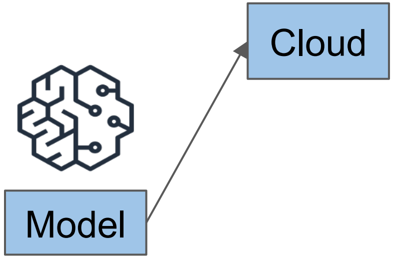
Model Deployment

Overview

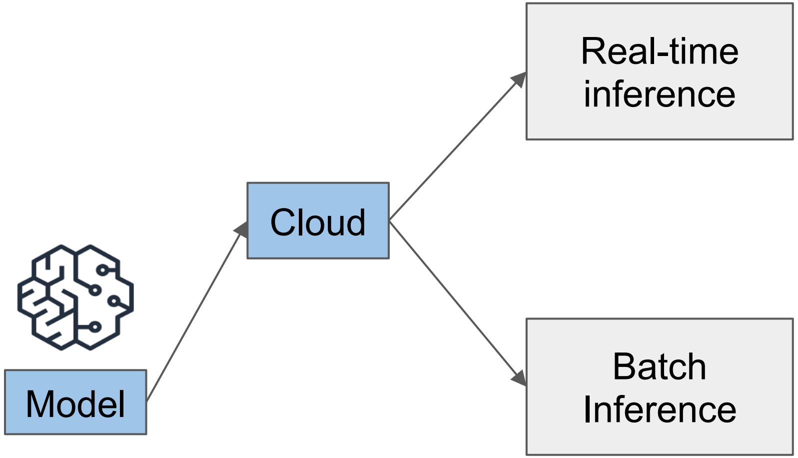
Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

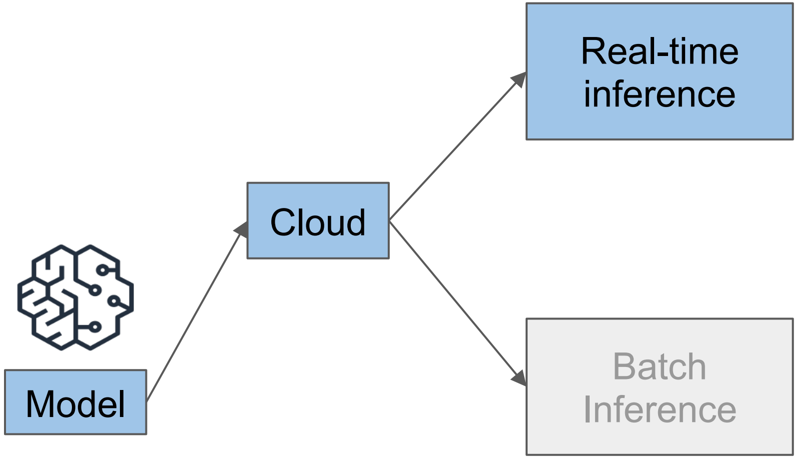
Deployment Options



Deployment Options



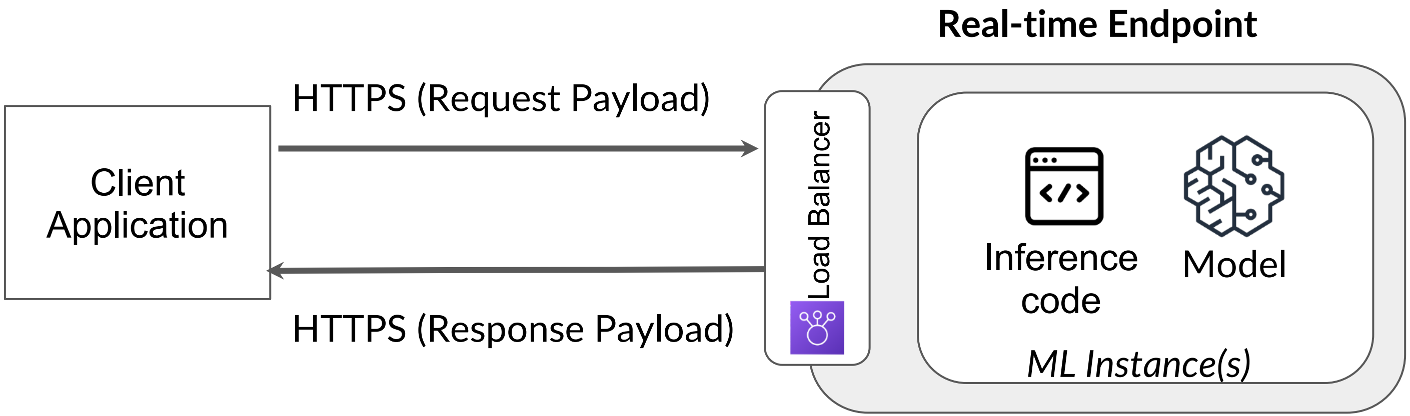
Deployment Options



Deployment Options

Real-Time Inference

But let's start with deploying a model for real time inference in the cloud deploying a model for real time Inference. This means deploying it to a persistent hosted environment that's able to serve requests for prediction and provide prediction responses back in real time or near real time. This 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, so some example use cases here would be fraud detection or product recommendation.

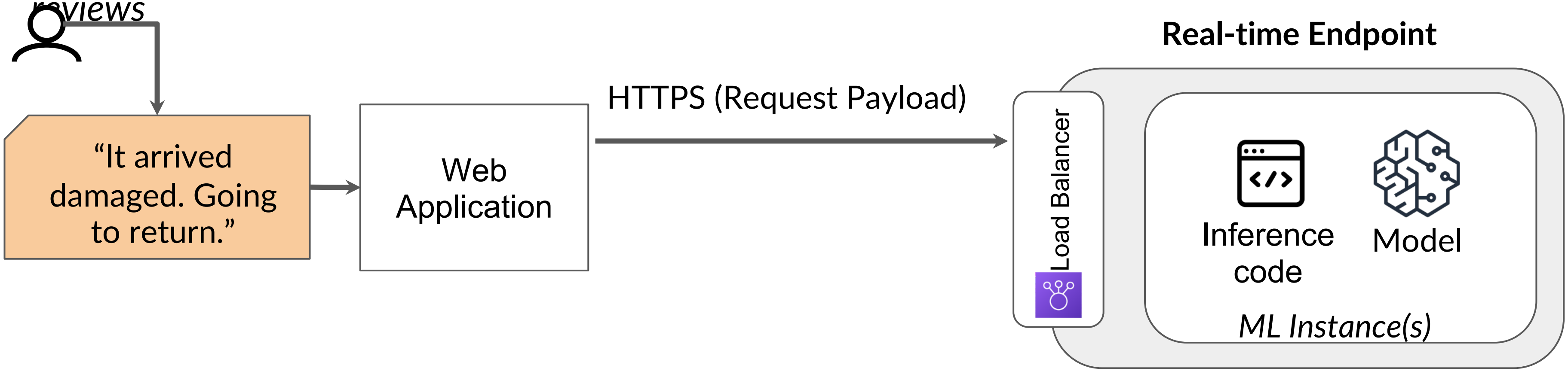


EXPOSE YOUR ENDPOINT THAT HAS A SERVING STACK

Deployment Options

Real-Time Inference - Product Review Example

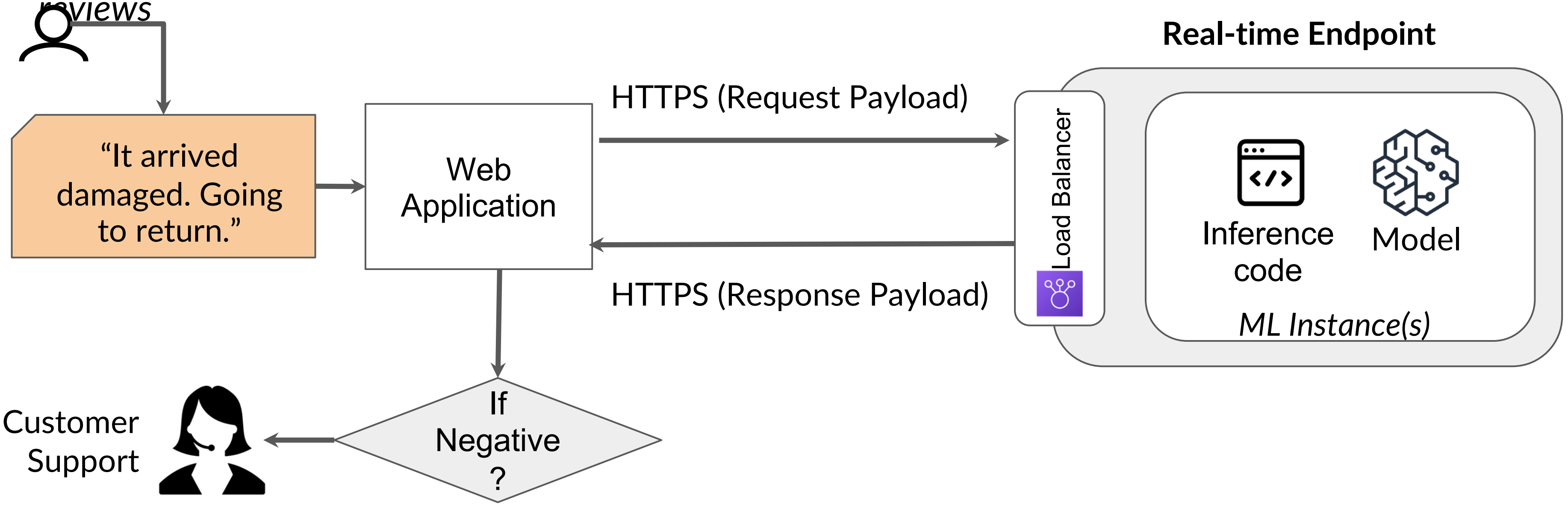
Goal: *Improve customer experience with quick responses to negative reviews*



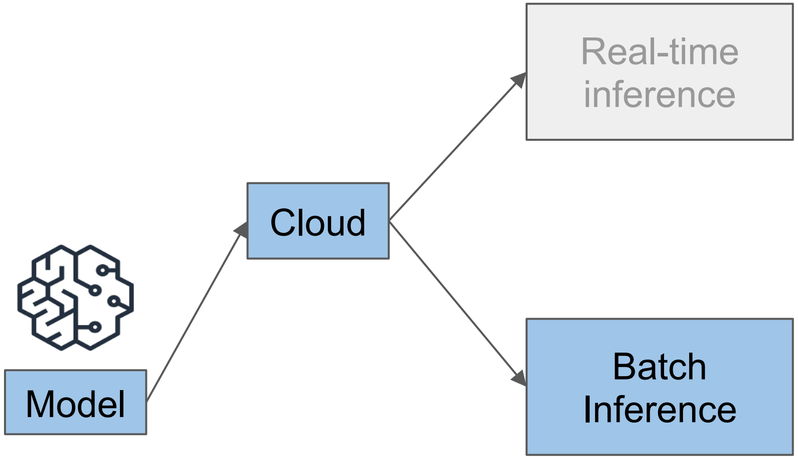
Deployment Options

Real-Time Inference - Product Review Example

Goal: *Improve customer experience with quick responses to negative reviews*



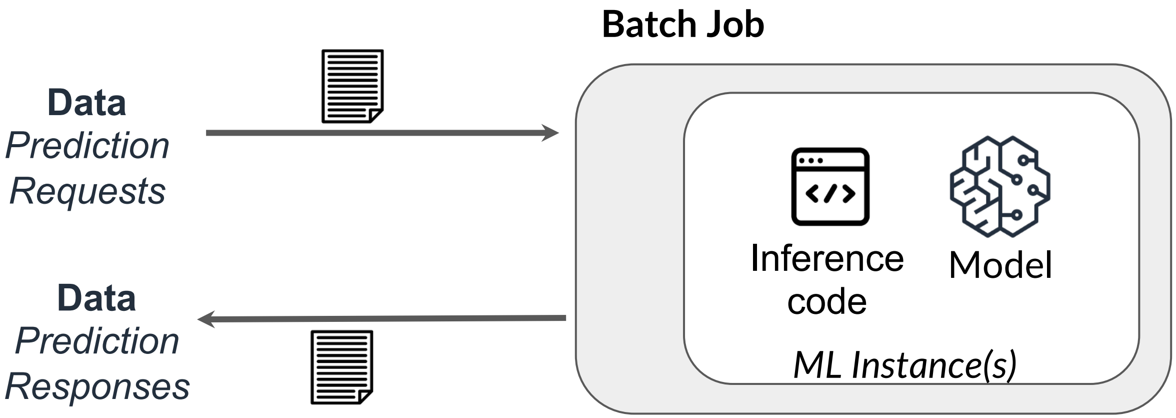
Deployment Options



Deployment Options

Batch Inference

You aren't hosting a model that persists and can serve requests for prediction as they come in. Instead, 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.



Deployment Options

Batch Inference

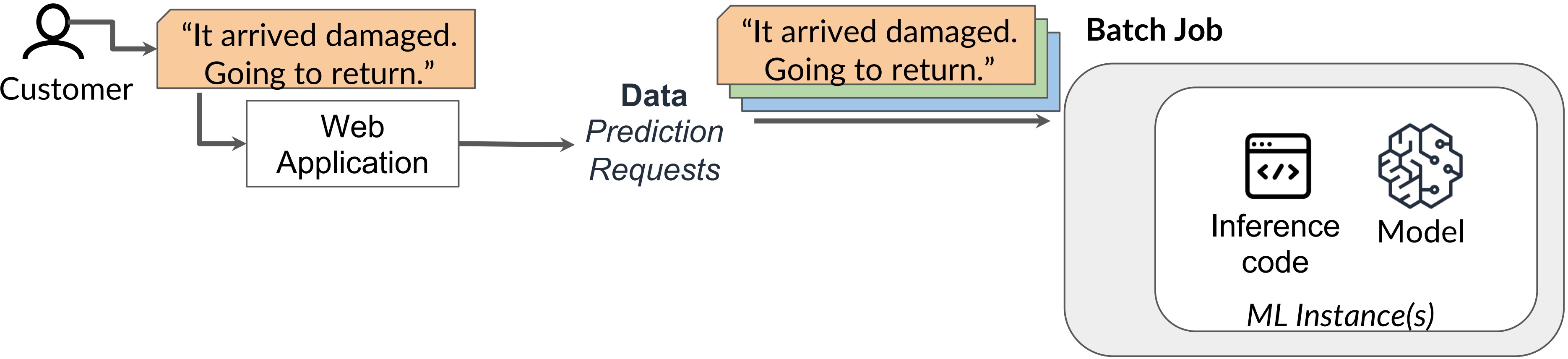
Deployment Options

Batch Inference

Deployment Options

Batch Inference - Product Review

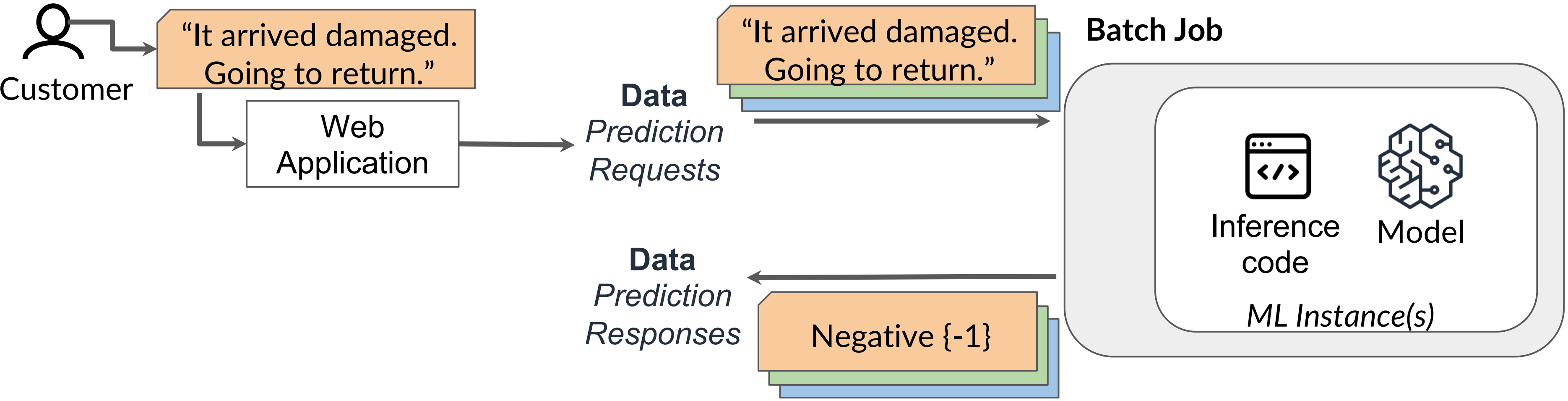
Goal: *Identify vendors with potential quality issues*



Deployment Options

Batch Inference - Product Review

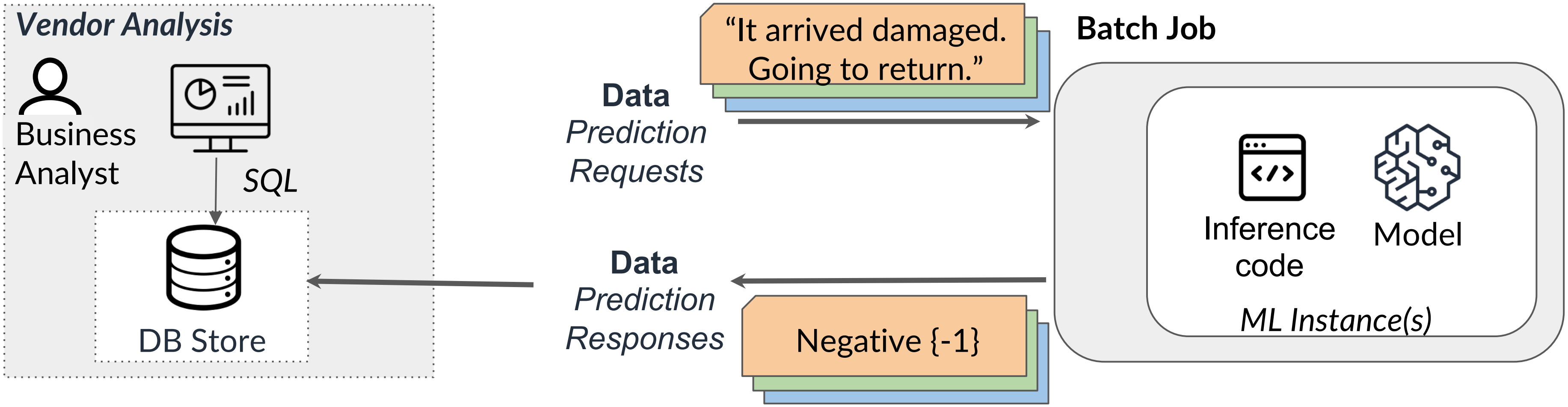
Goal: *Identify vendors with potential quality issues*



