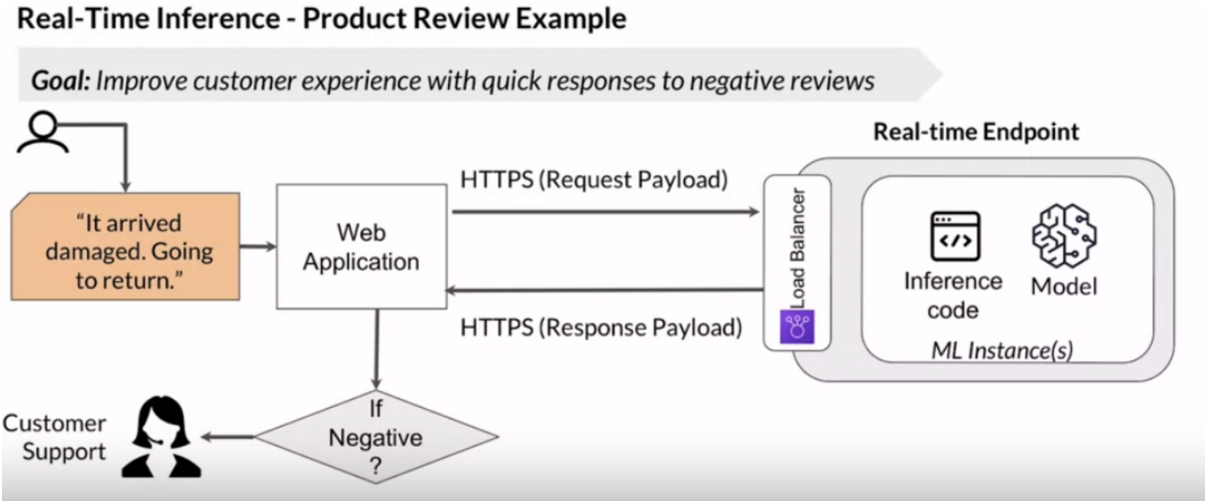
Model Deployment

Model Deployment Overview



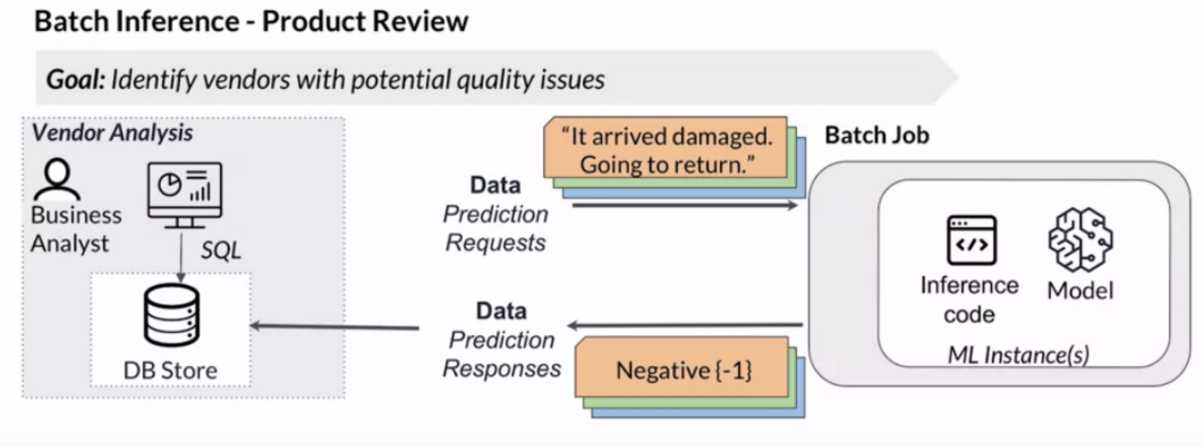
Real time Inference

his involves exposing an endpoint that has his serving stack that can accept and respond to requests. A serving stack needs to include a proxy that can accept incoming requests and direct them to an application that then uses your Inference code to interact with your model. This is a good option when you need to have low latency combined with the ability to serve new prediction requests that come in.



Batch inference

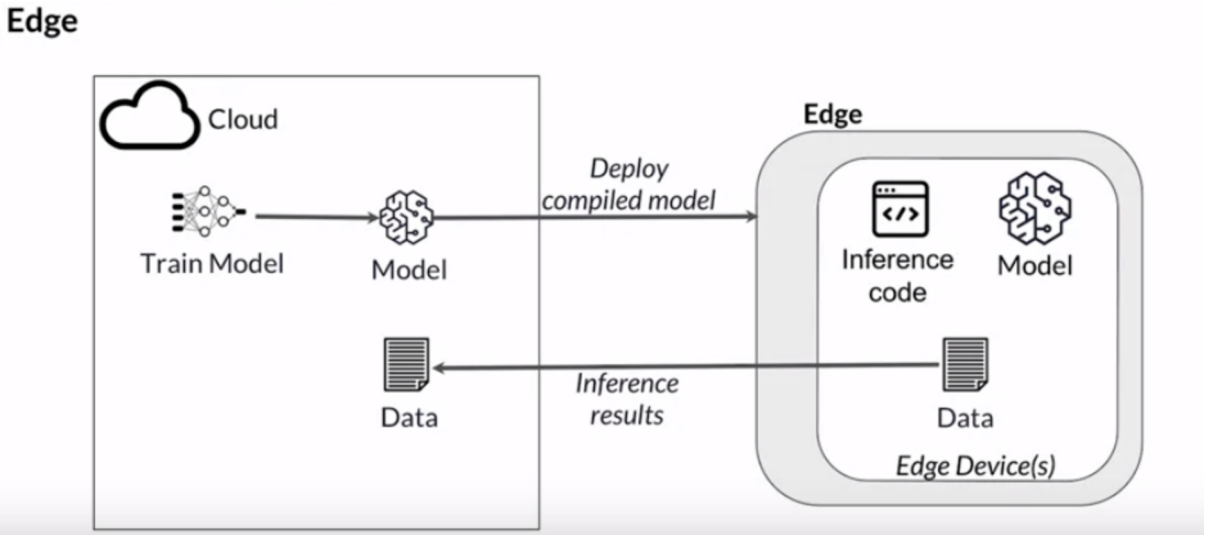
your batch in those requests for prediction, running a batch job against those batch requests and then out putting your prediction responses typically is batch records as well. Then once you have your prediction responses, they can then be used in a number of different ways. Those prediction responses are often used for reporting or are persisted into a secondary data store for use by other applications or for additional reporting.



Edge deployment

In the case of edge deployments, you train your models in another environment in this case in the cloud and then optimize your model for deployment to edge devices. This process is typically aimed at compiling or packaging your model in a way that is optimized to run at the edge. Which usually means things like reducing the model package size for running on smaller devices. In this case you could use something

like Sagemaker Neo to compile your model in a way that is optimized for running at the edge and use cases.



Comparisons

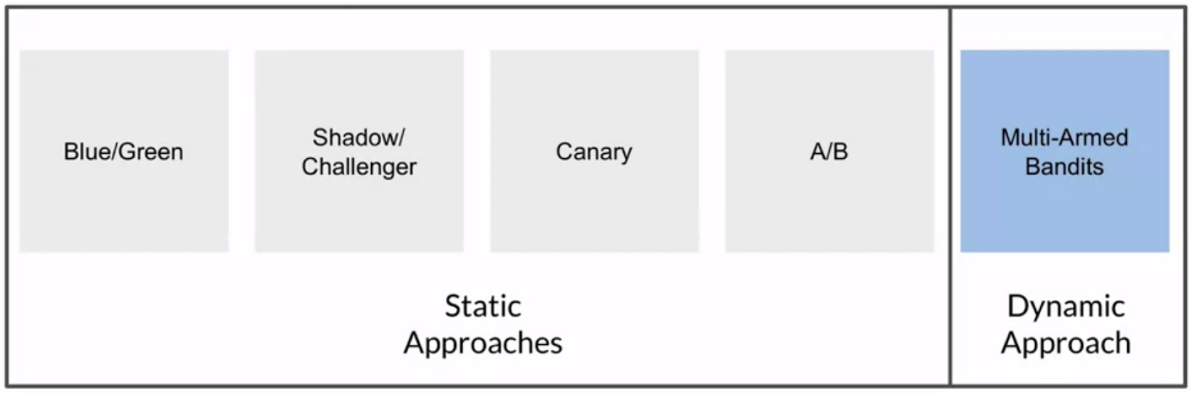
Choose the deployment option that best fits the use case