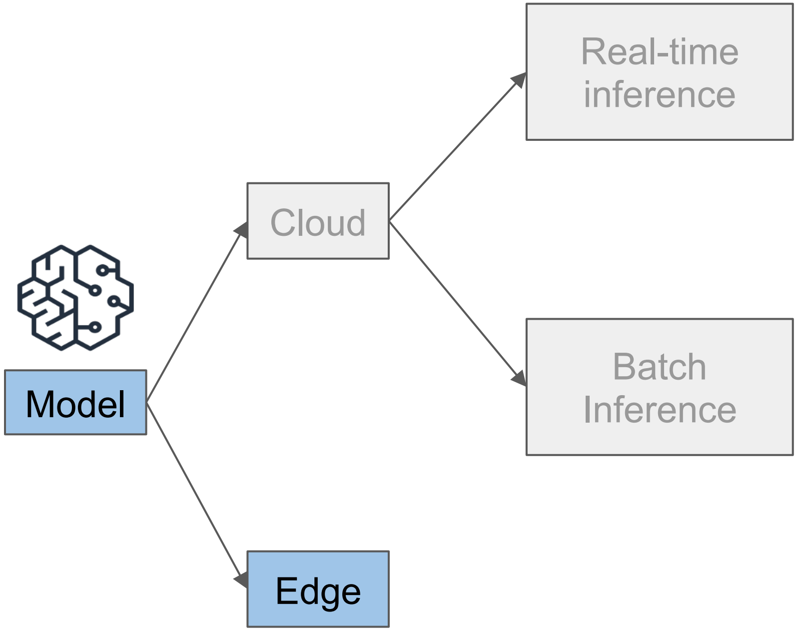
Deployment Options

Batch Inference - Product Review

Goal: *Identify vendors with potential quality issues*



Deployment Options

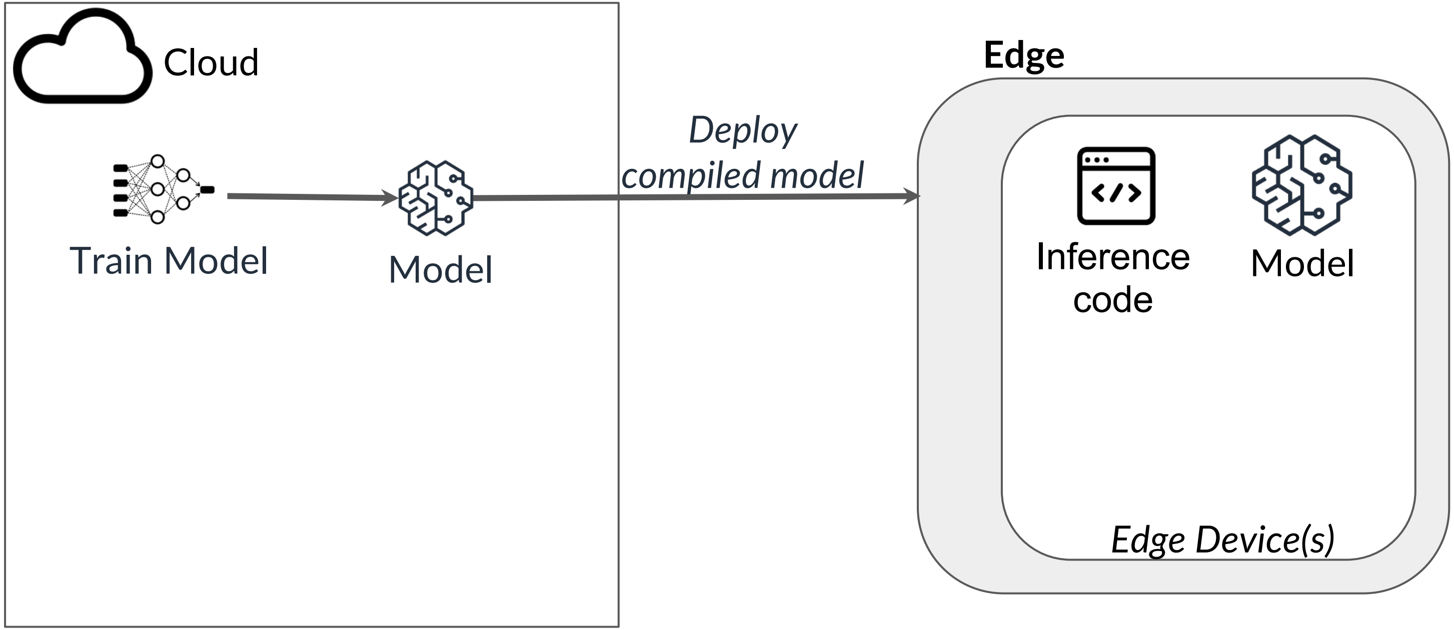


deployment to the edge which is an option that is not a cloud specific. But is a key consideration when deploying models closer to your users or in areas with poor network connectivity. In the case of edge deployments, you train your models in another environment in this case in the cloud and then optimize 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. Bring your model closer to where it will be used for prediction, so typical use cases here would be like manufacturing, where you have cameras on an assembly line.

Deployment Options

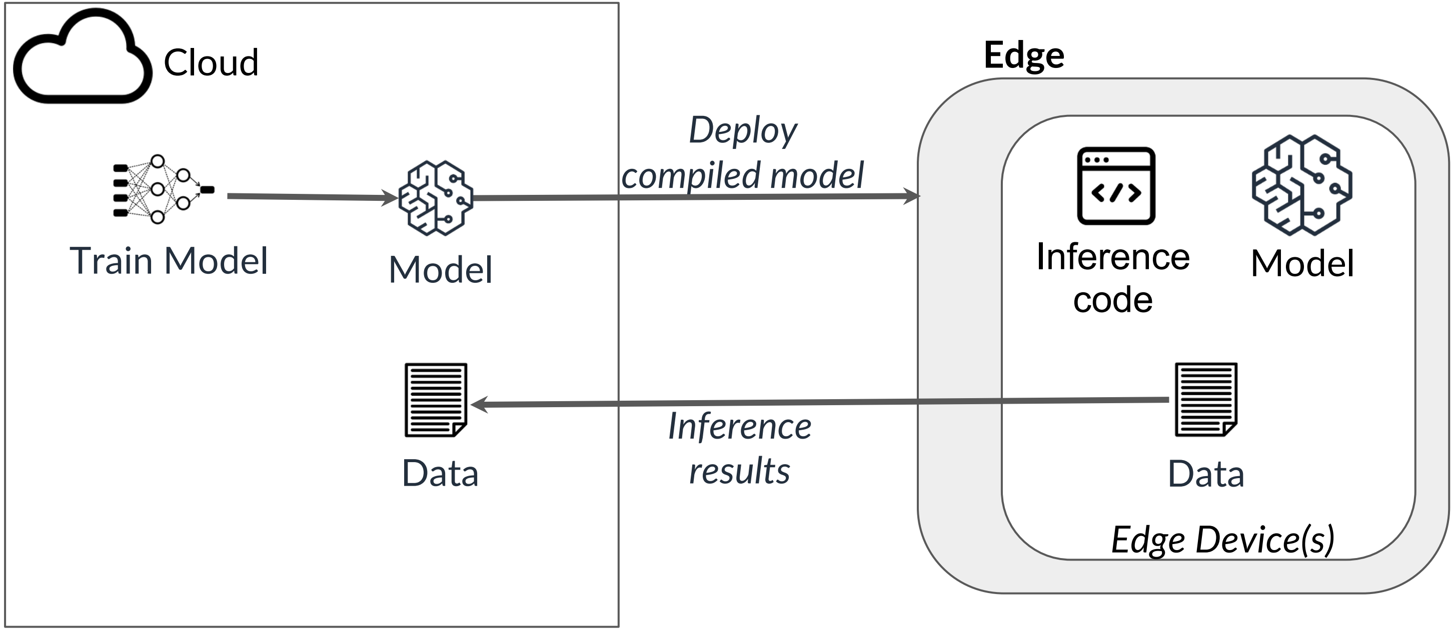
Edge

Use Sagemaker Neo



Deployment Options

Edge



Deployment Options

Choose the deployment option that best fits the use case

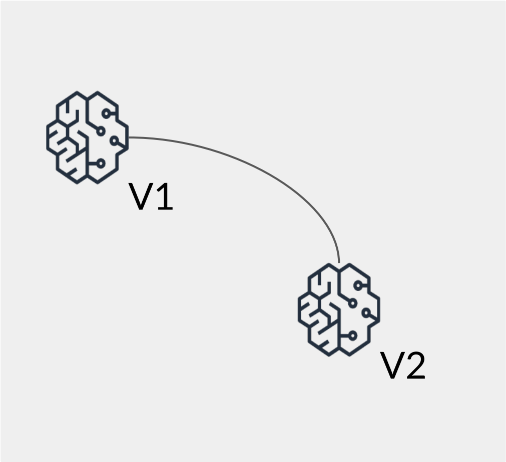
	Real-Time Inference	Batch Inference	Edge
When to use	Low latency real-time predictions (Ex. Interactive Recommenders)	Batch request & response prediction is acceptable for your use case (Ex. Forecasting)	Models need to deployed to edge devices (Ex. Limited connectivity, Internet of Things)
Cost	Persistent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

Model Deployment

Deployment Strategies

Deployment Strategies

Strategies to deploy new and updated models



Goals:

- Minimize risk
- Minimize downtime
- Measure model performance

Deployment Strategies

Common Strategies to deploy new and updated models

Blue/Green	Shadow/ Challenger	Canary	A/B	Multi-Armed Bandits

Deployment Strategies

Common Strategies to deploy new and updated models

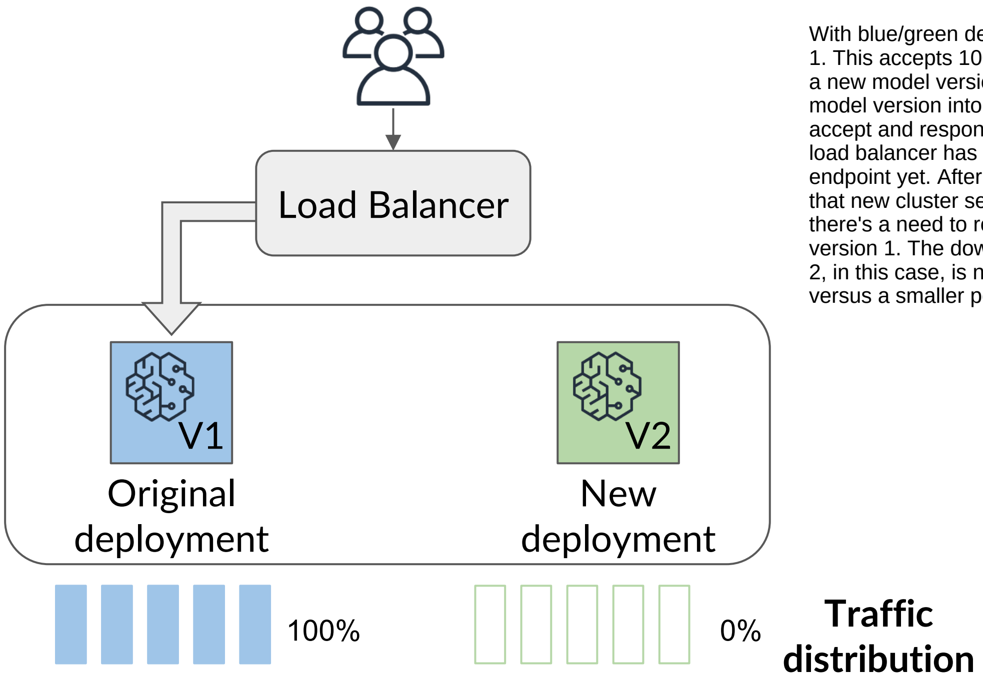
Blue/Green	Shadow/ Challenger	Canary	A/B	Multi-Armed Bandits

- Swap prediction request traffic
- Easy Rollback

With blue/green deployments, you deploy your new model version to a stack that conserved prediction and response traffic coming into an endpoint. Then when you're ready to have that new model version actually start to process prediction requests coming in, you swap the traffic to that new model version. This makes it easy to roll back because if there are issues with that new model or that new model version doesn't perform well, you can swap traffic back to the previous model version.

Deployment Strategies

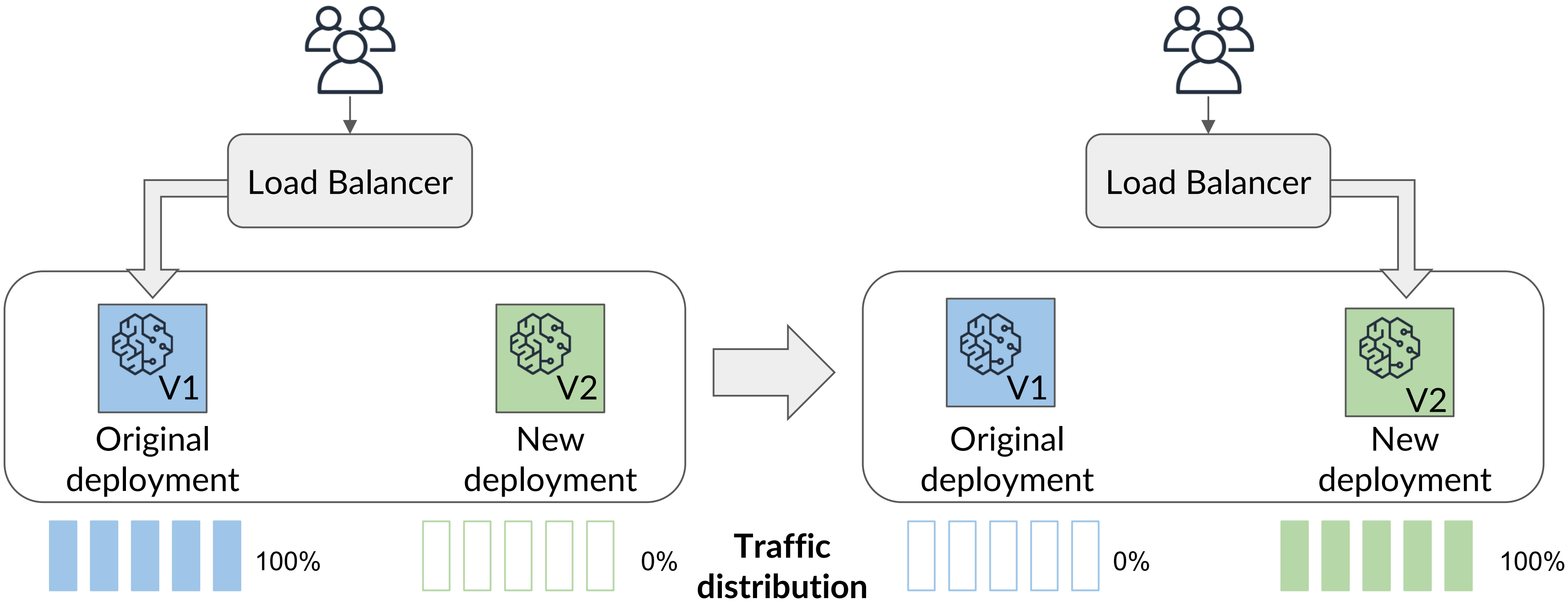
Blue/Green: Shift all traffic to the new model



With blue/green deployment, you have a current model version running in production. In this case, we have version 1. This accepts 100 percent of the prediction request traffic and responds with prediction responses. When you have a new model version to deploy, in this case, model version 2, you build a new server or container to deploy your model version into. This includes not only the new model version but also the code in the software that's needed to accept and respond to prediction requests. As you can see in this picture, the new model version is deployed, but the load balancer has not yet been updated to point to that new server hosting the model, so no traffic is hitting that endpoint yet. After the new model version is deployed successfully, you can then shift 100 percent of your traffic to that new cluster serving model version 2 by updating your load balancer. This strategy helps reduce downtime if there's a need to roll back and swap back to version 1 because you only need to re-point your load balancer back to version 1. 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.

Deployment Strategies

Blue/Green: Shift all traffic to the new model



Deployment Strategies

Common Strategies to deploy new and updated models

Blue/Green	Shadow/ Challenger	Canary	A/B	Multi-Armed Bandits

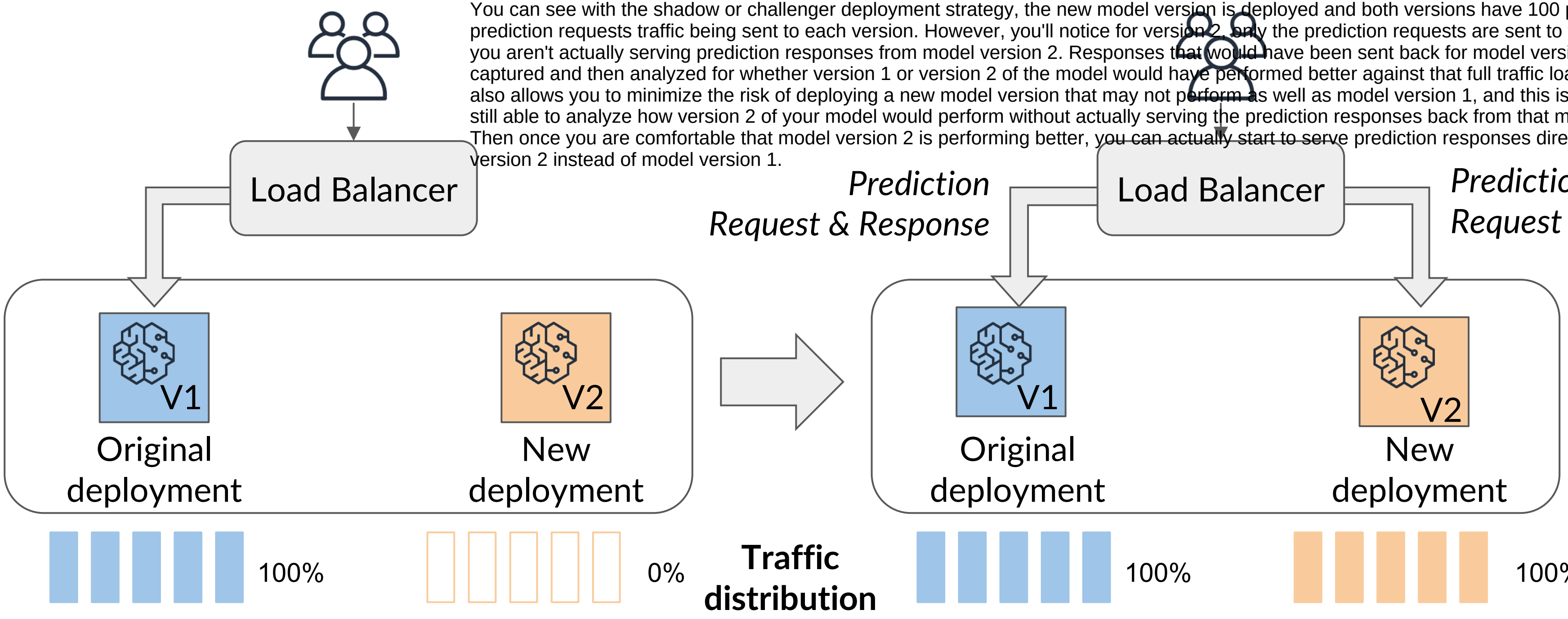
- Parallel prediction request traffic
- Validate new version without impact

This is often referred to as challenger models because in this case, you're running a new model version in production by letting the new version accept prediction requests to see how that new model would respond, but you're not actually serving the prediction response data from that new model version. This lets you validate the new model version with real traffic without impacting live prediction responses.