	Real-Time Inference	Batch Inference	Edge
When to use	Low latency real-time predictions (Ex. Interactive Recommenders)	Batch request & response prediction is acceptable for your use case (Ex. Forecasting)	Models need to deployed to edge devices (Ex. Limited connectivity, Internet of Things)
Cost	Persistent endpoint - pay for resources while endpoint is running	Transient environments - pay for resources for the duration of the batch job	Varies

The choice to deploy to the edge is typically an obvious one as there's edge devices and you might be working with use cases where there is limited network connectivity. You might also be working with internet of things or IOT use cases or use cases where the cost in terms of the time spent in data transfer is not an option even when it's single digit millisecond response. Now, the choice between real time inference and batch inference typically comes down to the ways that you need to request and consume predictions in combination with cost. A real time endpoint can serve real time predictions, where the prediction requests sent on input is unique and requires

an immediate response with low latency. The trade off is that a persistent endpoint typically cost more because you pay for the compute. And the storage resources that are required to host that model while that endpoint is up and running a batch job in contrast works well when you can batch your data for prediction. And that's your responses back, now, these responses can then be persisted into a secondary database that can serve real time applications when there is no need for new prediction requests. And responses per transaction, so in this case, you can run batch jobs in a transient environment. Meaning that the compute and storage environments are only active for the duration of your batch job. As a general rule, you should use the option that meets your use case and is the most cost effective

Deployment Strategies



static approaches: you decide when o swap traffic and how to distribute that traffic

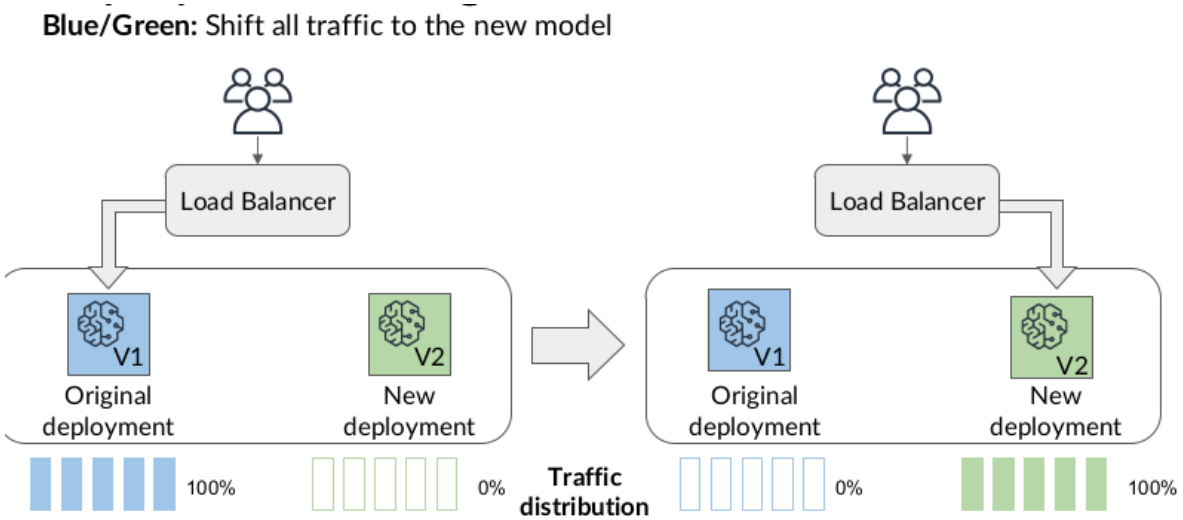
dynamic approach: incorporate machine learning to automatically decide when and how to distribute traffic between multiple versions of a deployed model

The goals of these strategies are minimizing risks of the new model behavior if it turns out worse, measure new model performance and minimize downtime in case you need to go back

Blue/ Green

- swap prediction request trafic
- easy rollback
- if new model is not good you can easly swap back

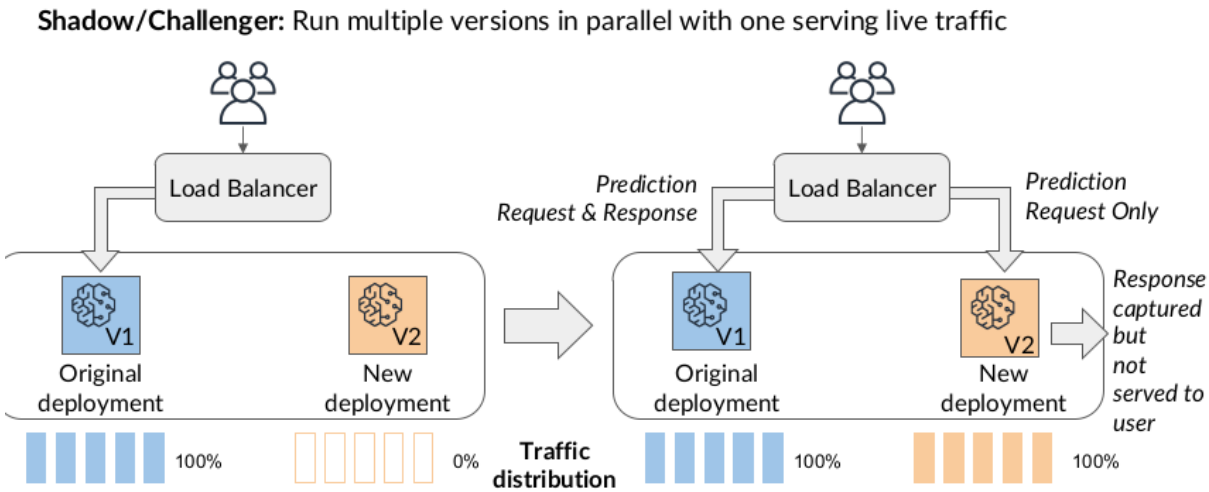
The downside to this strategy is that it is 100 percent swap of traffic. So if the new model version, version 2, in this case, is not performing well, then you run the risk of serving bad predictions to 100 percent of your traffic versus a smaller percentage of traffic



Shadow/Challenger

- Parallel prediction request traffic
- Validate new version without impact

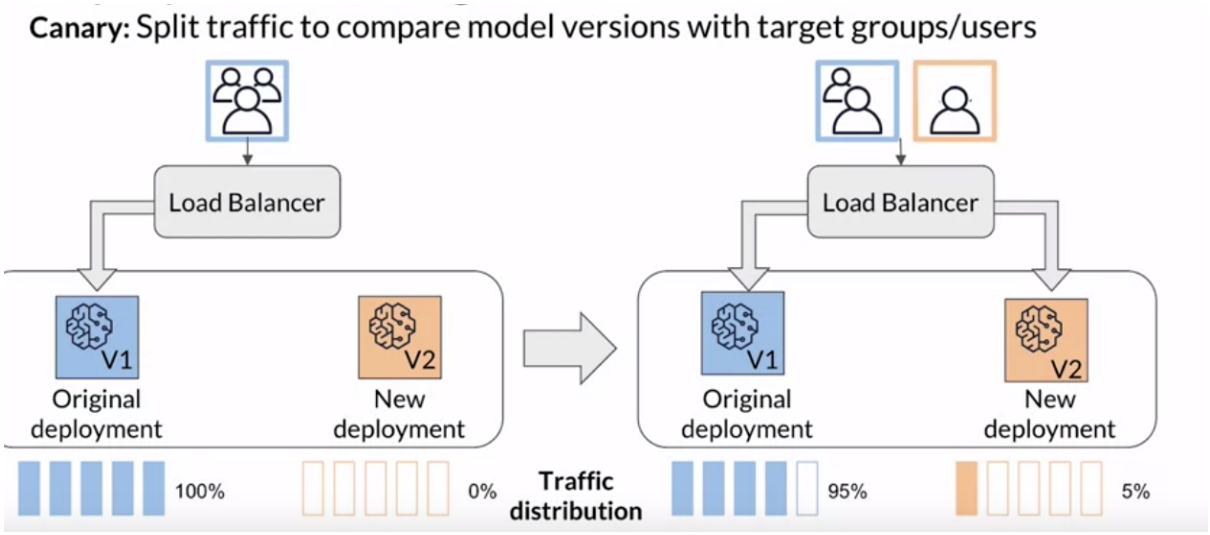
Really safe because you can compare both responses in real time without impacting the responses of the model to users



Canary Deployment

- splits traffic
- target smaller specif users/groups for 2nd model
- shorter validation cycles
- minimize risk of low performing model

you use a “control group” for testing the new model in real data/inferences before rolling out the model for all users.