Deployment Strategies

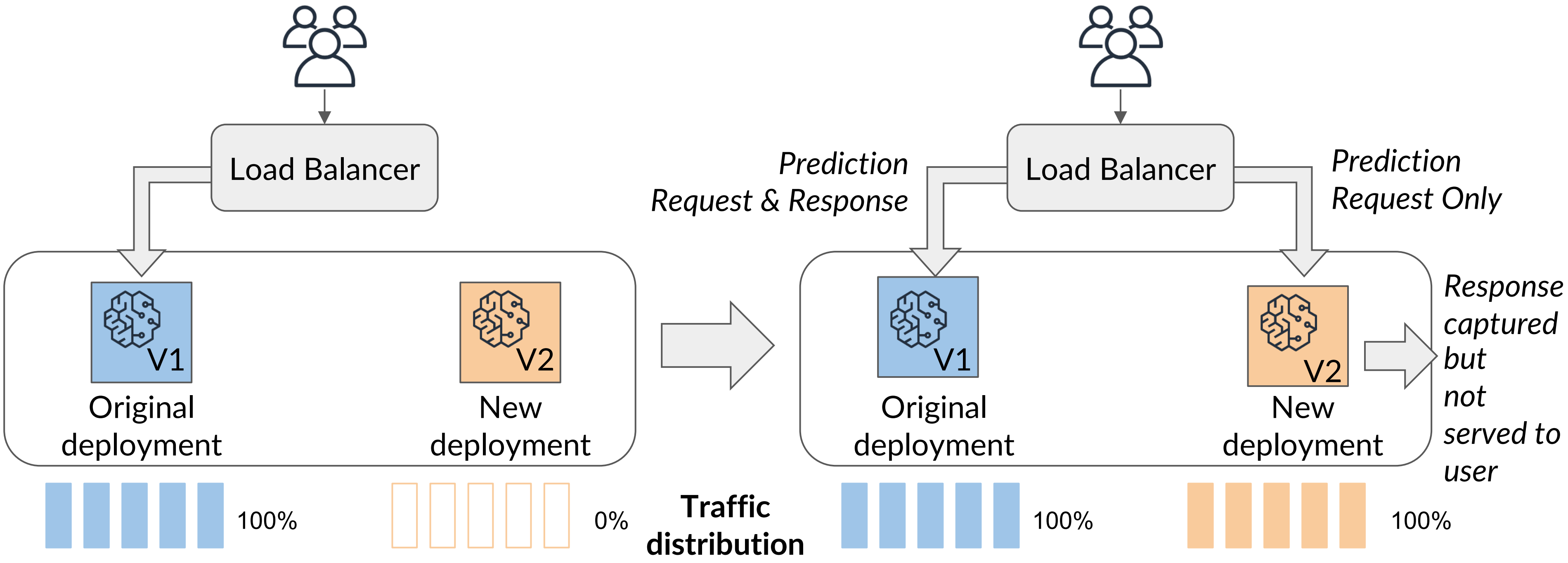
Shadow/Challenger: Run multiple versions in parallel with one serving live traffic

You can see with the shadow or challenger deployment strategy, the new model version is deployed and both versions have 100 percent of prediction requests traffic being sent to each version. However, you'll notice for version 2, only the prediction requests are sent to the model, and you aren't actually serving prediction responses from model version 2. Responses that would have been sent back for model version 2 are typically captured and then analyzed for whether version 1 or version 2 of the model would have performed better against that full traffic load. This strategy also allows you to minimize the risk of deploying a new model version that may not perform as well as model version 1, and this is because you're still able to analyze how version 2 of your model would perform without actually serving the prediction responses back from that model version. Then once you are comfortable that model version 2 is performing better, you can actually start to serve prediction responses directly from model version 2 instead of model version 1.



Deployment Strategies

Shadow/Challenger: Run multiple versions in parallel with one serving live traffic



Deployment Strategies

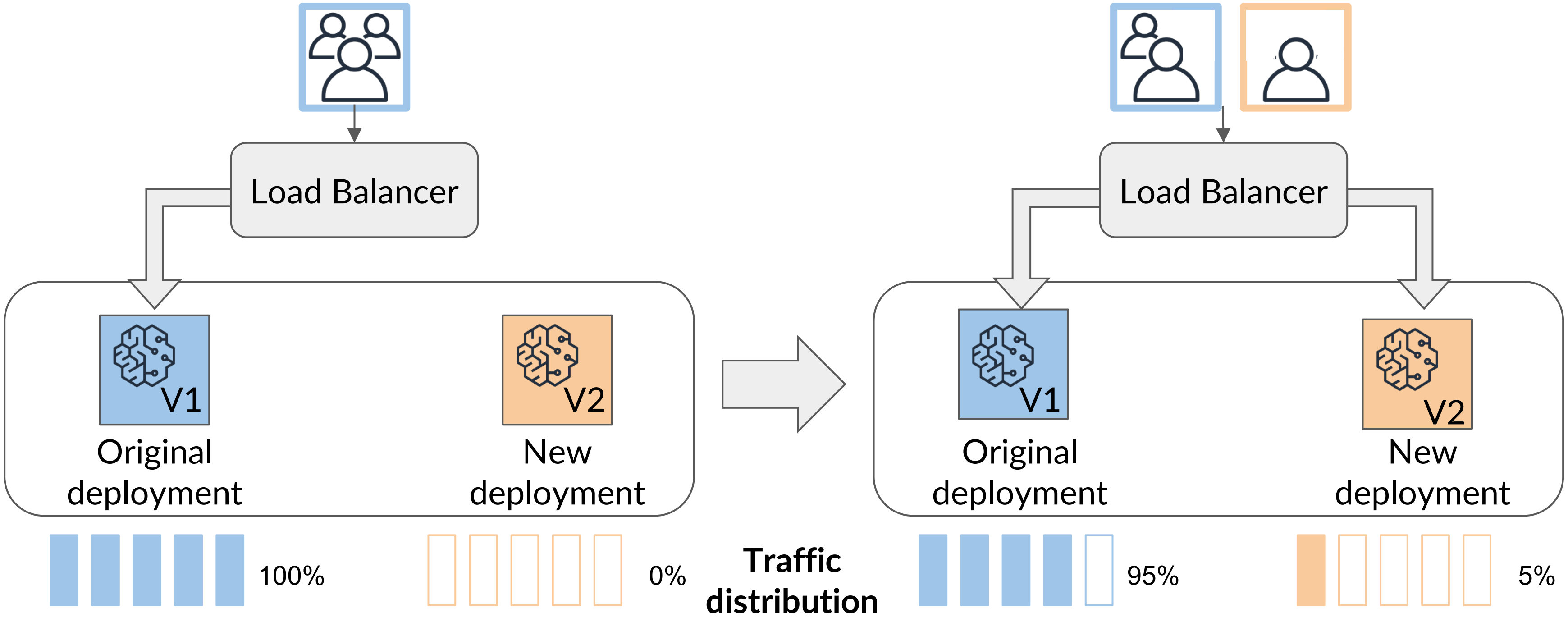
Common Strategies to deploy new and updated models

Blue/Green	Shadow/ Challenger	Canary	A/B	Multi-Armed Bandits

- Split traffic
- Target smaller specific users/groups
- Shorter validation cycles
- Minimize risk of low performing model

Deployment Strategies

Canary: Split traffic to compare model versions with target groups/users



Deployment Strategies

Common Strategies to deploy new and updated models

Blue/Green	Shadow/ Challenger	Canary	A/B	Multi-Armed Bandits

- Split traffic
- Target larger users/groups ~OR~ Distribute % of traffic
- Longer validation cycles
- Minimize risk of low performing model

Deployment Strategies

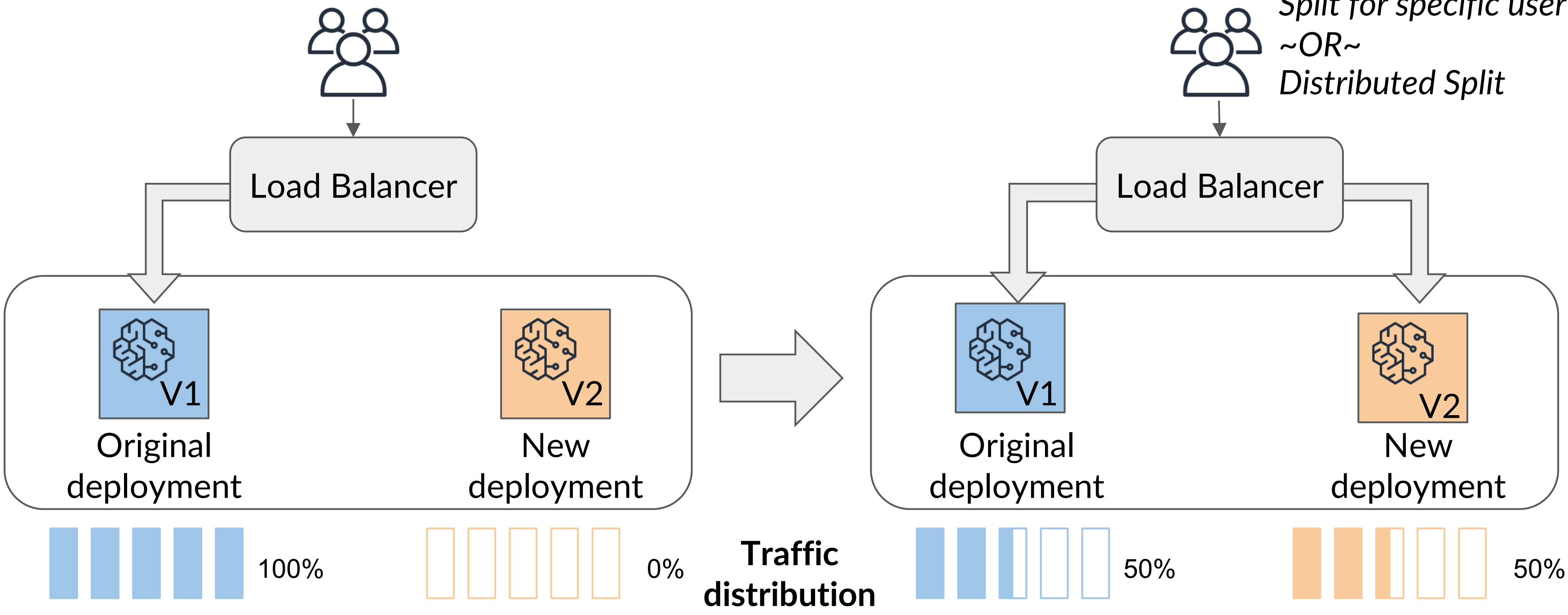
A/B: Split traffic to compare model versions

With A/B testing, again, you're also splitting your traffic to compare model versions. However, here you split traffic between those larger groups for the purpose of comparing different model versions in live production environments. Here, you typically do a larger split across users. So 50 percent one model version, 50 percent the other model version. Then you can also perform A/B testing against more than two model versions as well, although it's not shown here. While A/B testing seemed similar to canary deployments, A/B testing tests those larger groups, like I mentioned, and typically runs for longer periods of time than canary deployments. A/B tests are focused on gathering live data about different model versions.

Split for specific user groups

~OR~

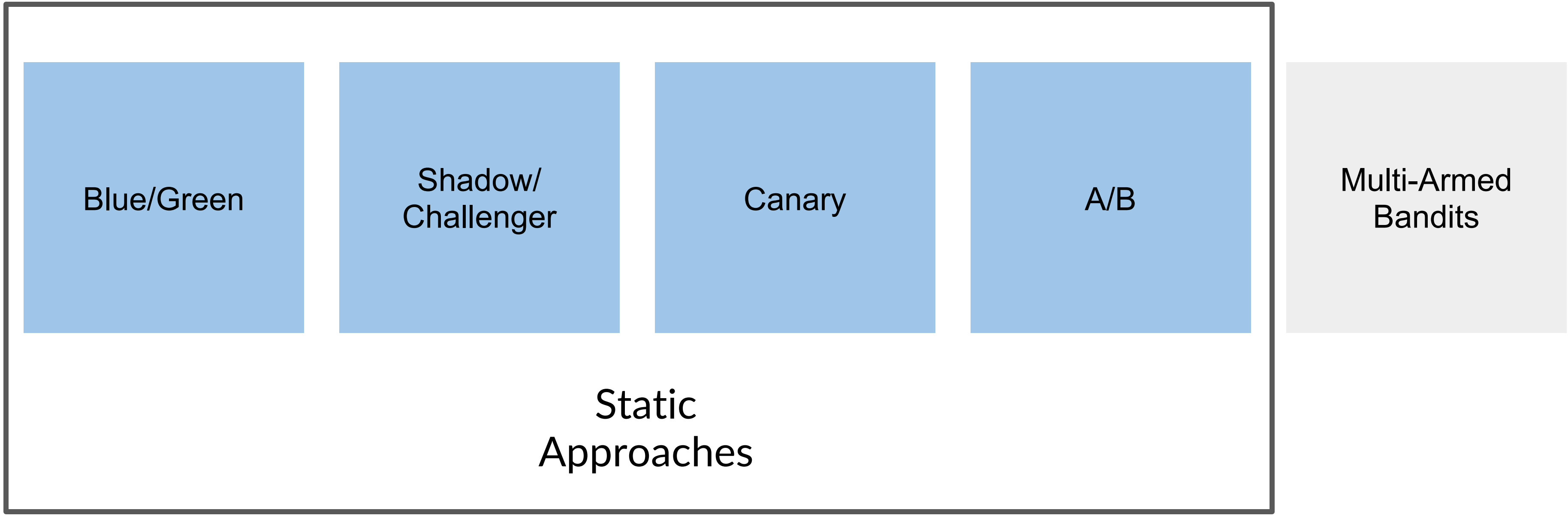
Distributed Split



Longer period of time to understand the performance with seasonality and other variations.

Deployment Strategies

Common Strategies to deploy new and updated models



Deployment Strategies

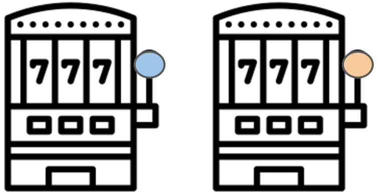
Common Strategies to deploy new and updated models

Static Approaches				Dynamic Approach
Blue/Green	Shadow/ Challenger	Canary	A/B	Multi-Armed Bandits

Deployment Strategies

Multi-Armed Bandits (MABs)

Multi-armed bandits use 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.



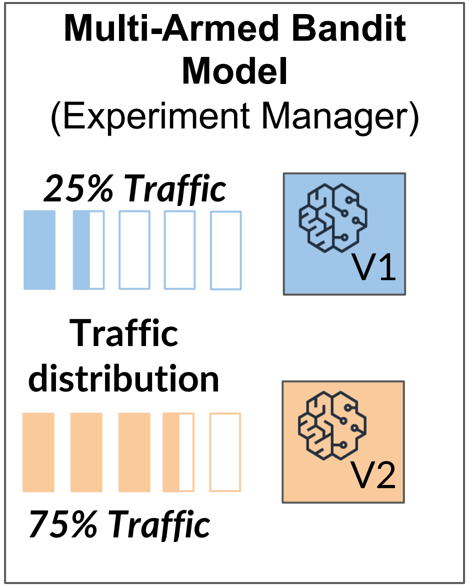
- **Dynamic testing method for model version using Reinforcement Learning**

- **Exploit & Explore**
  - **Exploit:** Reward the winning model with more traffic
  - **Explore:** Continue to send traffic to the non-winning model(s) in case behavior changes

Deployment Strategies

Multi-Armed Bandits: Dynamically shift traffic to the winning model

In this implementation, you first have an experiment manager, which is basically a model that uses reinforcement learning to determine how to distribute traffic between your model versions. This model chooses the model version to send traffic to based on the current reward metrics and the chosen exploit explore strategy. Exploitation refers to continuing to send traffic to that winning model, whereas exploration allows for routing traffic to other models to see if they can eventually catch up or perform as well as the other model.