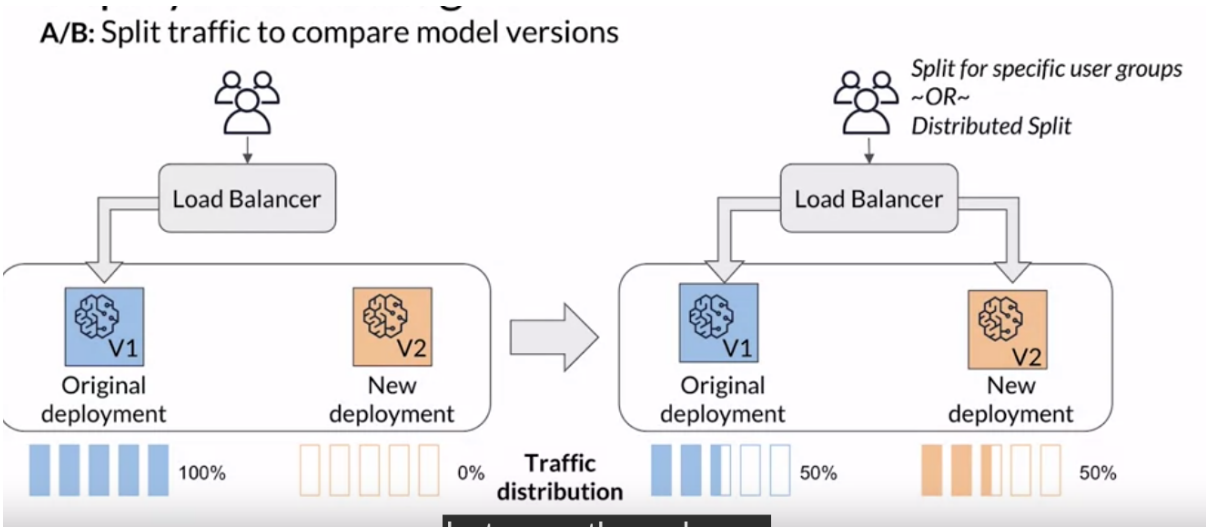


A/B testing

- split traffic
- target larger users groups or distribute % of trafic
- longer validation cycles
- minimize risk of low performance models

A/B tests are focused on gathering live data about different model versions. They typically, again, run for longer periods of time to be able to gather that performance data that is statistically significant enough, which provides that ability to confidently roll out Version 2 to a larger percent of traffic. Because you're running multiple

models for longer periods of time, A/B testing allows you to really validate your different model versions over multiple variations of user behavior

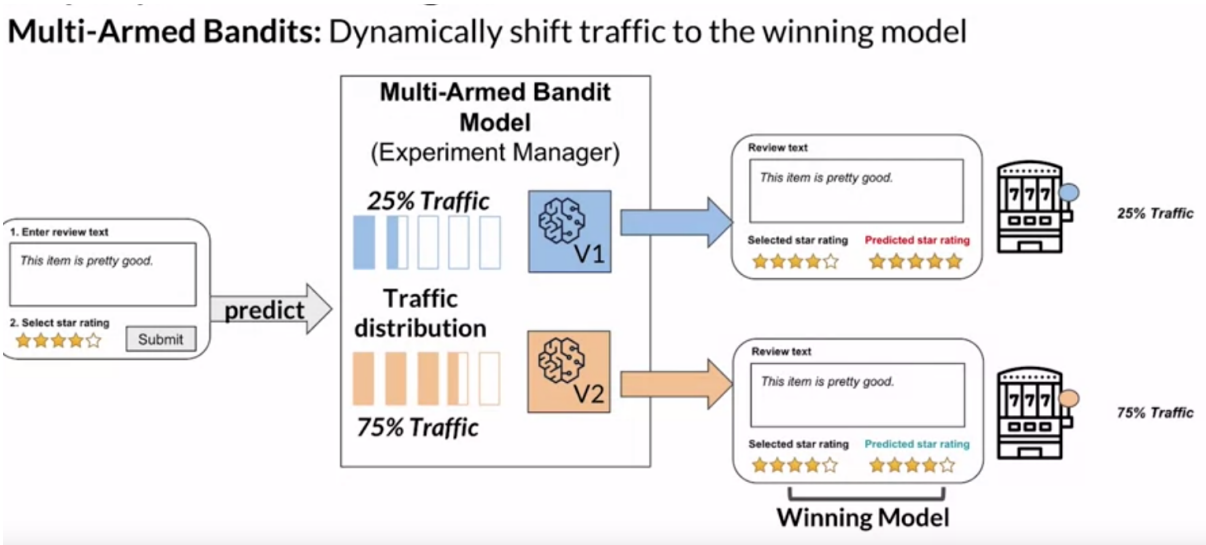


multi-armed bandits deployment

uses reinforcement learning as a way to dynamically shift traffic to the winning model versions by rewarding the winning model with more traffic but still exploring the nonwinning model versions in the case that those early winners were not the overall best models

it always send more traffic to the winning model.

Iteration 1: Model V2 wins, so it will receive more trafic next.



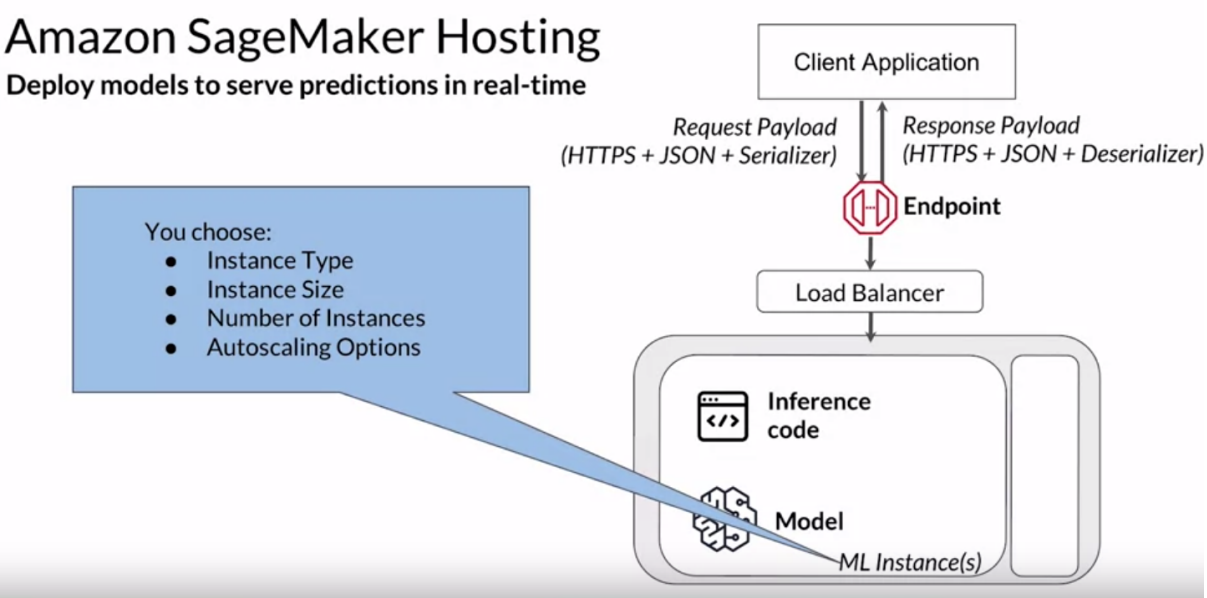
Amazon SageMaker Hosting: Real-Time Inference

Amazon SM hosting: Deploys models to serve predictions in real time, it is optimized for low latency of model predictions

Serving your predictions in real-time requires a model serving stack that not only has your trained model, but also a hosting stack to be able to serve those predictions. That hosting stack typically include some type of a proxy, a web server that can interact with your loaded serving code and your trained model. the model will be consumed by API real time request

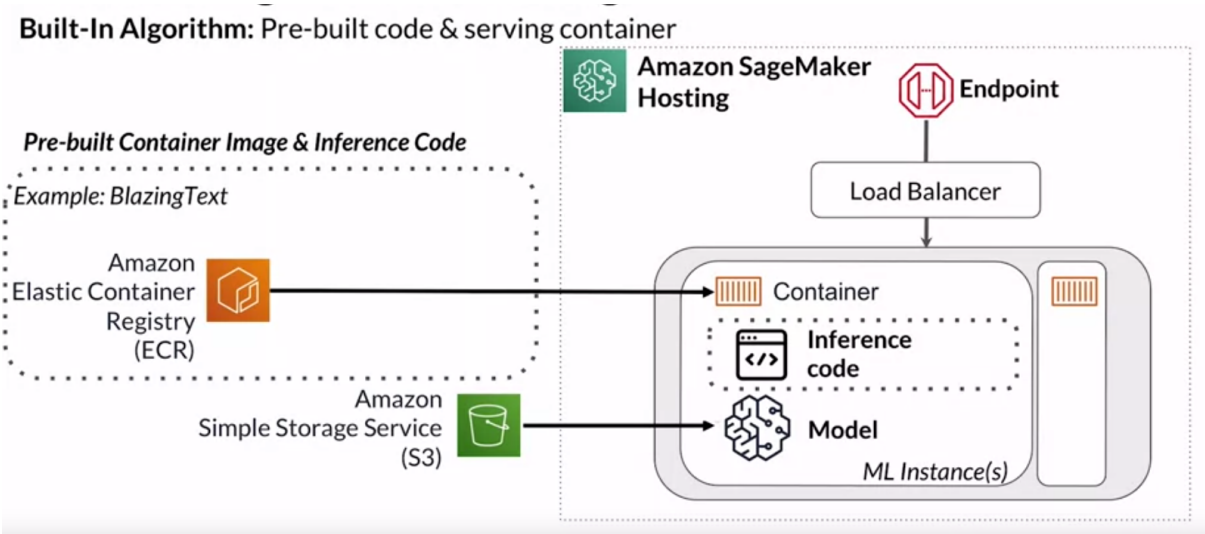
SageMaker has several built-in serializers and deserializers that you can use depending on your data formats.

With SageMaker model hosting, you choose the machine-learning instance type, as well as the count combined with the docker container image and optionally the inference code, and then SageMaker takes care of creating the endpoint, and deploying that model to the endpoint.



Built-in algorithms Deployment

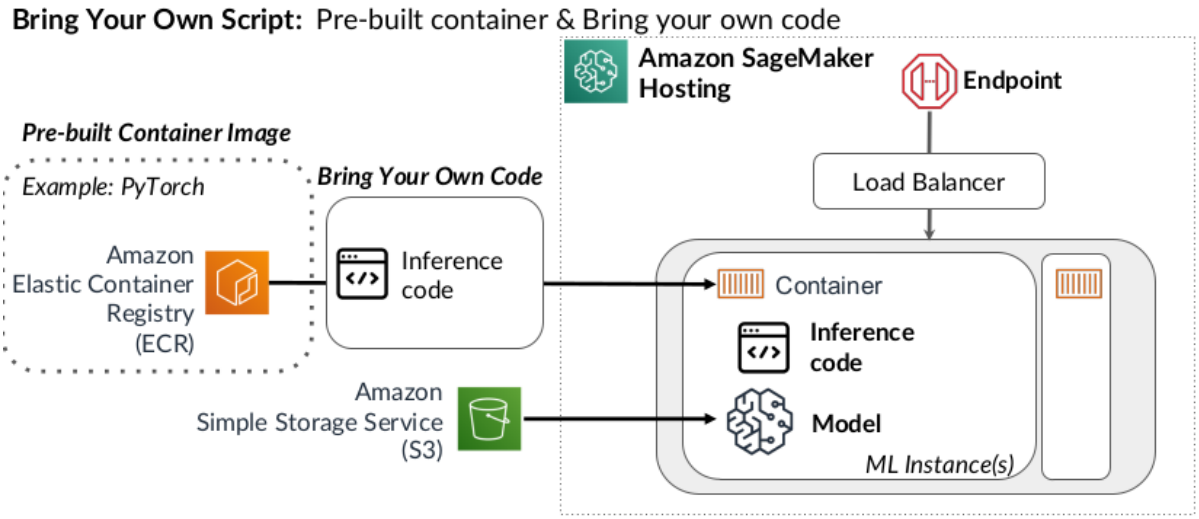
For this scenario to deploy your endpoint, you identify the prebuilt container image to use and then the location of your trained model artifact in S3.



Bring your own script

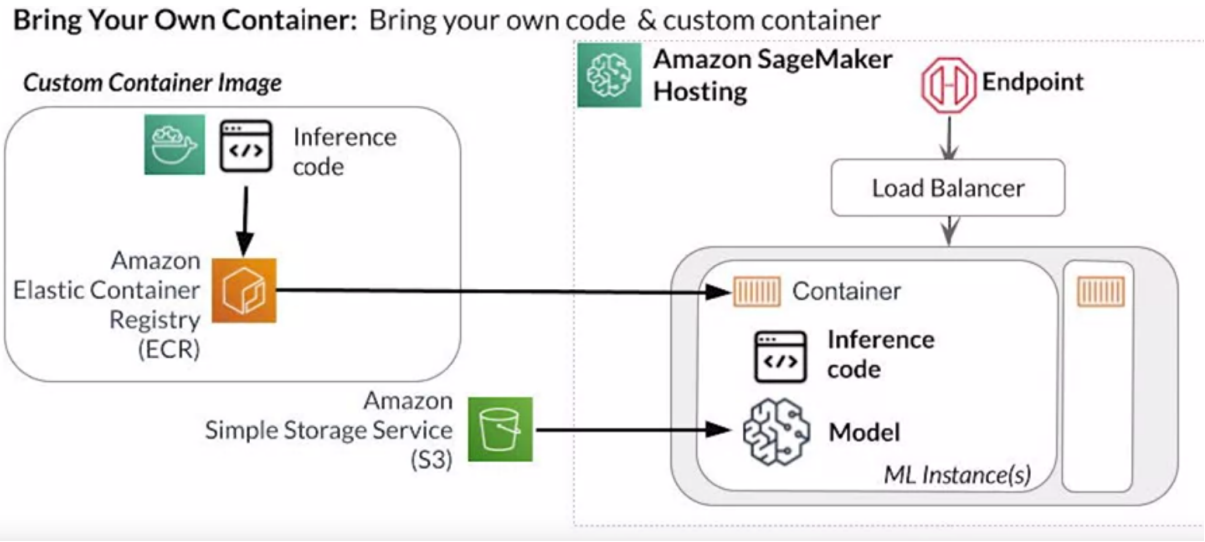
still uses a pre-built container, you just add code to the algorithm

using a built-in framework like TensorFlow or PyTorch, where you're still using pre-built container images for inference, but with the option of bringing your own serving code as well



Bring your own container

you create the docker image container that



AutoScalling Endpoints

- ensures you meet demands of workload
- cost optimization

iNot only can you scale your instances up to meet the higher workload demands when you need it, but you can also scale it back down to a lower level of compute when it is no longer needed. Second, using autoscaling allows you to maintain a minimum footprint during normal traffic workloads, versus overprovisioning and paying for compute that you don't need. The on-demand access to compute and storage resources that the Cloud provides allows for this ability to quickly scale up and down.