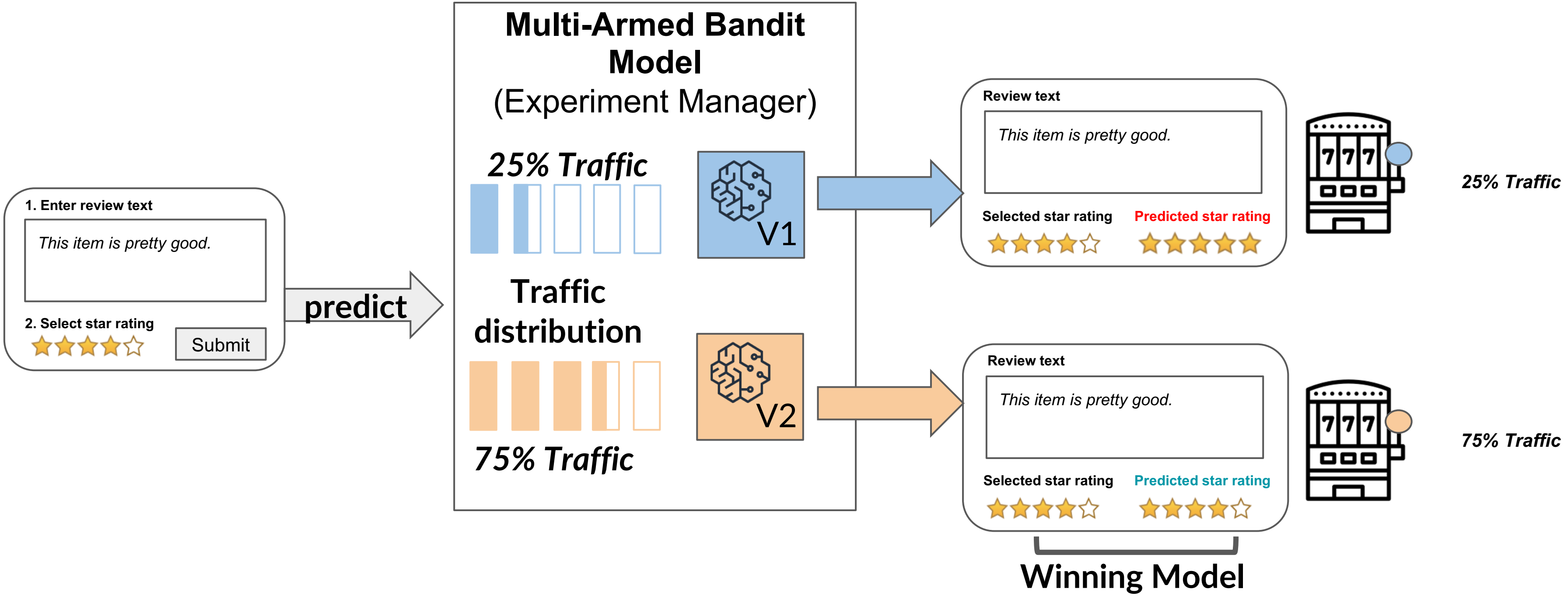


25% Traffic

75% Traffic

Deployment Strategies

Multi-Armed Bandits: Dynamically shift traffic to the winning model



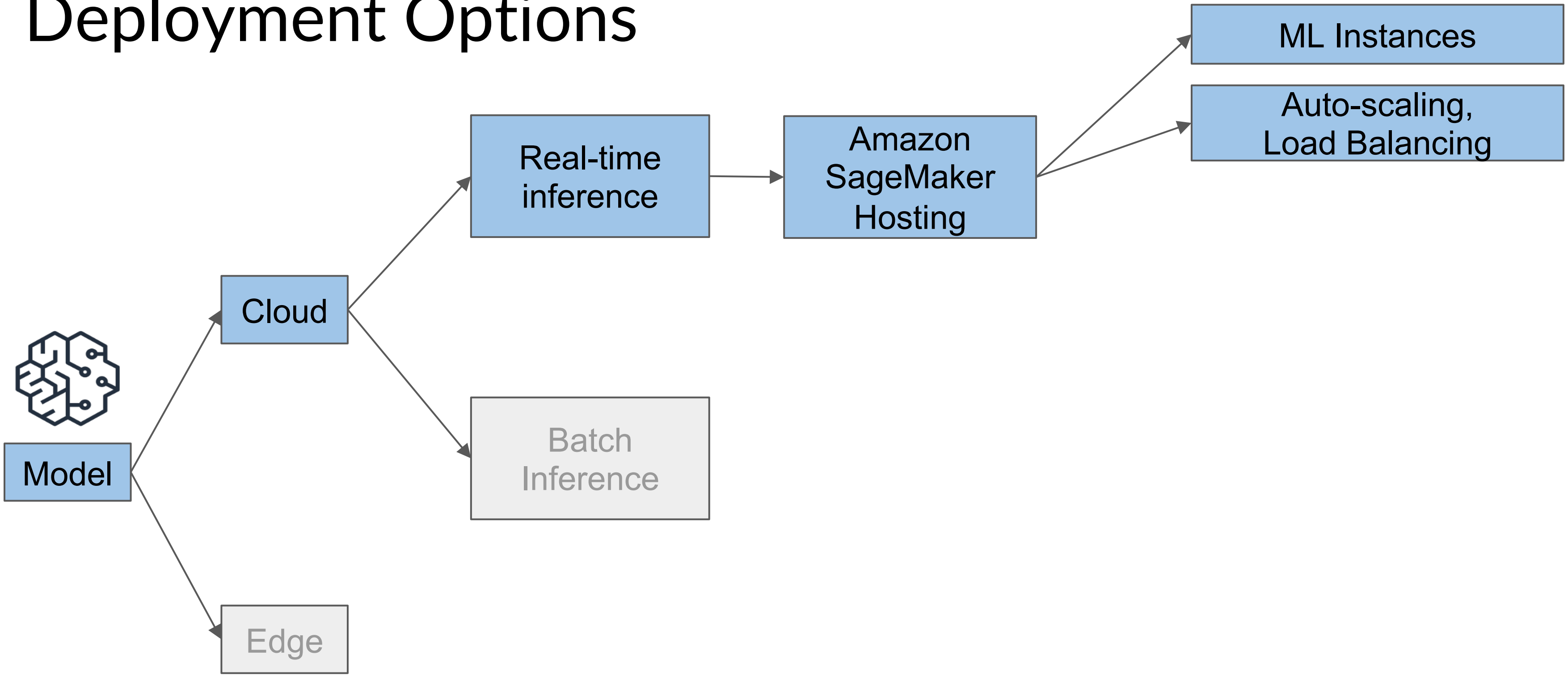
Amazon SageMaker Hosting

Real-Time Inference

Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

Deployment Options



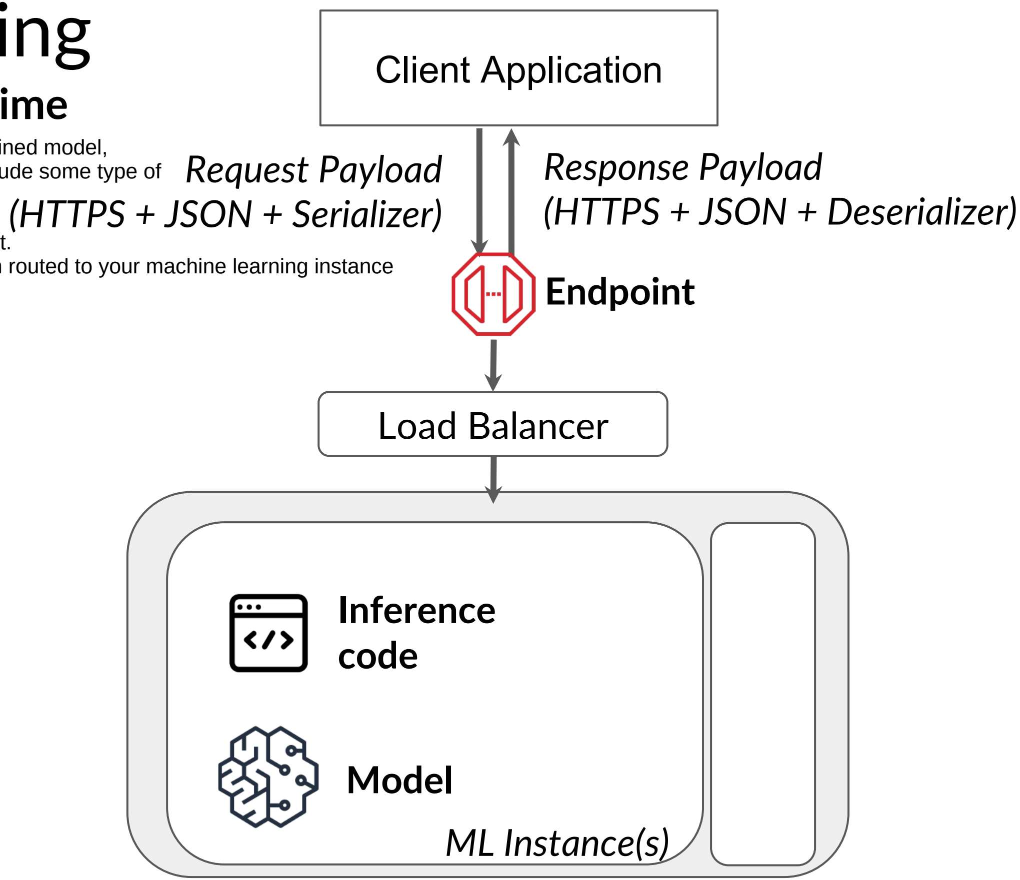
Amazon SageMaker Hosting

Deploy models to serve predictions in real-time

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.

Your model can then be consumed by client applications through real time invoke API request. The request payload sent when you invoke the endpoint is routed to a load balancer and then routed to your machine learning instance or instances that are hosting your models for prediction.

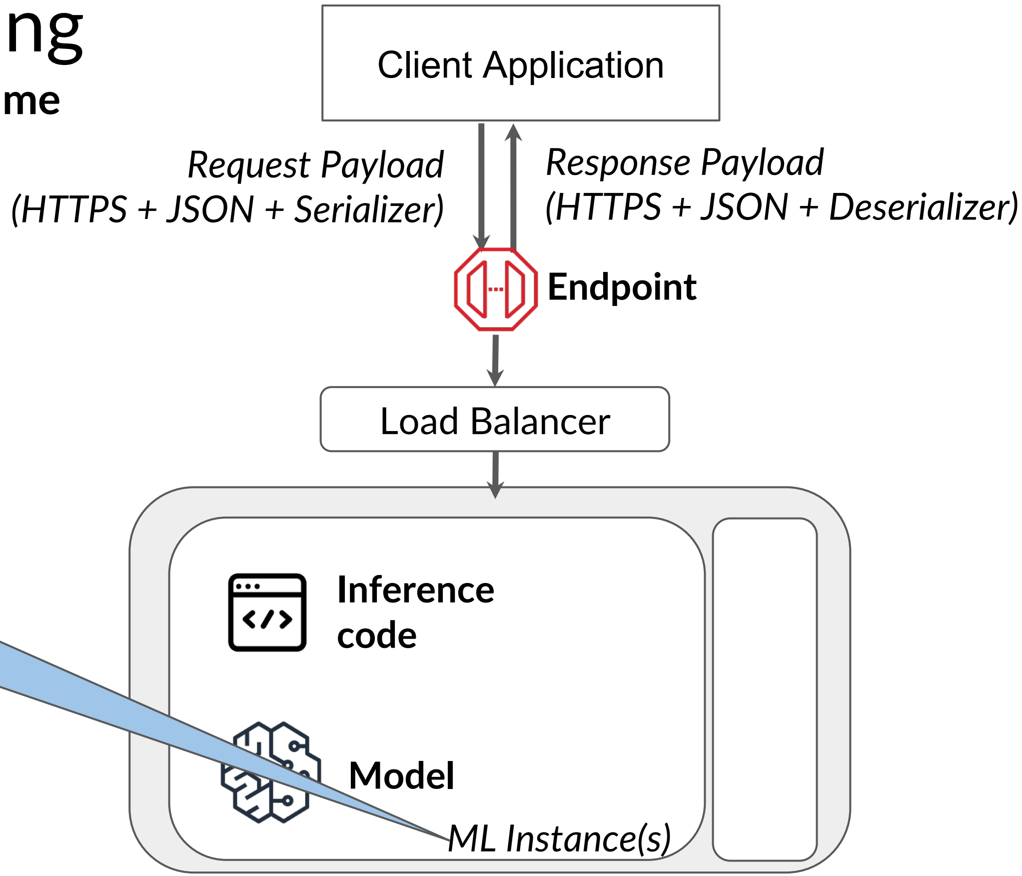
- Optimized for **low latency** of model predictions
- **Example:** As product reviews are coming in through online channels, you want to predict the sentiment for immediate action



Amazon SageMaker Hosting

Deploy models to serve predictions in real-time

You choose:
- Instance Type
- Instance Size
- Number of Instances
- Autoscaling Options



Amazon SageMaker Hosting

Options to deploy models to serve predictions in real-time

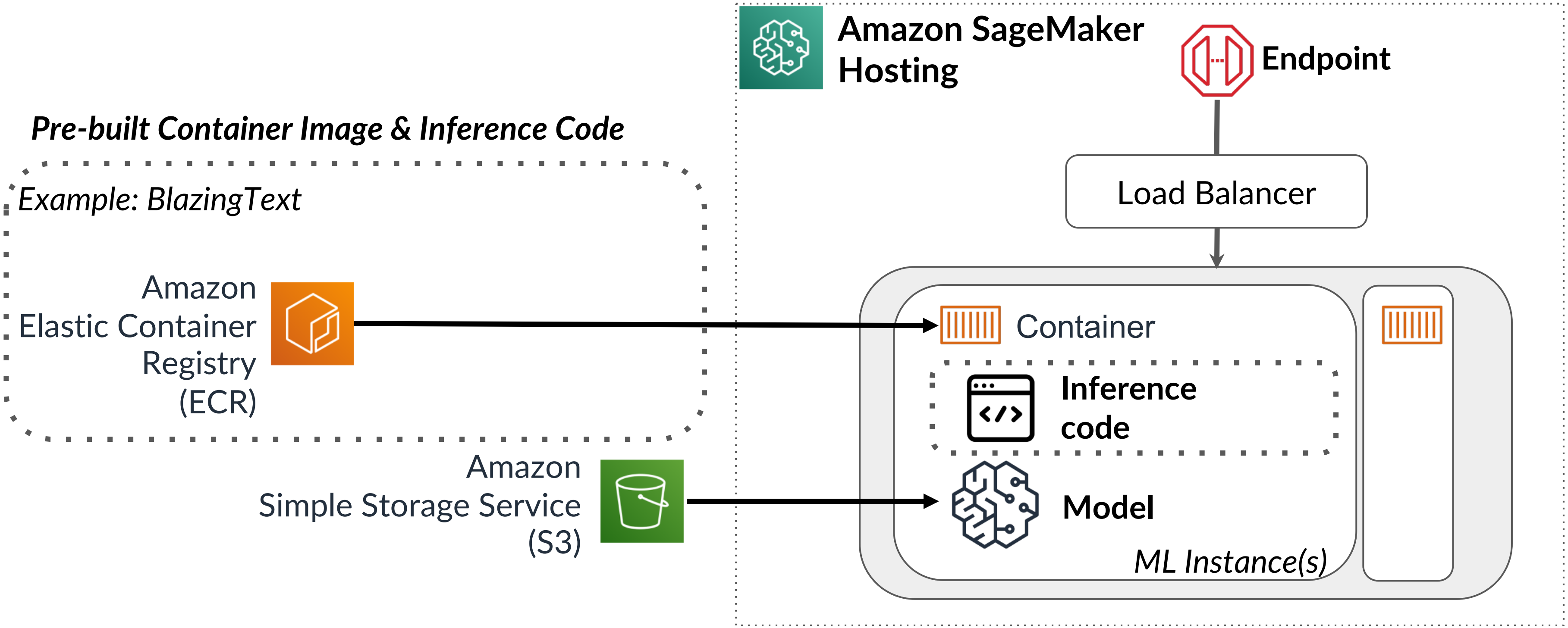
Amazon SageMaker	Amazon SageMaker	Amazon SageMaker
Built-in Algorithms	*Bring Your Own Script*	*Bring Your Own Container*

Less Code

More Customizable

Amazon SageMaker Hosting

Built-In Algorithm: Pre-built code & serving container



Amazon SageMaker Hosting

Options to deploy models to serve predictions in real-time

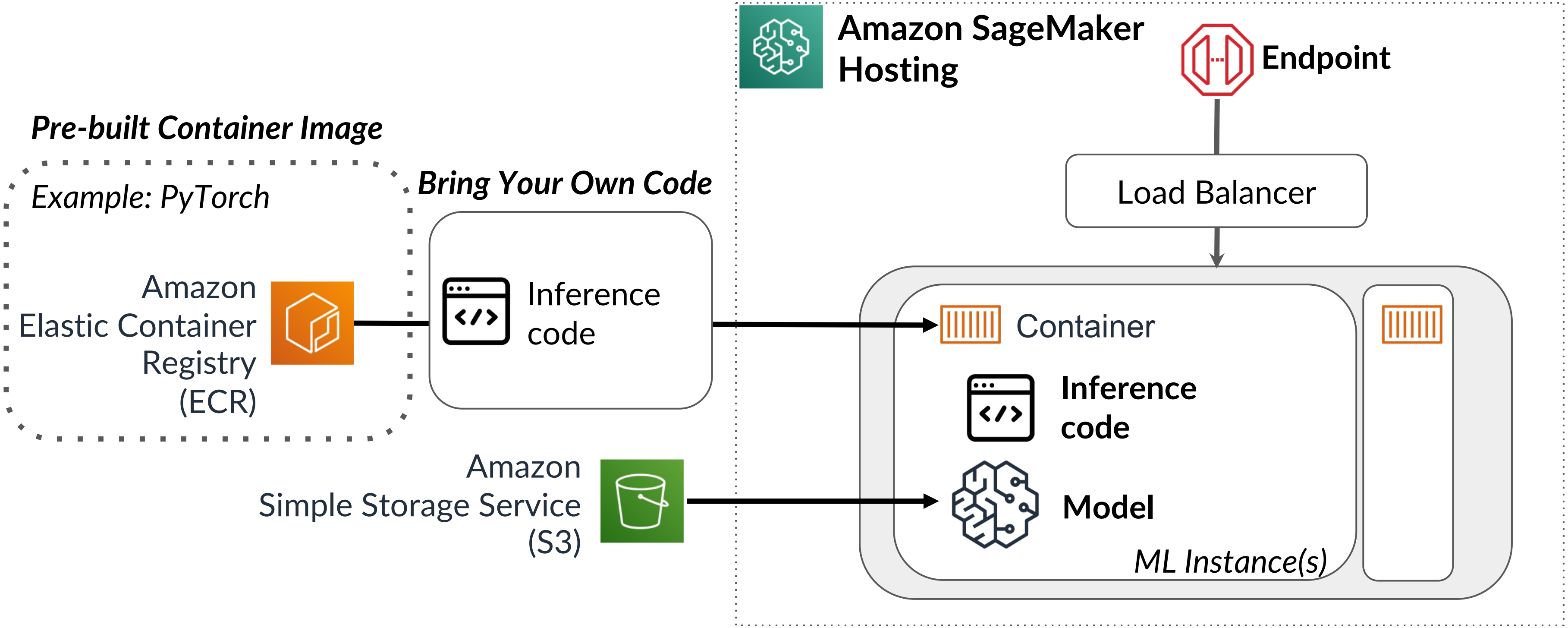
Amazon SageMaker	Amazon SageMaker	Amazon SageMaker
Built-in Algorithms	*Bring Your Own Script*	*Bring Your Own Container*

Less Code

More Customizable

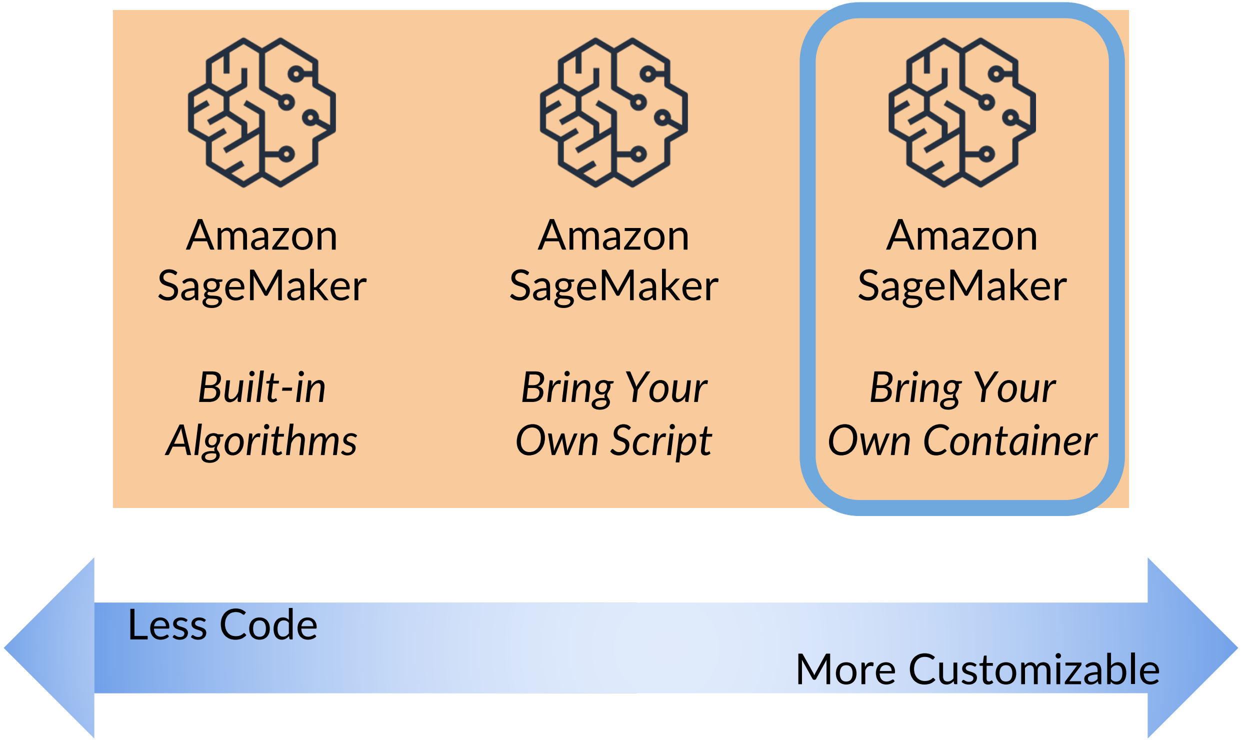
Amazon SageMaker Hosting

Bring Your Own Script: Pre-built container & Bring your own code



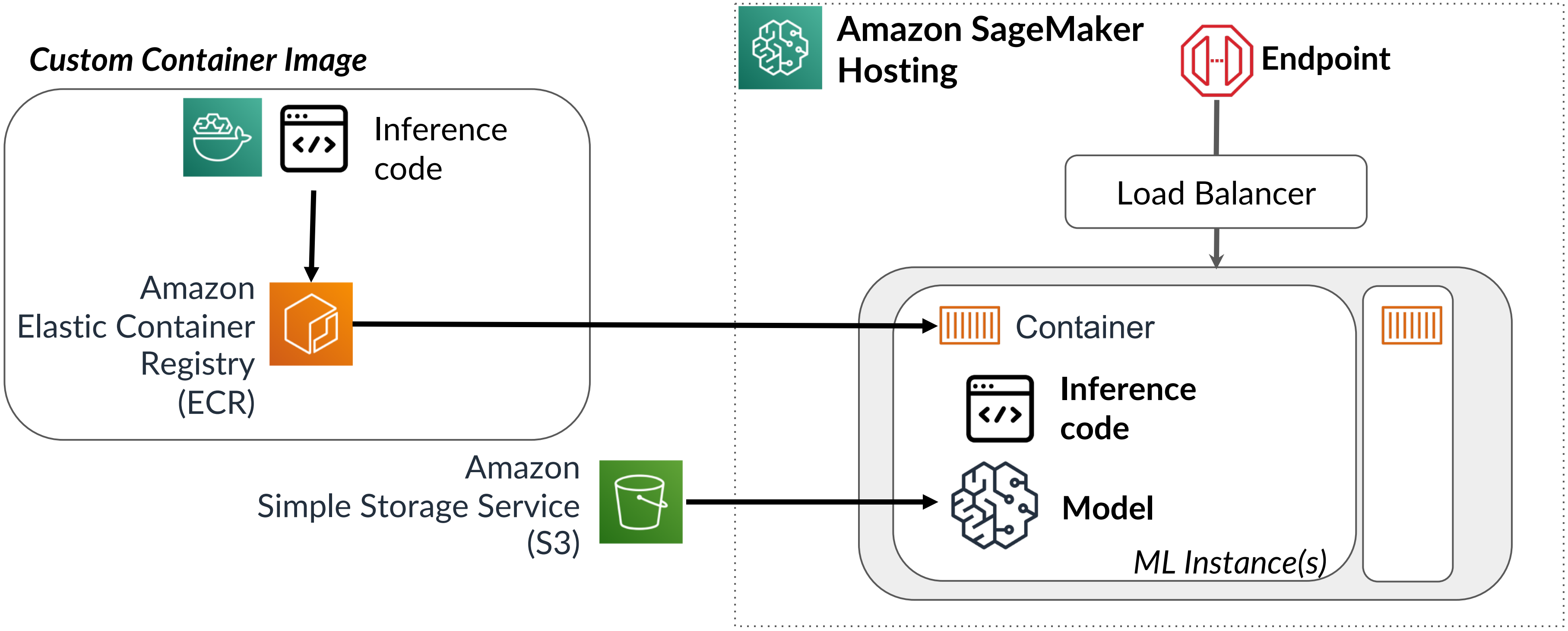
Amazon SageMaker Hosting

Options to deploy models to serve predictions in real-time



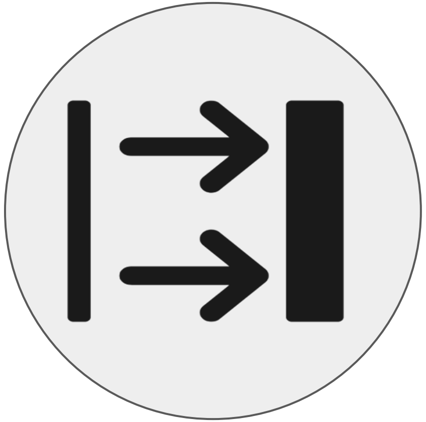
Amazon SageMaker Hosting

Bring Your Own Container: Bring your own code & custom container



Amazon SageMaker Hosting

Autoscaling Endpoints

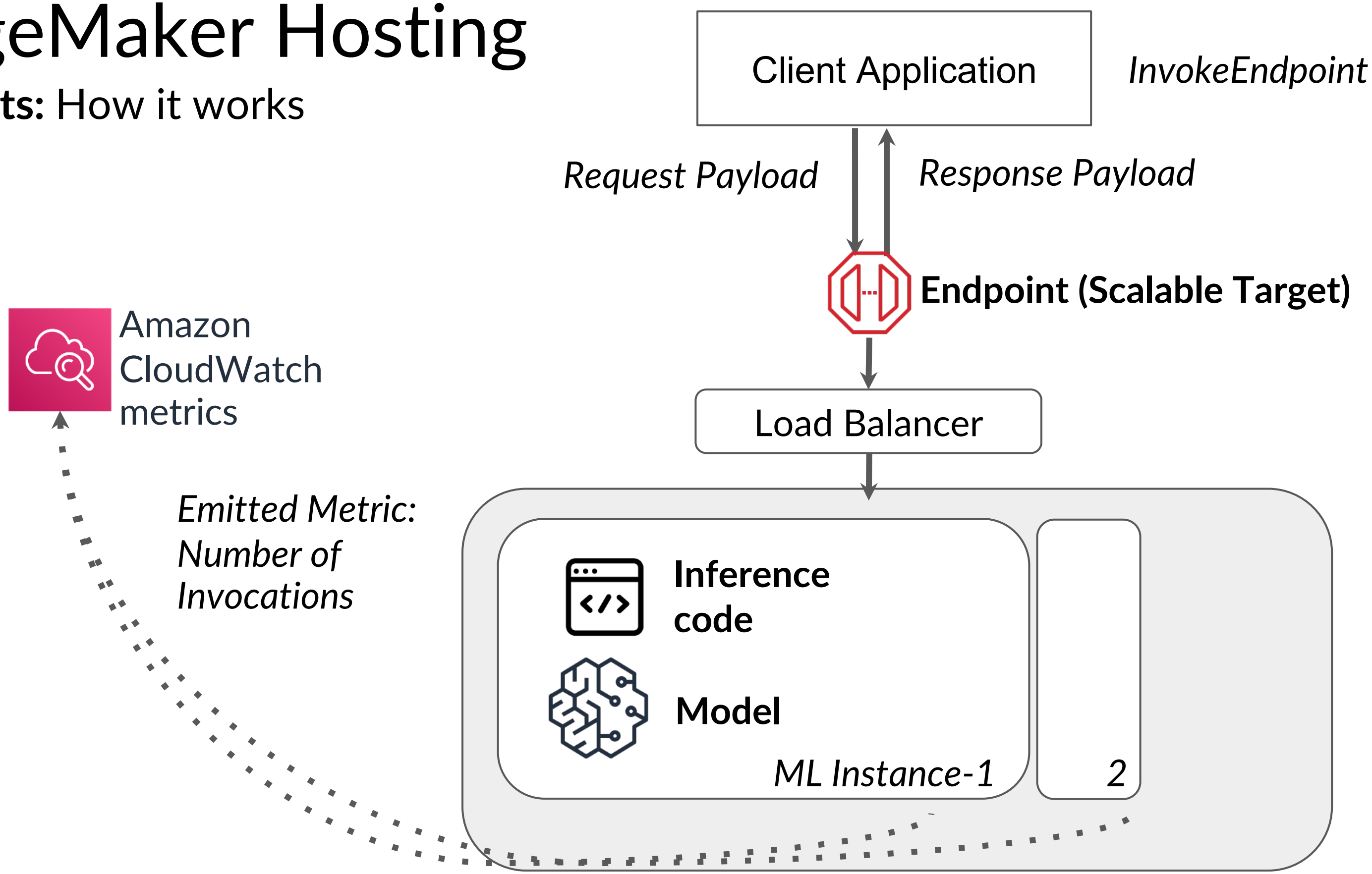


Why?

- Ensure you can meet the demands of your workload
- Cost optimization

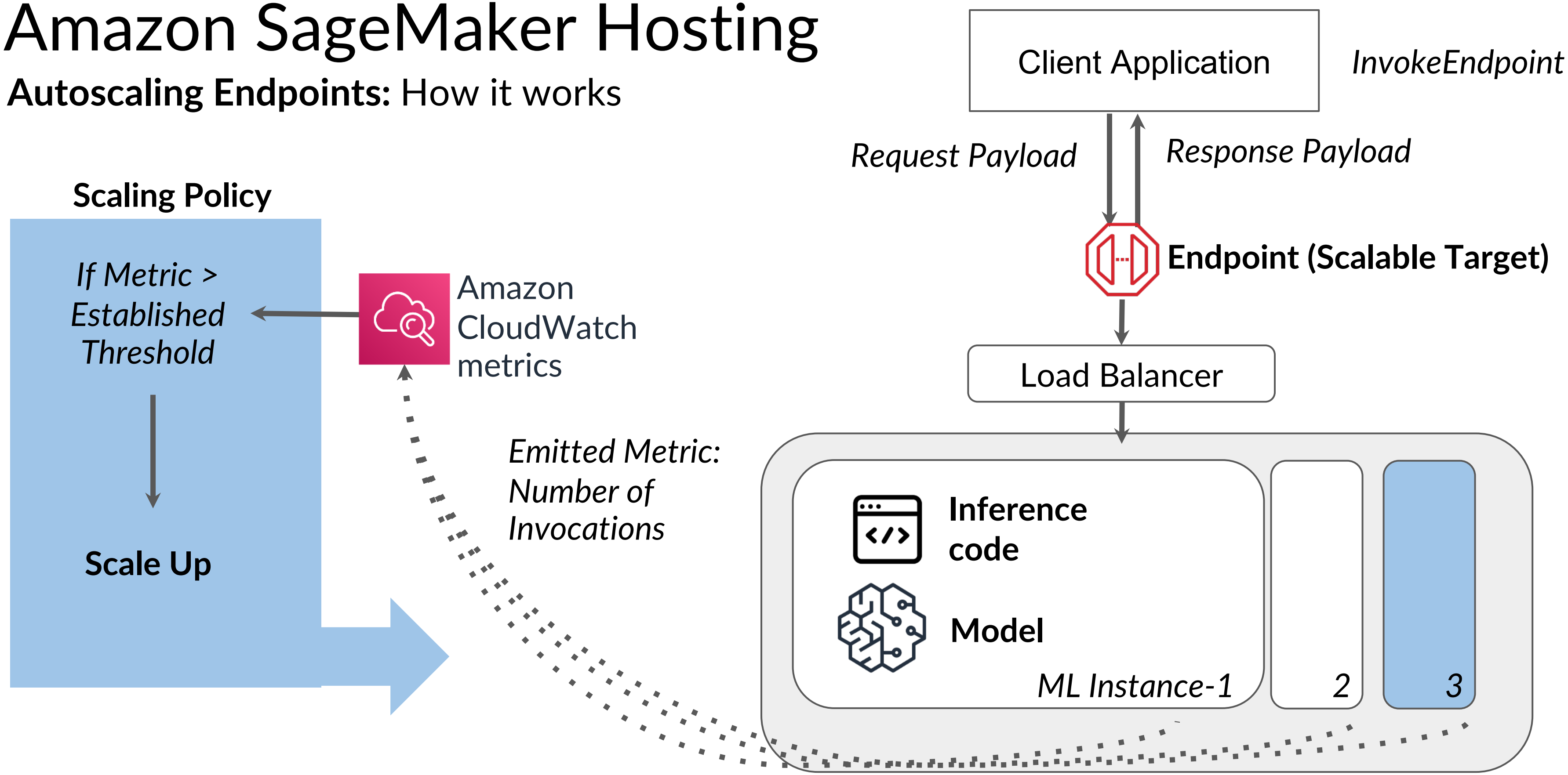
Amazon SageMaker Hosting

Autoscaling Endpoints: How it works



Amazon SageMaker Hosting

Autoscaling Endpoints: How it works



Autoscale Amazon SageMaker Endpoints

Register Scalable Target

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.

```
autoscale.register_scalable_target(
         ServiceNamespace="sagemaker",
    ResourceId="endpoint/" + endpoint_name,
    ScalableDimension="sagemaker:variant:DesiredInstanceCount",
    MinCapacity=1,
    MaxCapacity=2,
    RoleARN=role,
    SuspendedState={
        "DynamicScalingInSuspended": False,
        "DynamicScalingOutSuspended": False,
        "ScheduledScalingSuspended": False,
    })
```

Autoscale Amazon SageMaker Endpoints

Register Scalable Target → **Define Scaling Policy**

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.

```
scaling_policy = {
        "TargetValue": 2.0,
        "PredefinedMetricSpecification": {
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance",
        },
        "ScaleOutCooldown": 60,
        "ScaleInCooldown": 300,
        },
}
```

Scaling Metric

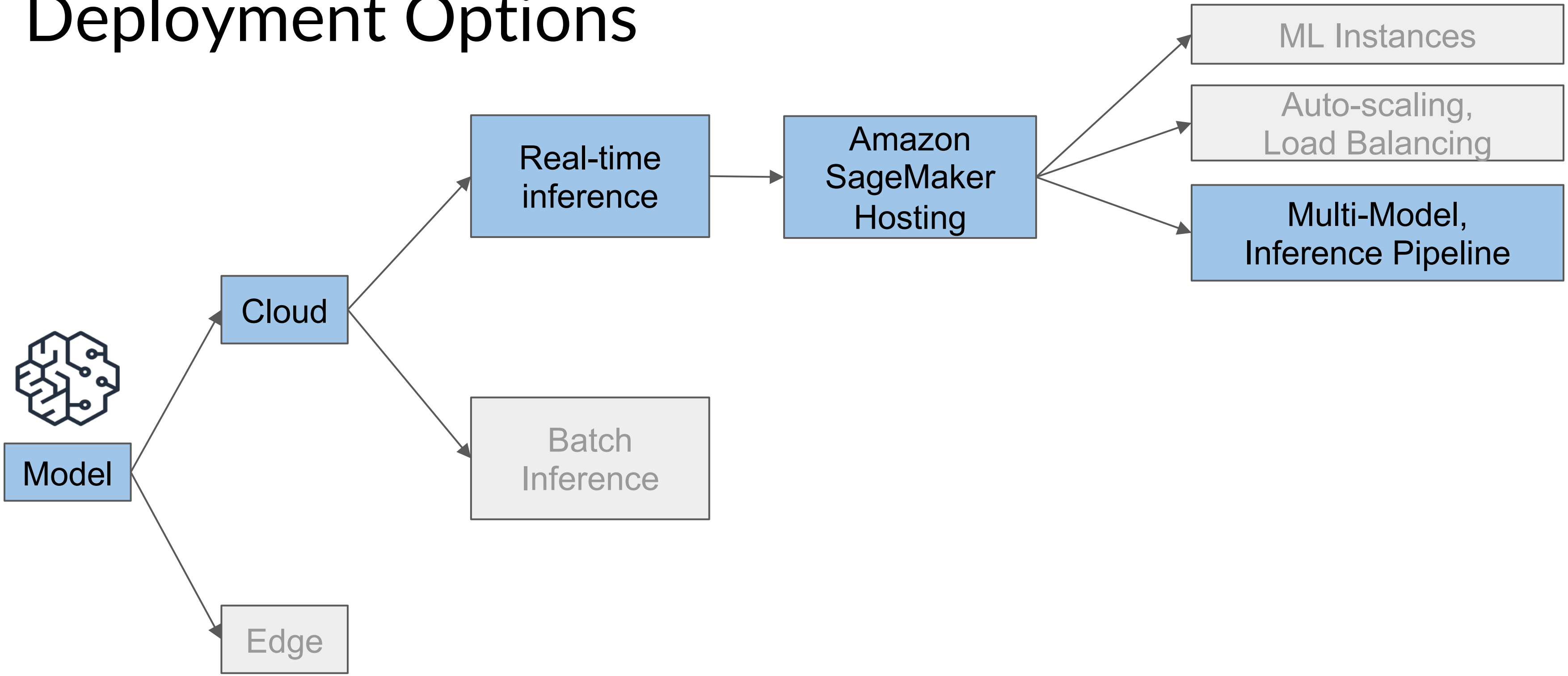
Wait time, in seconds, before beginning another **scale out** activity after last one completes

Wait time, in seconds, before beginning another **scale in** activity after last one completes

Autoscale Amazon SageMaker Endpoints


```
autoscale.put_scaling_policy(
          PolicyName=...,
          ServiceNamespace="sagemaker",
          ResourceId="endpoint/" + endpoint_name,
          ScalableDimension="sagemaker:variant:DesiredInstanceCount",
          PolicyType="TargetTrackingScaling",
          TargetTrackingScalingPolicyConfiguration=scaling_policy)
```