how it works:

SageMaker emits a number of metrics about that deployed endpoints such as utilization metrics and invocation metrics. Invocation metrics indicate the number of times an invoke endpoint request has been run against your endpoint, and it's the default scaling metric for SageMaker autoscaling. You can actually define a custom scaling metric as well, such as CPU utilization. Let's assume you've set up your autoscaling on your endpoint and you're using the default scaling metric of number of invocations. Each instance will emit that metric to CloudWatch. As part of the

scaling policy that you can figure. If the number of invocations exceeds the threshold that you've identified, then SageMaker will apply the scaling policy and scale up by the number of instances that you've configured. After scaling policy for your endpoint, the new instances will come online and your load balancer will be able to distribute traffic load to those new instances automatically. You can also add a cool down policy for scaling out your model, which is the value in seconds that you specify to wait for a previous scaled-out activity to take effect. The scale out cooldown period is intended to allow instances to scale out continuously, but not excessively. Finally, you can specify a cool down period for scaling in your model as well. This is the amount of time in seconds, again, after a scale-in activity completes, before another scale-in activity can start. This allows instances to scale in slowly.

Code on how to do this in the slides: c3_w3 53-55, description bellow:

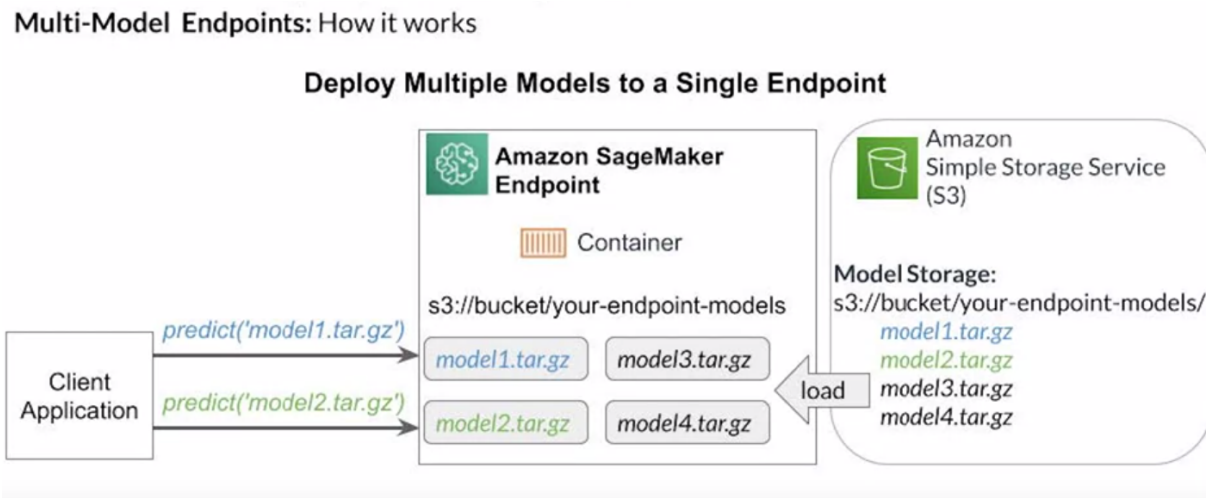
First, you register your scalable target. A scalable target is an AWS resource, and in this case, you want to scale the SageMaker resource as indicated in the service namespace. This is accepted as your input parameter. Because autoscaling is used by other AWS resources, you'll see a few parameters that specifically indicate that you want to scale a SageMaker endpoint resource. Similarly, the scalable dimension is a set value for SageMaker endpoint scaling. Some of the additional input parameters that you need to configure include the resource ID, which in this case is the endpoint variant that you want to scale. You'll also need to specify a few key parameters that control the minimum and maximum number of machine learning instances. The minimum capacity indicates the minimum value you plan to scale into. The maximum capacity is the maximum number of instances that you want to scale out to. In this case, you always want to have at least one instance running, and a maximum of two during peak periods. After you register your scalable target, you need to then define the scaling policy. The scaling policy provides additional information about the scaling behavior for your instances. In this case, you have your predefined metric, which is the number of invocations on your instance, and then your target value, which indicates the number of invocations per machine learning instance that you want to allow before invoking your scaling policy. You'll also see the scale-out and scale-in cooldown metrics that I mentioned previously. In this case, you see a scale-out cooldown of 60, which means that after autoscaling successfully scales out, it starts to calculate that cool-down time. The Scaling policy will increase again to that desired capacity until the cool down period ends. The ScaleInCool

down setting of 300 seconds means a SageMaker will not attempt to start another cool down policy within 300 seconds when the last one completed. In your final step to set up autoscaling, you will apply autoscaling policy, which means you apply that policy to your endpoint. Your endpoint will now be skilled in and scaled out according to that scaling policy that you've defined. You'll notice here you refer to the previous configuration that was discussed, and you'll also see a new parameter called policy type. Target tracking scaling refers to the specific autoscaling type that is supported by SageMaker. This uses a scaling metric and a target value as an indicator to scale. You'll have the opportunity to get hands on your lab for this week in setting up and applying autoscaling to SageMaker endpoints.

Multi-model endpoint

Instead of downloading your model from S3 to them machine learning instance immediately when you create the endpoint, with multi-model endpoints, SageMaker dynamically loads your models when you invoke them. You invoke them through your client applications by explicitly identifying the model that you're invoking.

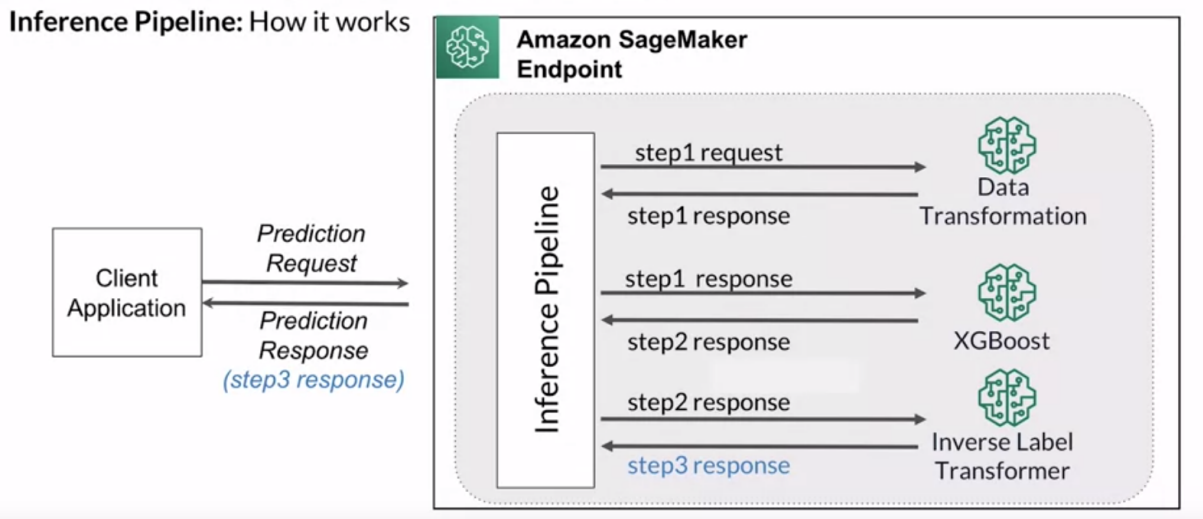
ALL models must share the same container image.



Inference Pipeline

Inference pipeline allows you to host multiple models behind a single endpoint. But in this case, the models are sequential chain of models with the steps that are required for inference. This allows you to take your data transformation model, your

predictor model, and your post-processing transformer, and host them so they can be sequentially run behind a single endpoint. As you can see in this picture, the inference request comes into the endpoint, then the first model is invoked, and that model is your data transformation. The output of that model is then passed to the next step, which is actually your XGBoost model here, or your predictor model. That output is then passed to the next step, where ultimately in that final step in the pipeline, it provides the final response or the post-process response to that inference request.