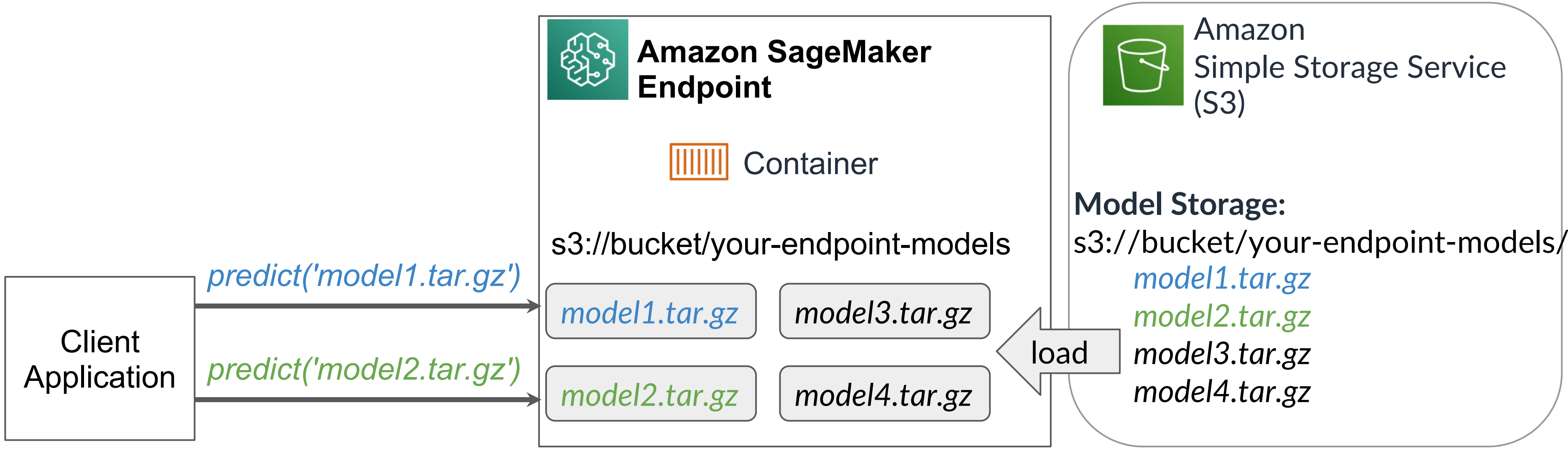
Deployment Options



Advanced Deployment Options

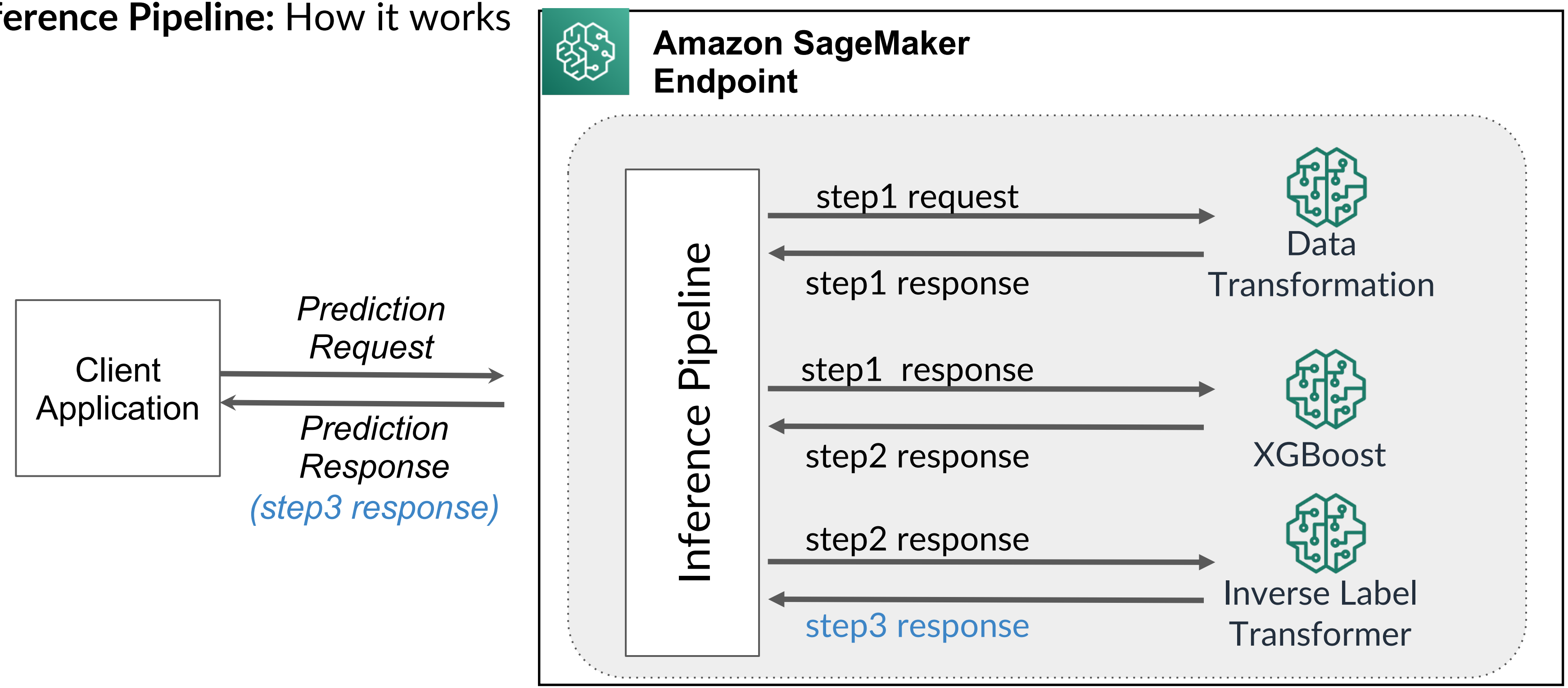
Multi-Model Endpoints: How it works

Deploy Multiple Models to a Single Endpoint



Advanced Deployment Options

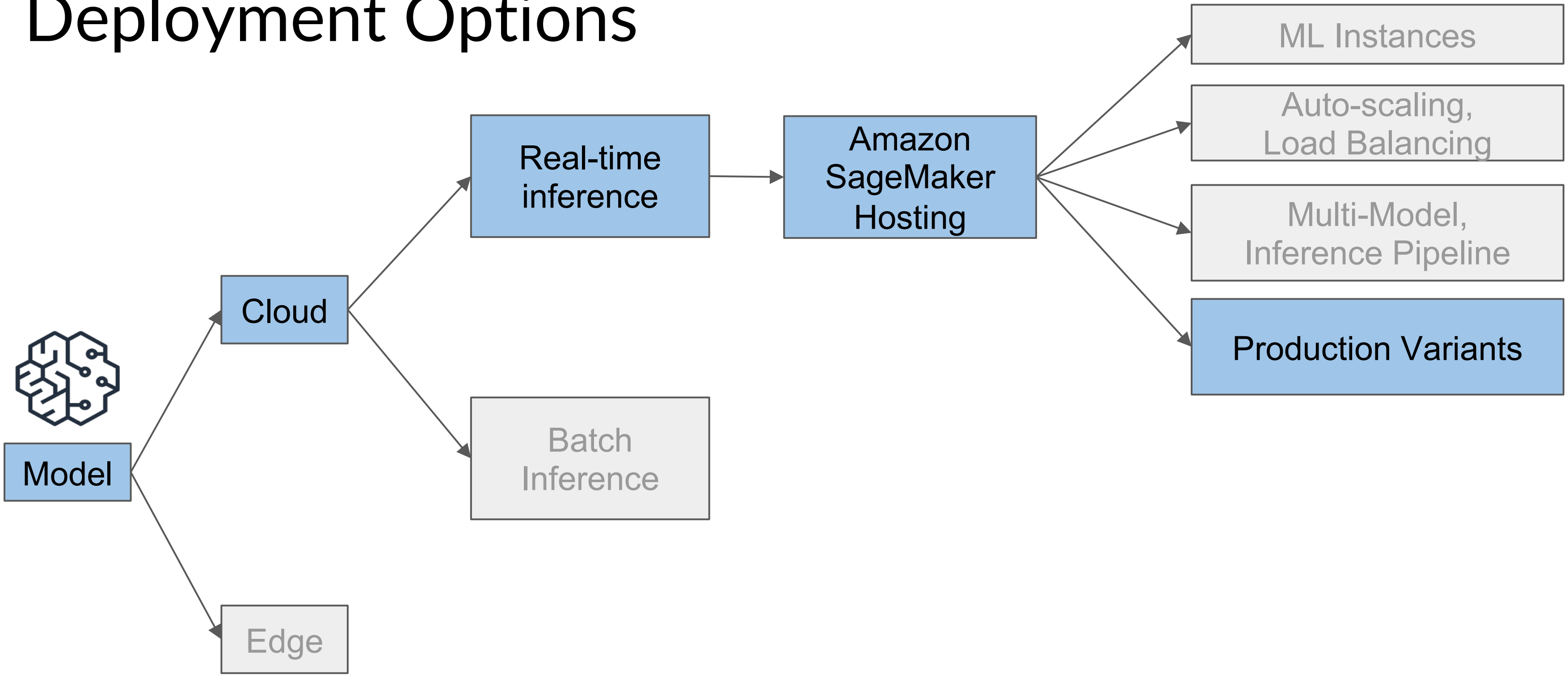
Inference Pipeline: How it works



Amazon SageMaker Hosting

Real-Time Inference
Production Variants

Deployment Options



Amazon SageMaker Production Variants

What is a Production Variant?

SageMaker Model

Configuration Example(s):

- *Amazon S3 model artifact*
- *Inference image(s)*
- *Execution AWS IAM Role*
- ***Model name***

+

Hosting Resources Configuration

Example:

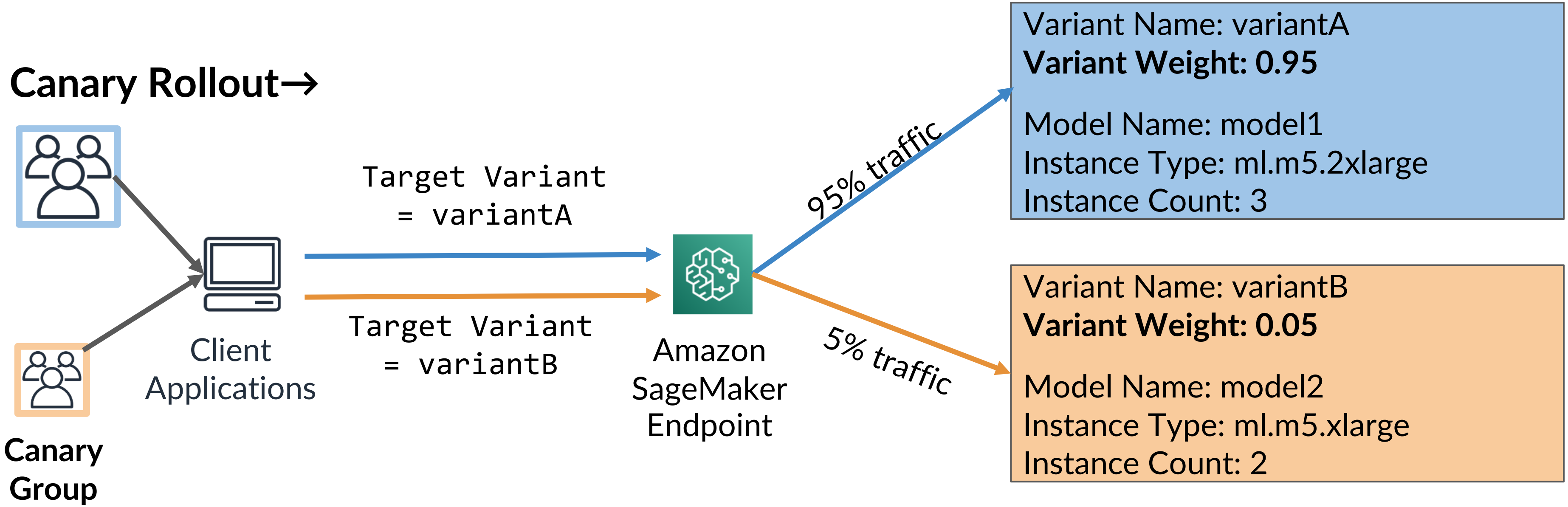
- *Number of instances*
- *Instance type*
- ***Model name***
- *Variant name*
- *Variant weight*

=

Production Variant

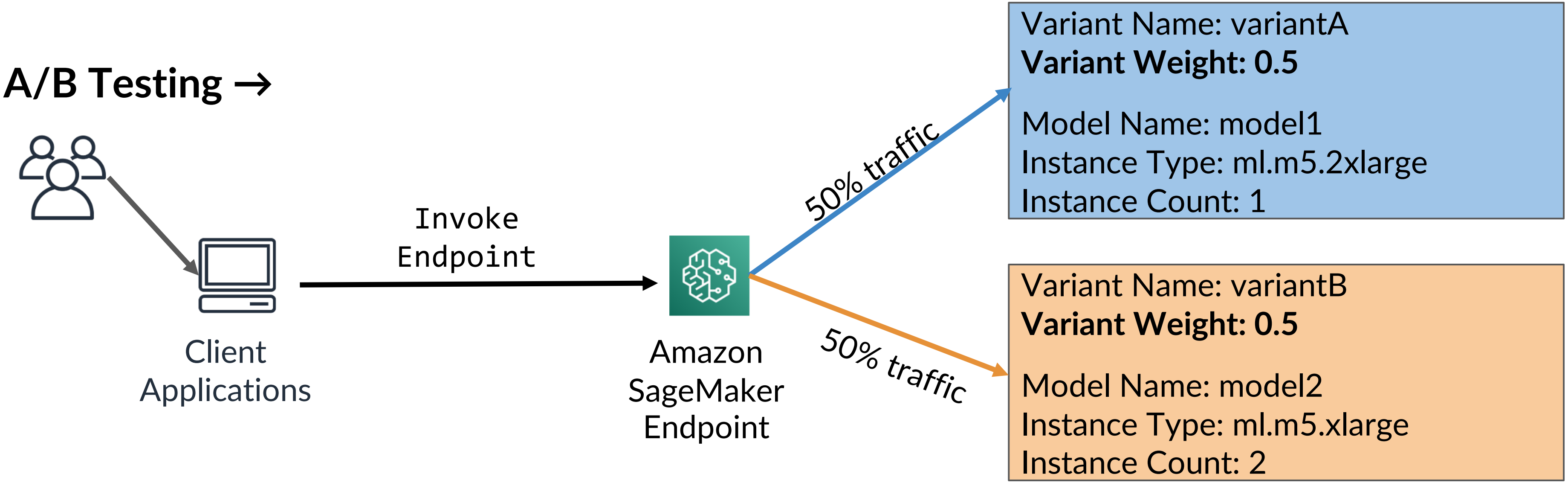
Amazon SageMaker Production Variants

Using Production Variants for a Canary Rollout



Amazon SageMaker Production Variants

Using Production Variants for A/B Testing



Amazon SageMaker Hosting

Using Production Variants for A/B Testing with Bring-Your-Own Script

Amazon SageMaker	Amazon SageMaker	Amazon SageMaker
Built-in Algorithms	*Bring Your Own Script*	*Bring Your Own Container*

Less Code

More Customizable

Using Production Variants for A/B Testing

A/B Testing with PyTorch Bring-Your-Own Script

Construct Docker Image URI

```
import sagemaker

inference_image_uri = sagemaker.image_uris.retrieve(
     framework=..., # PyTorch, TensorFlow, etc...
     version='1.6.0',
     instance_type='ml.m5.xlarge',
     py_version='py3',
     image_scope='inference'
)
```

Using Production Variants for A/B Testing

A/B Testing with PyTorch Bring-Your-Own Script

Construct Docker Image URI → Create SageMaker Models

```
sm.create_model(
        name=model_name_a,
        ...
)
```

```
sm.create_model(
        name=model_name_b,
        ...
)
```

SageMaker Model A

SageMaker Model B

Using Production Variants for A/B Testing

A/B Testing with PyTorch Bring-Your-Own Script

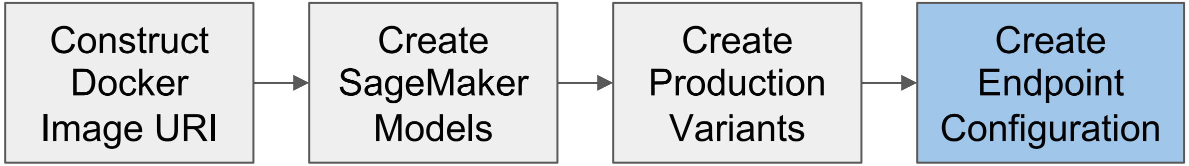
Construct Docker Image URI → Create SageMaker Models → **Create Production Variants**

```
from sagemaker.session \
    import production_variant

variantA = production_variant(
         model_name=...,
     instance_type=...,
         initial_instance_count=1,
         variant_name='VariantA',
         initial_weight=50,
)
```

```
from sagemaker.session \
     import production_variant

variantB = production_variant(
         model_name=...,
     instance_type=...,
         initial_instance_count=1,
         variant_name='VariantB',
         initial_weight=50,
)
```

Using Production Variants for A/B Testing

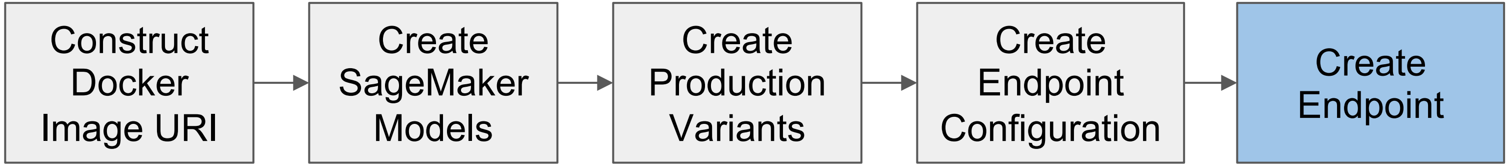
A/B Testing with PyTorch Bring-Your-Own Script




```
endpoint_config = sm.create_endpoint_config(
          EndpointConfigName=...,
     ProductionVariants=[variantA, variantB]
)
```

Using Production Variants for A/B Testing

A/B Testing with PyTorch Bring-Your-Own Script




```
endpoint_response = sm.create_endpoint(EndpointName=...,
                                       EndpointConfigName=...)
```

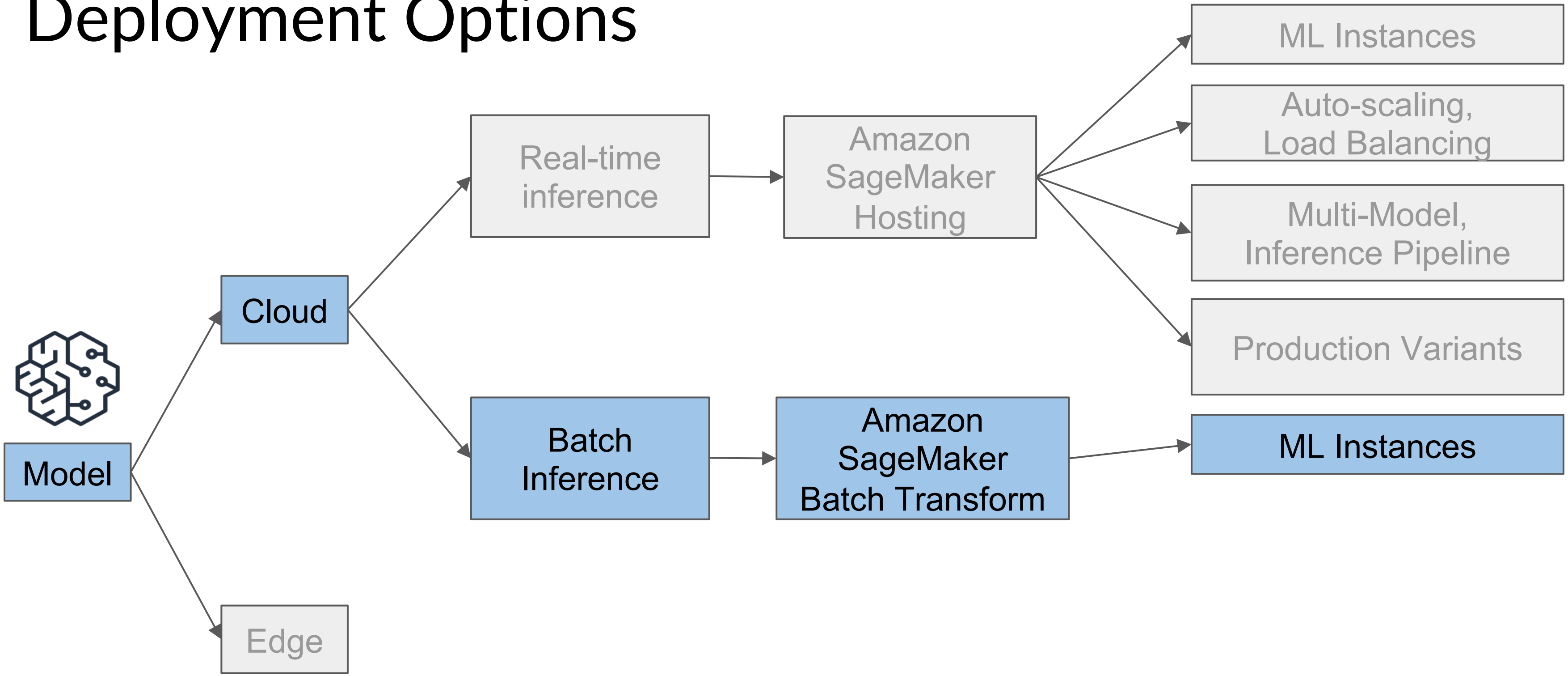
Amazon SageMaker Batch Transform

Batch Inference

Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

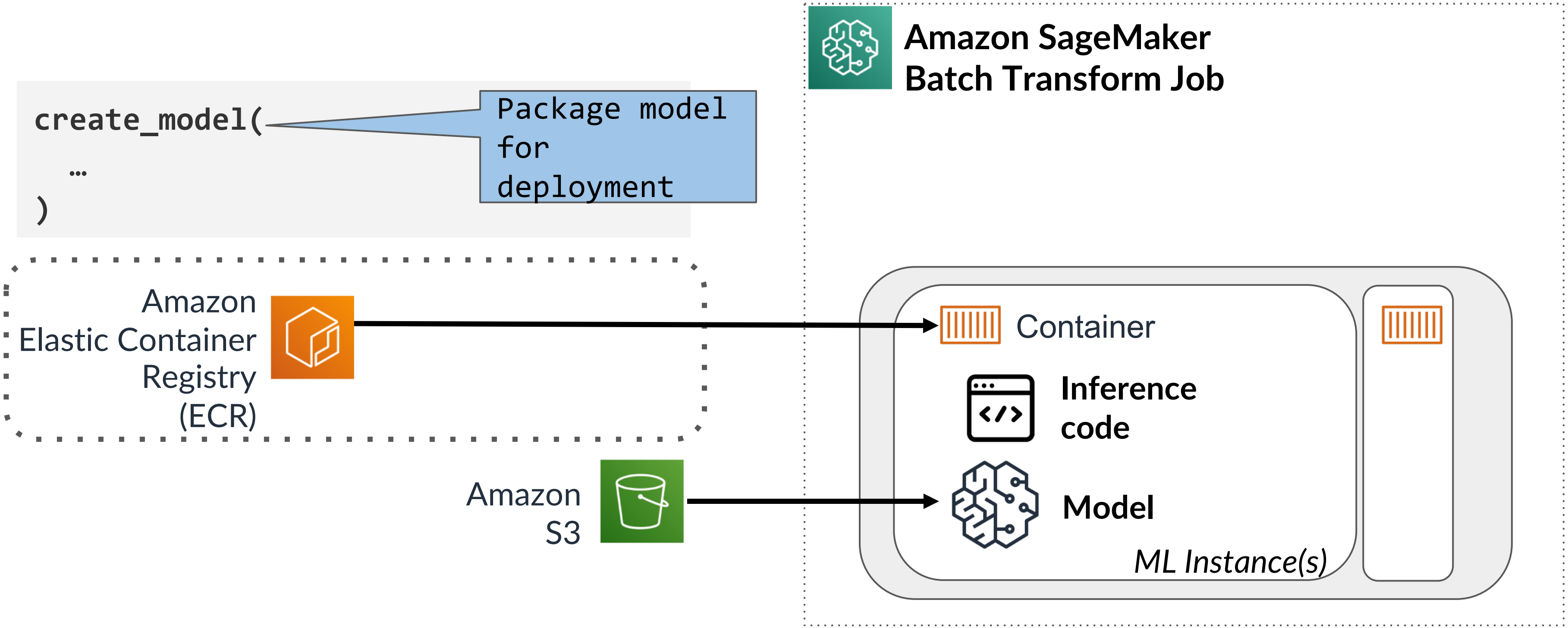
Deployment Options



Amazon SageMaker Batch Transform

Deploy Model For Batch Inference

```
create_model(
  …
)
```



Amazon SageMaker Batch Transform

Run Batch Transform Job For Batch Inference

```
sm_transformer = Transformer(
    ...
)
```

Configure Batch Transform Job

Amazon SageMaker Batch Transform Job

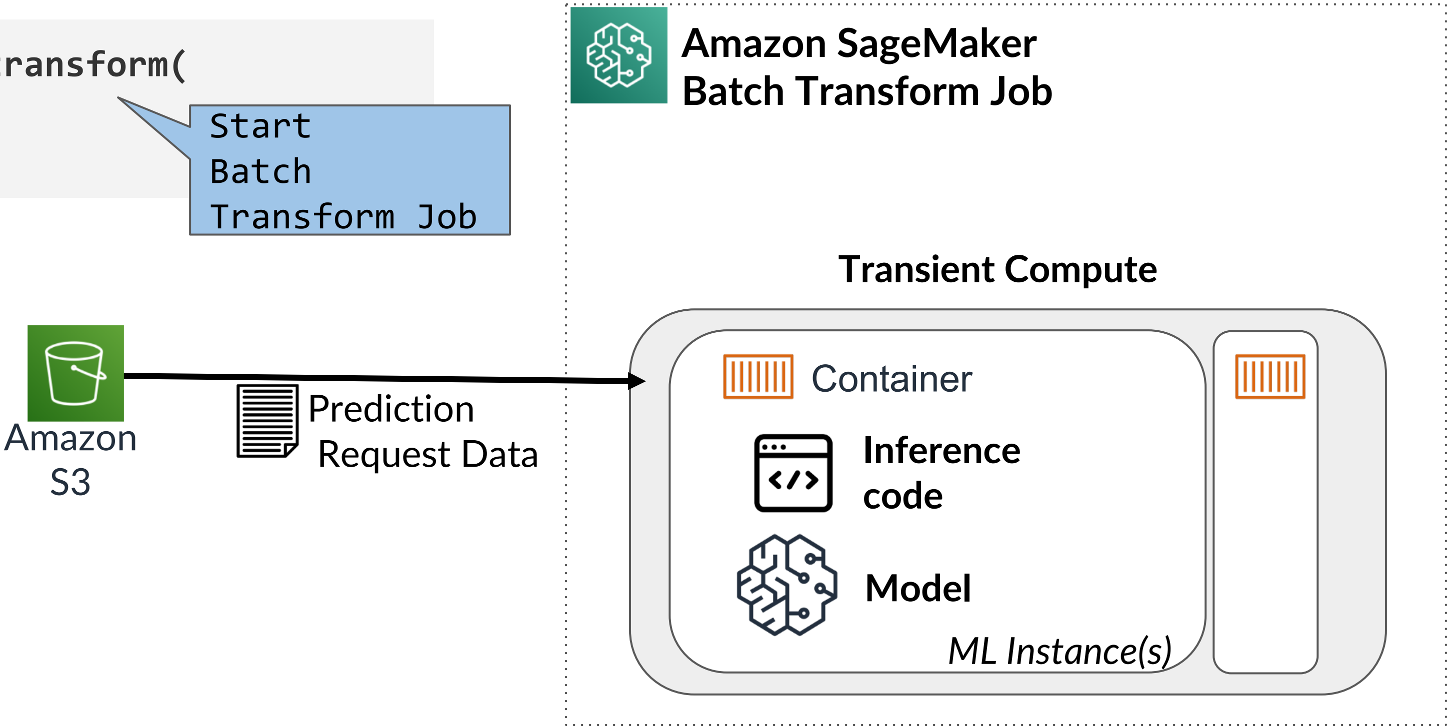
Identify Configuration →

- Instance type
- Instance count
- Model name
- S3 output path
- ...

Amazon SageMaker Batch Transform

Run Batch Transform Job For Batch Inference

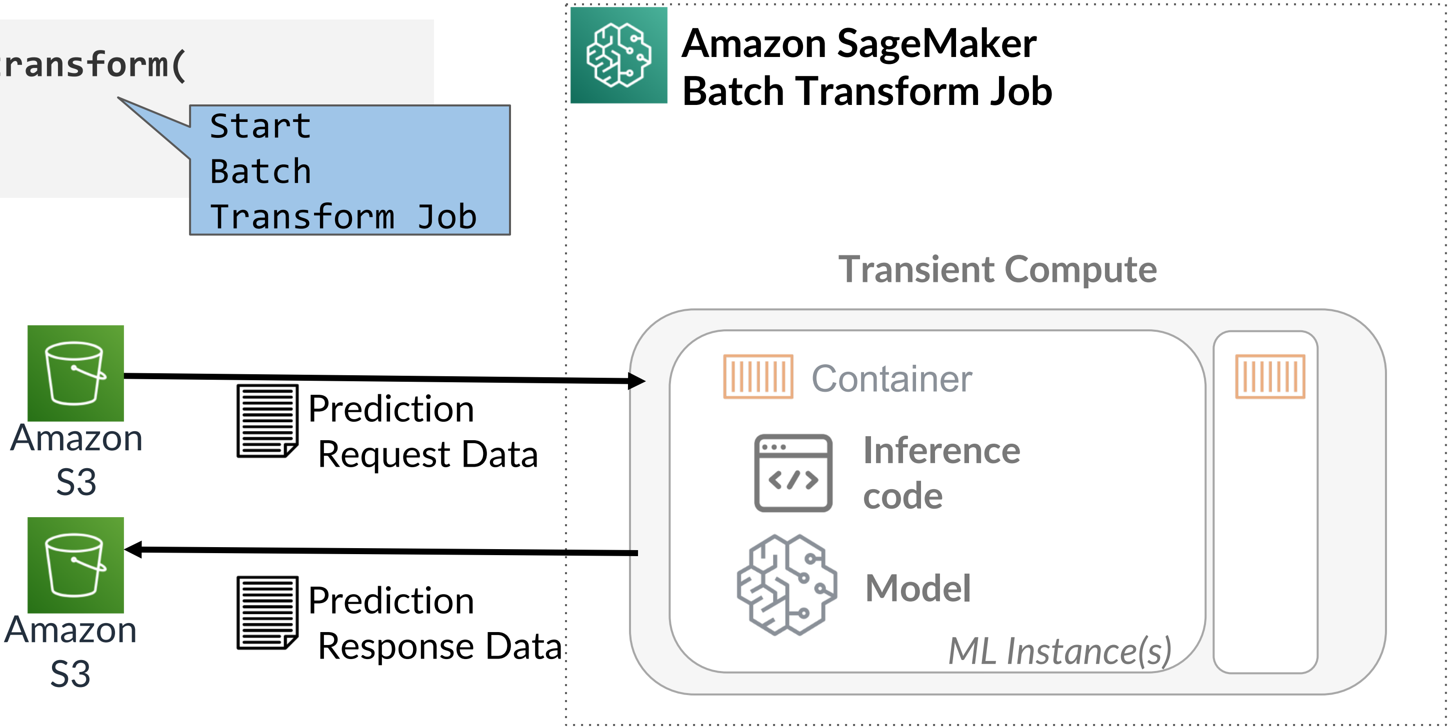
```
sm_transformer.transform(
    ...
)
```



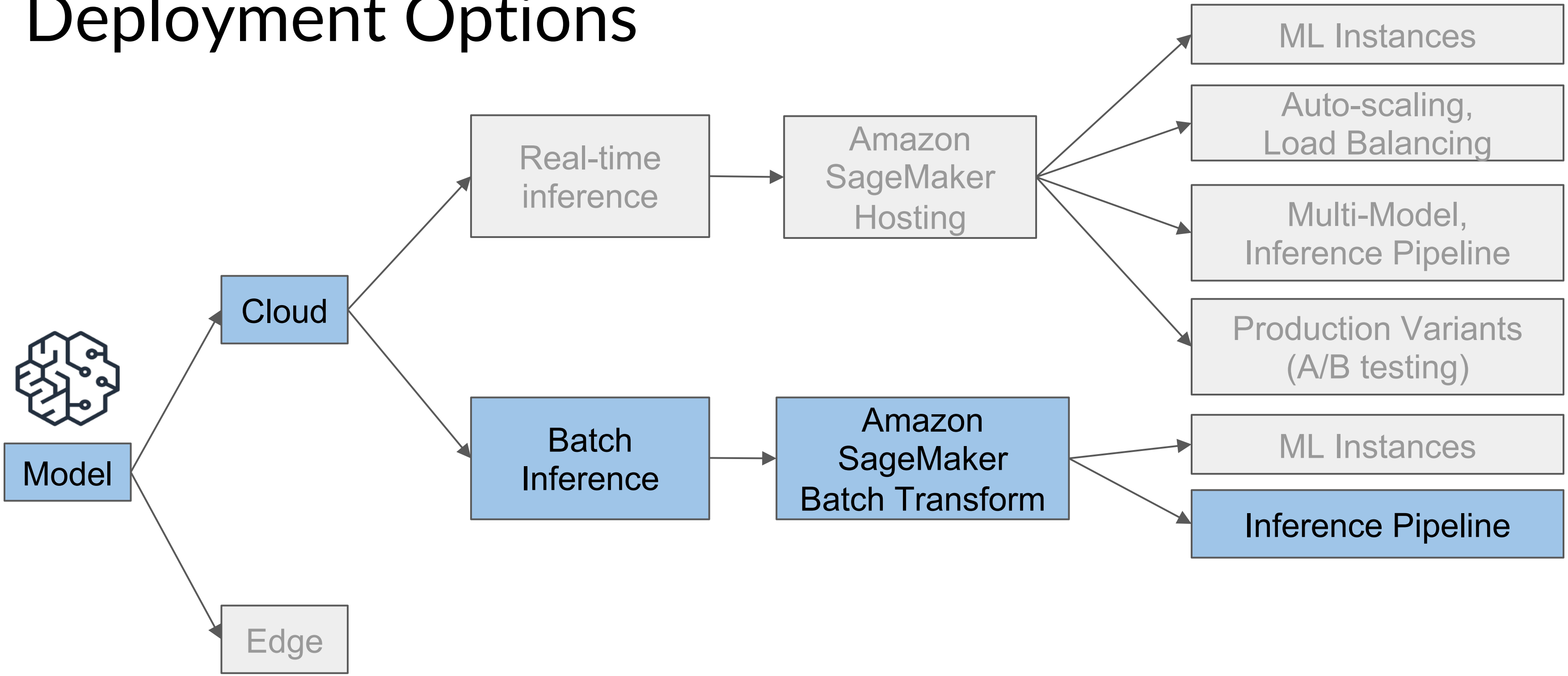
Amazon SageMaker Batch Transform

Run Batch Transform Job For Batch Inference

```
sm_transformer.transform(
    ...
)
```