This allows you to couple your pre and post-processing code behind the same endpoint and helps ensure that your training and your inference code stay synchronized

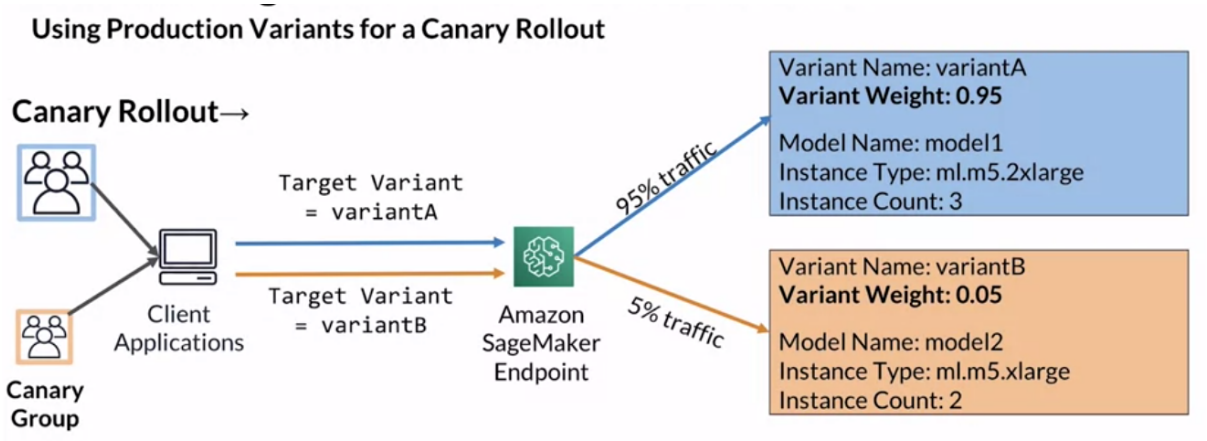
Production Variants

A **production variant** is a package SageMaker model combined with the configuration that defines how that model will be hosted.

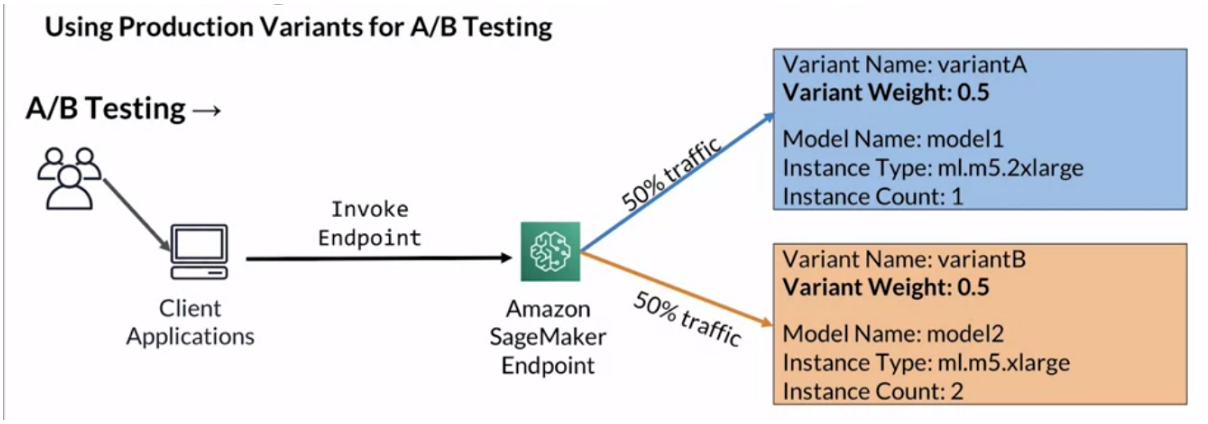
Used to perform more advanced deployment strategies, sharing the traffic between models.

SageMaker model includes information such as the S3 location of that trained model artifact, the container image that can be used for inference with that model, and the service run-time role and the model's name

The hosting resources configuration includes information about how you want that model to be hosted: no and type of ML instances, variant name and weight:



The client application controls the traffic. You can also do this specification in SM, the client application just invokes the endpoint and then the SM specifies the traffic.



Production Variants CODE on slides: 65-69, description bellow:

In this case, you'll learn how to use production variants for the SageMaker option where you're using a pre-built container image. In this first step, you construct the URI for the pre-built Docker container image. For this, you're using a SageMaker provided function to generate the URI of the Amazon Elastic Container Registry image that'll be used for hosting. The next step includes creating two model objects which packages each of your trained models for deployment to a SageMaker endpoint. To create the model packages, you'll use the URI information from the previous step and supply a few other items for packaging, such as the location of your trained model artifact is stored in Amazon S3, the AWS identity and access

management or the IAM role that will be used by the inference code to access AWS resources. Next, you configure the production variants that will be used when you create your endpoint. Each variant points to one of the previously configured model packages, and it also includes the hosting resources configuration. You can see in this case, you're indicating that you want 50 percent of your traffic sent to model variant A and 50 percent of your traffic sent to model variant B. Recall that the model package, combined with the hosting resources configuration, make up a single production variant. Now that you've configured your production variants, you now need to configure the endpoint to use these two variants. In this step, you create the endpoint configuration by specifying the name and pointing to the two production variants that you just configured. The endpoint configuration tells SageMaker how you want to host those models. Finally, you create the endpoint, which uses your endpoint configuration to create a new endpoint with two models or two production variants

Amazon SageMaker Batch Transform: Batch Inference

You need to package your **model** first. Your model package contains information about the S3 location of your trained model artifact, and the container image to use for inference.

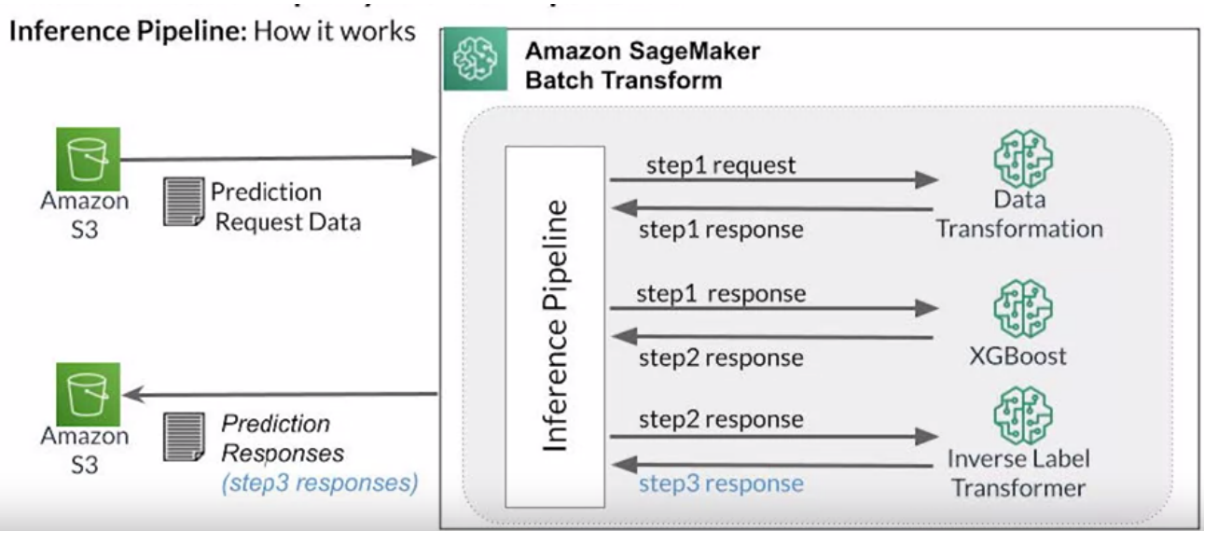
Next, you create your **transformer**. For this, you provide the configuration information about how you want your batch job to run. This also includes parameters such as the size and the type of machine learning instances that you want to run your batch job with, as well as the name of the model package that you previously created. Additionally, you specify the output location, which is the S3 bucket, where you want to store your prediction responses.

then you can run you batch transform job.

when the job is complete, the predictions will be predicted to the S3 bucket specified.

Batch jobs operate in a transient environment, which means that the Compute is only needed for the time it takes to complete the batch Transform job

Also includes **Inference Pipeline** (chain multiple models, the batch job will then include: data preprocessing and also inference and post inference processing)



Model Integration

To apply the model to new data you also need to apply the data transformations that you applied to the training data. To do this you can:

- **implement data transformations in client application**
  - difficult to manage and scale if multiple apps interact with the model
  - hard to ensure that both codes (training and inference) stays in sync
- **implement transformation code before calling hosted model**
  - still need to ensure that transformation code and training code stay in sync
- **implement data transformations in Inference Pipeline**
  - code is all in sync

Monitoring ML Workloads

Model Monitoring

Models need to be monitored to ensure its predictions are still accurate and retrain the model if needed.

Concept Drift

happens when the environment you trained your model in no longer reflects the current environment

detecting concept drift:

A method for detecting concept drift includes continuing to collect ground truth data that reflects your current environment. And running this labeled ground truth data against your deployed model to evaluate your model performance against your current environment. Here, you're looking to see if the performance metric that you optimize for during training like accuracy still performs within an acceptable range for your current environment.

Data drift

changes in the model input data or changes to the feature data. for signals that the serving data has shifted from the original expected data distribution that was actually used for training. This is often referred to as training serving skew

detecting data drift:

One is an open source library called **Deequ**, which performs a few steps to detect signs of data drift. First, you do data profiling to gather statistics about each feature that was used to train the model. So collecting data like the number of distinct values for categorical data or statistics like min and max for numeric features. Using those statistics that are gathered during that data profiling, constraints get established to then identify the boundaries for normal or expected ranges of values for your feature data. Finally, by using the profile data in combination with the identified constraints, you can then detect anomalies to determine when your data goes out of range from the constraints

System Monitoring

system monitoring is also key to ensuring your models in the surrounding resources that are supporting your machine learning workloads are monitored for signals of disruption as well as potential performance decline

why: ensure your model and supporting resources are functioning as expected

what to monitor:

- model latency (time)
- system metrics (cpu utilization)
- machine learning pipelines (if there are any potential issues with model retraining or deploying a new model version)

Business Monitoring

monitor Model business impact

why: ensuring your deployed model is actually accomplishing what you intend for it to do, which ties back to the impact to your business objectives. This can be difficult to monitor or measure.

SageMaker Model Monitor

includes 4 different monitor types:

- data quality: monitor data drift
- model quality: monitor drift in model quality metrics
- statistical bias drift: monitor statistical bias drift in model predictions
- feature attribution drift: monitor drift in feature attribution

Data Quality

data quality, monitor your monitoring for signals that the feature data that was used to train your models has now drifted or is statistically different from the current data that's coming in for Inference model monitor uses DQ. Which is an open source library built on Apache spark that performs data profiling generates constraints based on that data profile. And then detects for anomalies when data goes out of bounds from the expected values or constraints

code on slides 101-105

you can define how much of data should be use, when to run this job and so on. This info is also on amazon cloud watch so that you can have alerts for drift.

model quality

With the model quality monitor, you're actually using new ground truth data that is collected to evaluate against your deployed model for signs of concept drift. So here you use the new label data to evaluate your model against the performance metric that you've optimized for during training, which could be something like accuracy. You then compare the new accuracy value to the one you identify during model training to save your accuracy is potentially going down.

you need to ensure that you have a process to collect new data.

statistical bias

The statistical bias drift monitor, monitors for predictions for signals of statistical bias. And it does this by integrating with Sage maker, clarify again, the process to set up this monitor is similar to the others. In this case you create a baseline that is specific to bias drift and then schedule your monitoring jobs just like the other monitoring types

feature attribution

Finally, feature attribution, monitors for drift in your features for this model, monitor, monitors drift by comparing how the ranking of individual features changed from the training data to the live data. This helps explain model predictions over time. Again, the steps for this model monitor type are similar to the others. However, the baseline job in this case uses SHAP behind the scenes SHAP or shapely additive explanations is a common technique used to explain the output of a machine learning model

Reading Material

- A/B Testing
- Autoscaling

- Multi-armed bandit
- Batch Transform
- Inference Pipeline
- Model Monitor

W3 → Data Labeling and human-in-the-loop pipelines

Data Labeling

data labeling: is the process of identifying raw data such as images, text files, and videos, among others, and adding one or more meaningful and informative labels to the data

> **ex:** For example, labels might indicate whether a photo contains a dog or a cat which words were uttered in an audio recording, or if an X ray image shows a tumor.

In Supervised Learning models learn from this labels, so taking time to accurately define these labels is crucial to have good models.

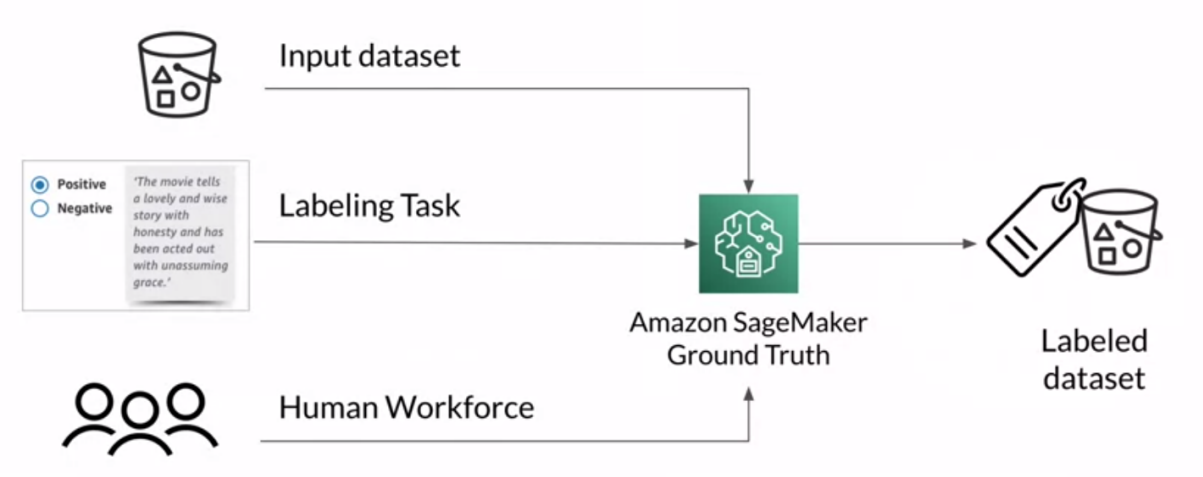
Data Labeling challenges:

- massive scale , lots of data
- need for high accuracy for model performance
- time consuming

Amazon Ground Truth

Ground Truth provides a managed experience where you can set up an entire data labeling job with only a few steps. Ground Truth helps you efficiently perform highly accurate data labeling using data stored in Amazon Simple Storage Service or

Amazon S3, using a combination of automated data labeling and human performed labeling.



how it works:

First, as part of the data labeling job creation, you provide a pointer to the S3 bucket that contains the input dataset to be labeled, then, you define the labeling task and provide relevant labeling instructions. Ground Truth offers templates for common labeling tasks, where you need to click only a few choices and provide minimal instructions on how to get your data labeled. Alternatively, you can create your own custom template. As the last step of creating a labeling job, you select a human workforce. You can choose between a public crowdsourced workforce, a curated set off third-party data labeling service providers, or your own workers through a private workforce team. Once the labeling job is completed, you can find the labeled data set in Amazon S3.

Input data

you only need to provide the s3 bucket where the data is. in the automated version Ground Truth will detect the labeling data by itself. otherwise you need to pass the file with the specifications.

Select Labeling Task

the task you can select depend on your data type:

- image data:
  - classification

    - multi-label classification
    - bounding box
    - semantic segmentation
    - label verification (verifying the current labels)
- video data:
    - video clip classification
    - video object detection/tracking
        - bounding box
        - polygon
        - polyline
        - key point
- text data:
    - classification
    - multi-label classification
    - name entity recognition
- custom tasks: via costum AWS Lambda Functions

Select Human Workforce

variable options:

- amazon mechanical turk
- private (your own employees)
- companies with specialized knowledge and with confidential agreements

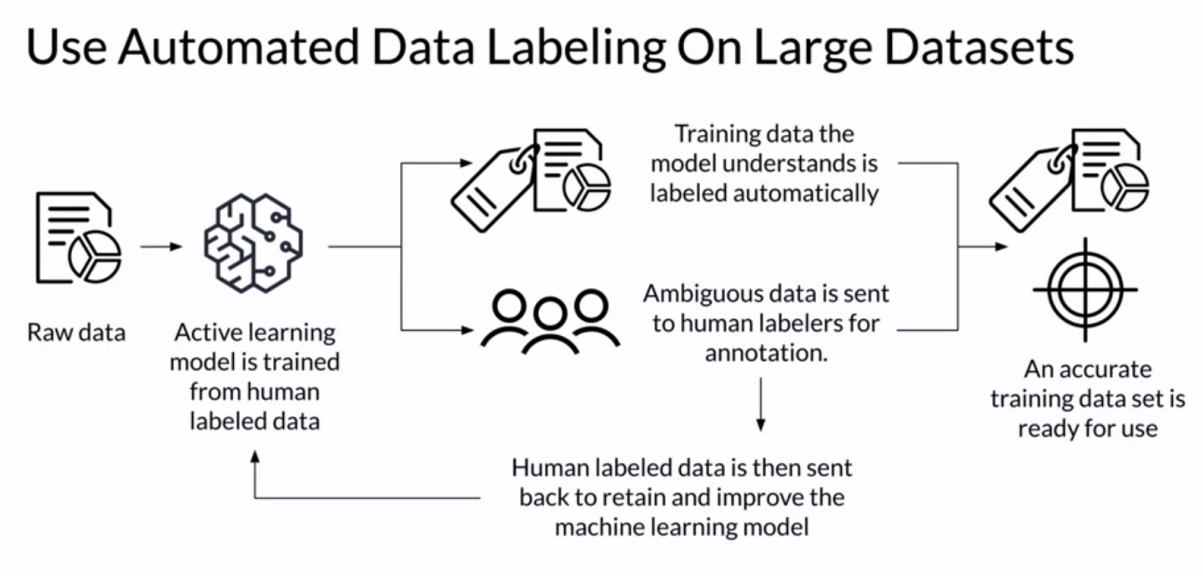
Best Practices in data labeling

- provide clear instructions
    - provide examples
    - only show relevant labels
    - use multiple worker per task

- verify and adjust labels
- re-use prior laeling jobs to create hierarchical labels:
  - ex: 1st step → cat/dog, 2nd step → baby dog/adult dog

Automated Data Labeling in AWS Ground Truth

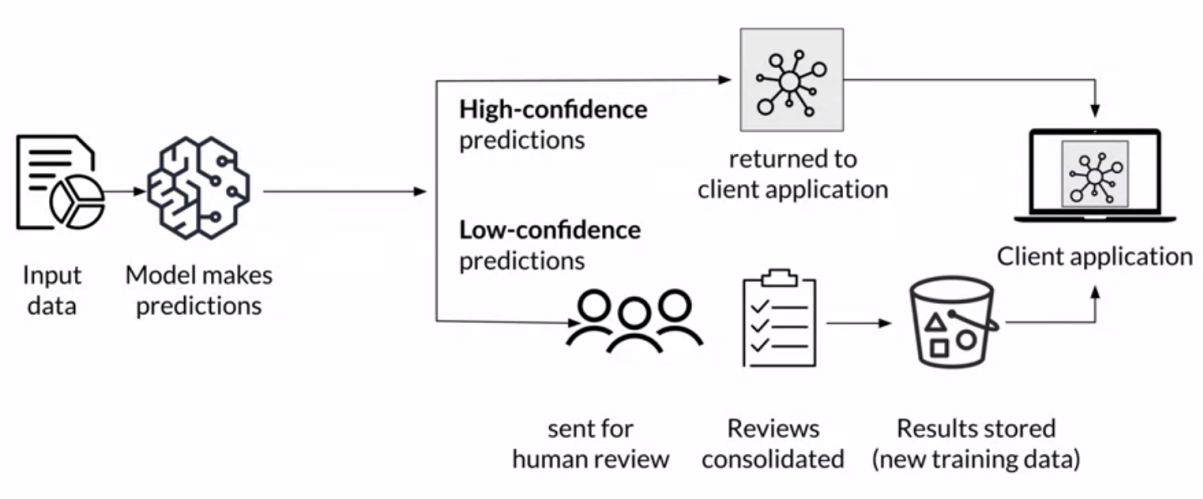
they use SM built-in models to do the labeling.



Human-in-the-loop Pipelines

concept: Integrating machine learning systems into your workflow while keeping a human eye on the results to provide a required level or position

It’s used mainly for reviewing the labels predicted by the model that have low confidence stores! This should be decided with a confidence treshhold. On the other side you can also review a random number of predictions.



Amazon Augmented AI or Amazon A2I allows you to orchestrated these pipelines.

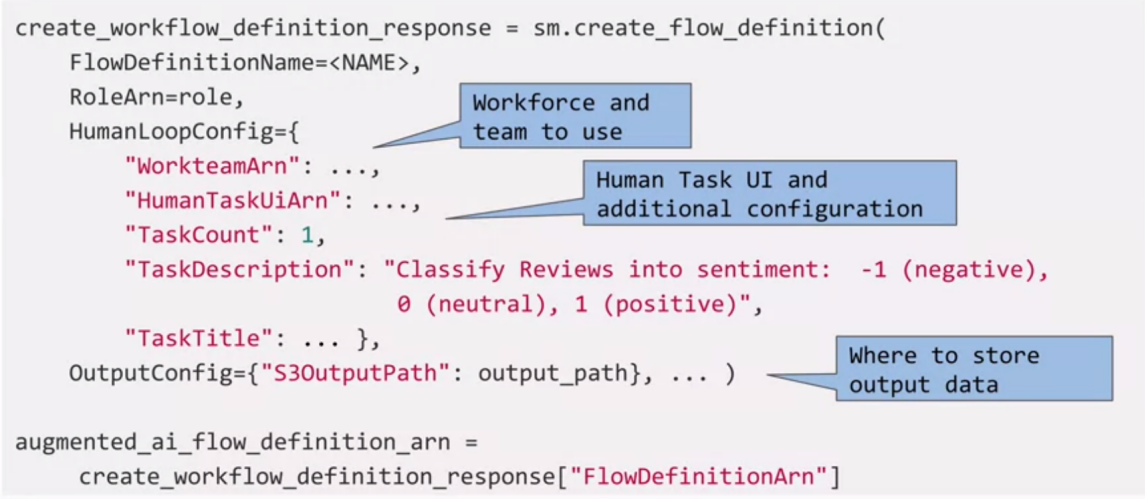
Amazon A2I provides built-in human review workflows for common machine learning use cases such as text extraction from documents, which allows predictions from, for example, Amazon Textract to be reviewed easily. and you can also create your own review flows.

How it works:

the individual steps for building a human-in-the-loop review system

- define human work force (the same as Ground Truth)
- define task (same options as Ground Truth for each data type)
- define human review workflow (
  - start human loop with AWS AI Service API calls
  - start human loop with custom ML models

  Amazon A2I provides built-in human review workflows for common ML use cases such as content moderation and text extraction from documents. For this purpose, Amazon A2I is integrated with AWS AI Services, including Amazon Textract. You can also create your custom workflows to integrate with additional AI services or ML models built on SageMaker or any other tools.



Reading Material

- Towards Automated Data Quality Management for Machine Learning
- Amazon SageMaker Ground Truth Developer Guide
- Create high-quality instructions for Amazon SageMaker Ground Truth labeling jobs
- Amazon SageMaker Augmented AI (Amazon A2I) Developer Guide
- Amazon Augmented AI Sample Task UIs
- Liquid open source Template Language