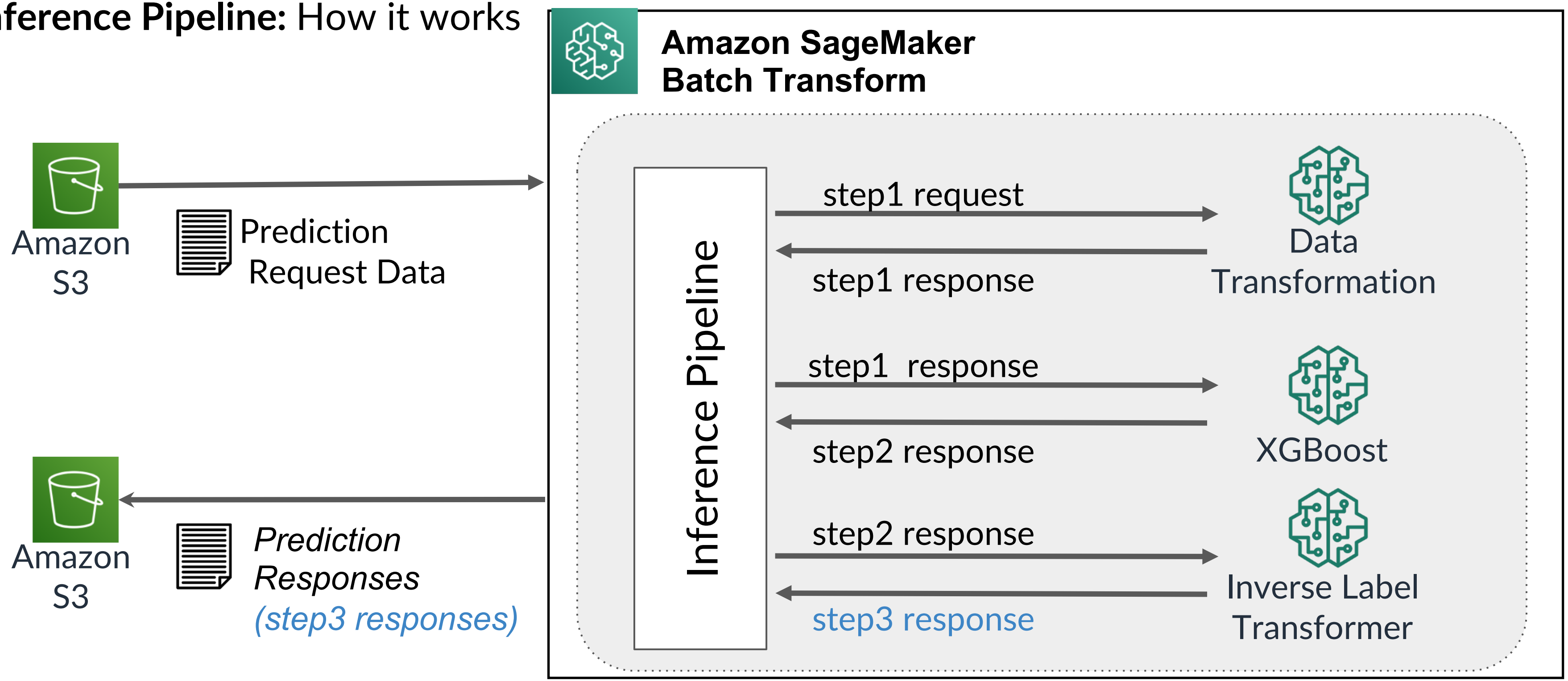


Deployment Options



Advanced Deployment Options

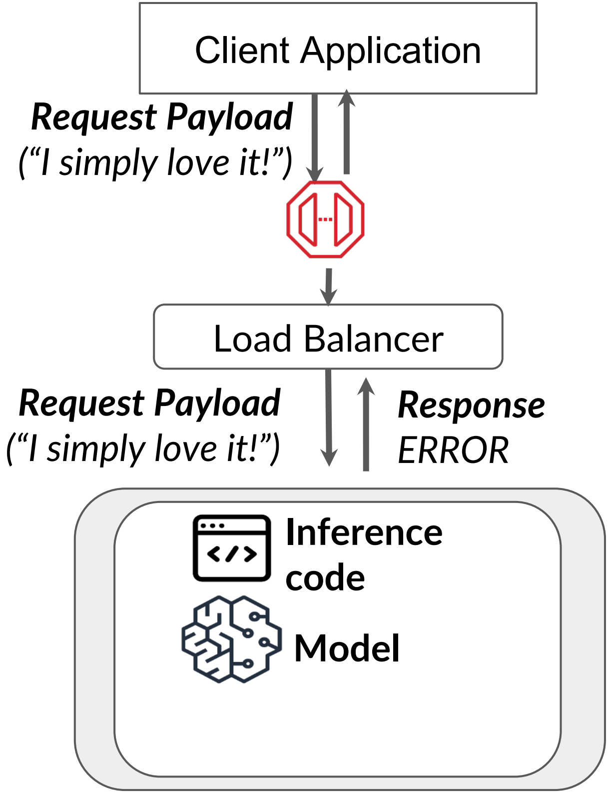
Inference Pipeline: How it works



Model Integration

Amazon SageMaker Model Integration

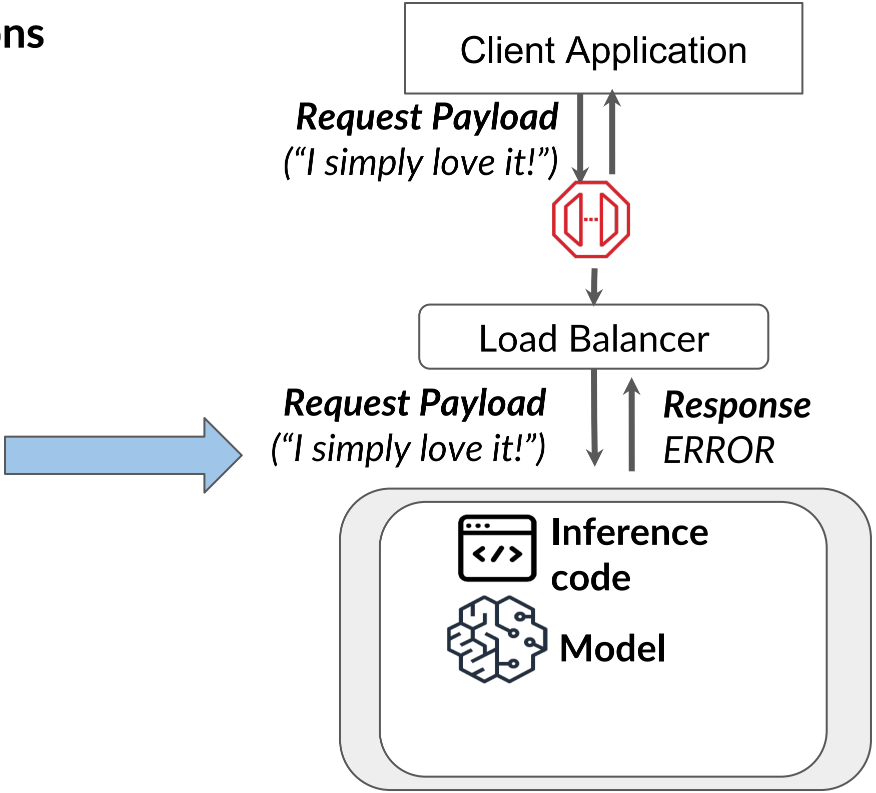
Integrating Models with ML Applications



Amazon SageMaker Model Integration

Integrating Models with ML Applications

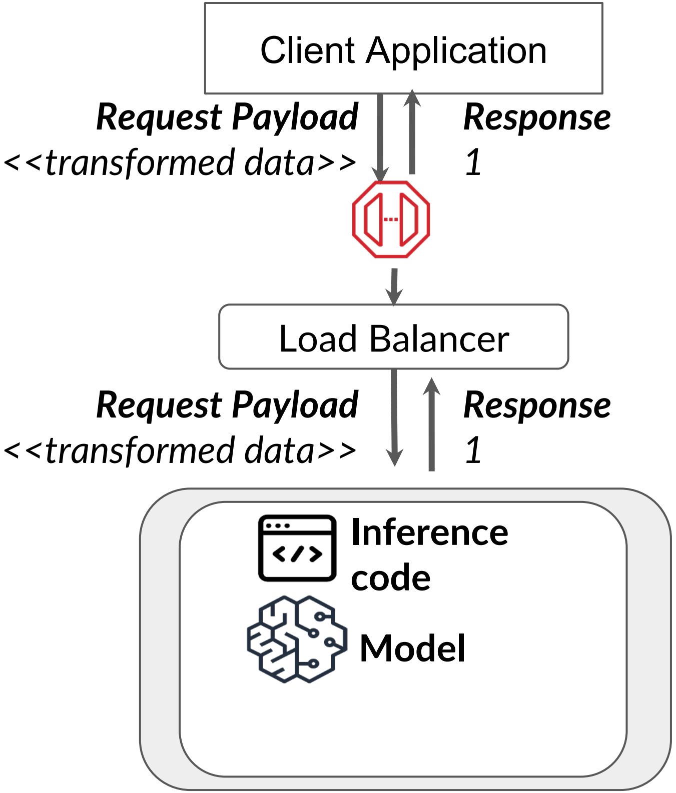
Need to apply the same data transformations used during training



Amazon SageMaker Model Integration

Prepare Data for Inference in Client Application

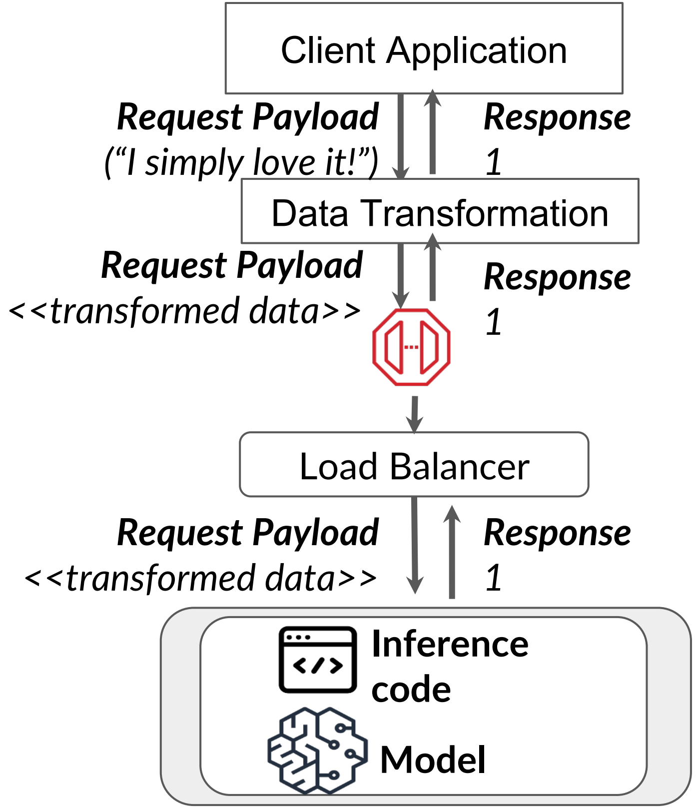
- **Implement data transformations in Client Application**
  - **Challenge:** Difficult to scale & manage
  - **Consideration:** Response may need to be transformed
    (1 = Positive)



Amazon SageMaker Model Integration

Prepare Data for Inference in Client Application

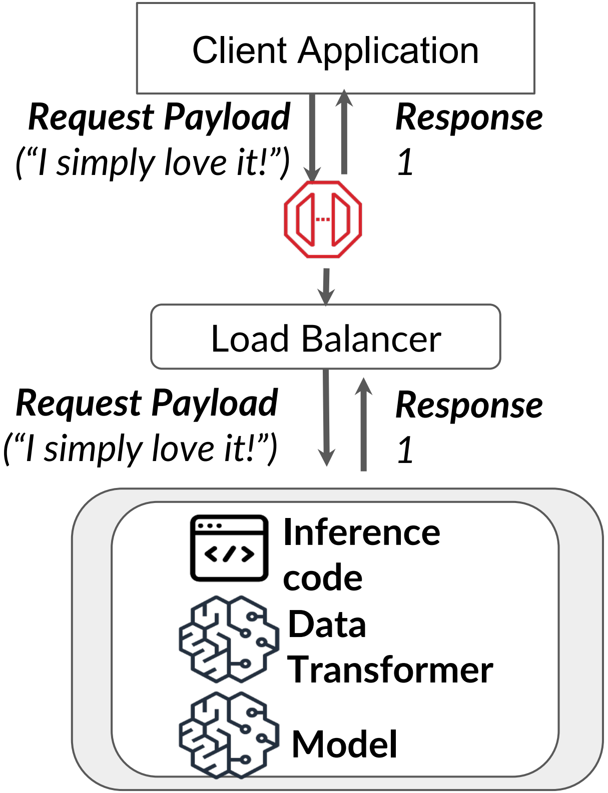
- **Implement transformation code before calling hosted model**
    - **Challenge:** Need to ensure transformation code stays in sync with training code



Amazon SageMaker Model Integration

Prepare Data as Part of an Inference Pipeline

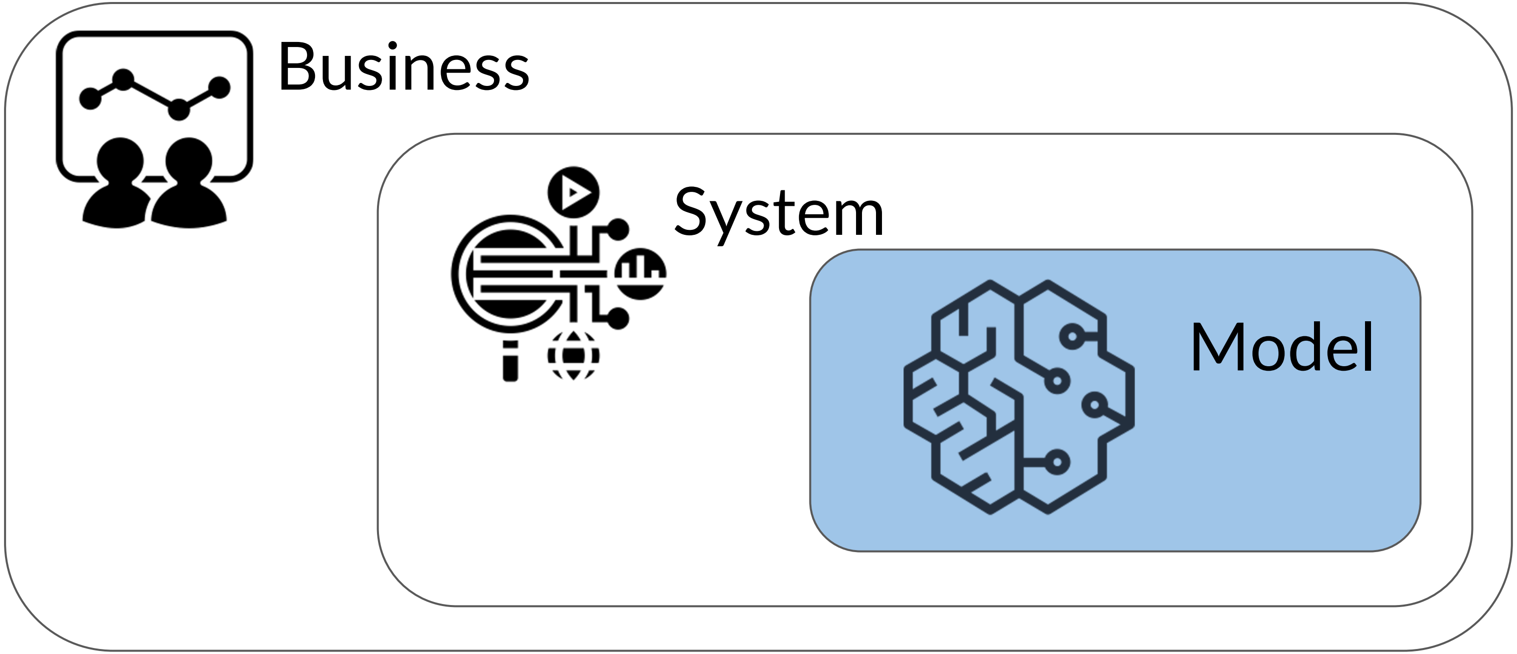
- **Implement data transformations in Inference Pipeline**
  - **Benefit:** Keep training & inference code in sync
  - **Consideration:** Additional Data Transformer for response may need to be transformed (1 = Positive)



Monitoring ML Workloads

Monitoring Machine Learning Workloads

Considerations



Monitoring Machine Learning Workloads

Model Monitoring

Why? Models degrade over time

Monitoring Machine Learning Workloads

Model Monitoring

Why? Models degrade over time

Examples

Customer behavior change
Ex. Product change, Demand change

Monitoring Machine Learning Workloads

Model Monitoring

Why? Models degrade over time

Examples

Customer behavior change
Ex. Product change, Demand change

Changing business environment
Ex. New products

Monitoring Machine Learning Workloads

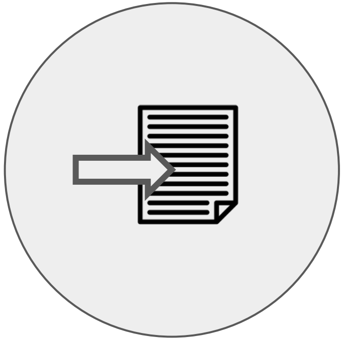
Model Monitoring

Why? Models degrade over time

Examples

Customer behavior change
Ex. Product change, Demand change

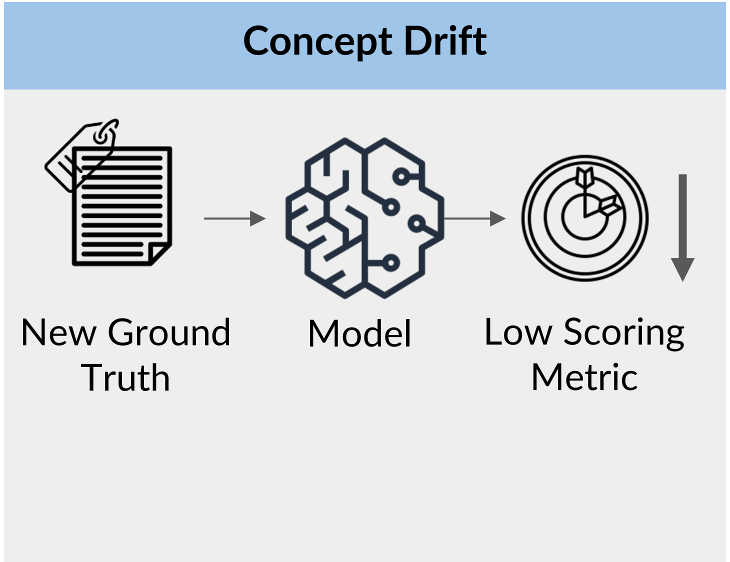
Changing business environment
Ex. New products

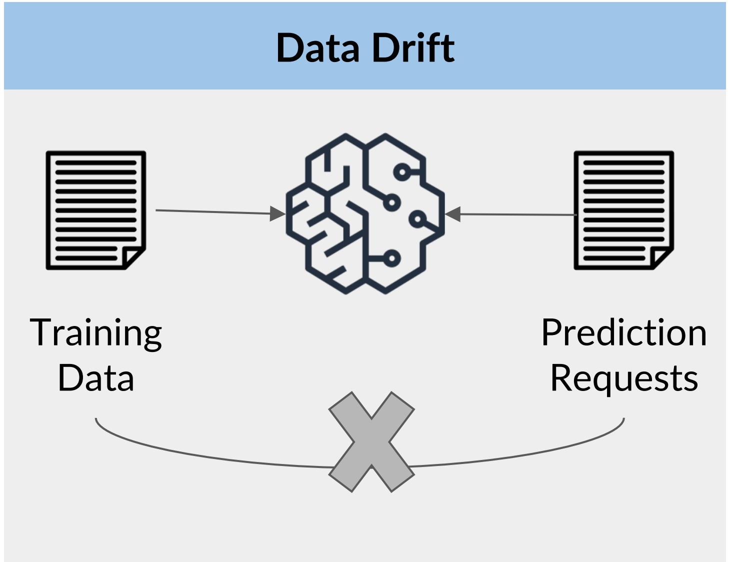


Changing data pipeline
Ex. Feature data suddenly missing

Monitoring Machine Learning Workloads

Model Monitoring





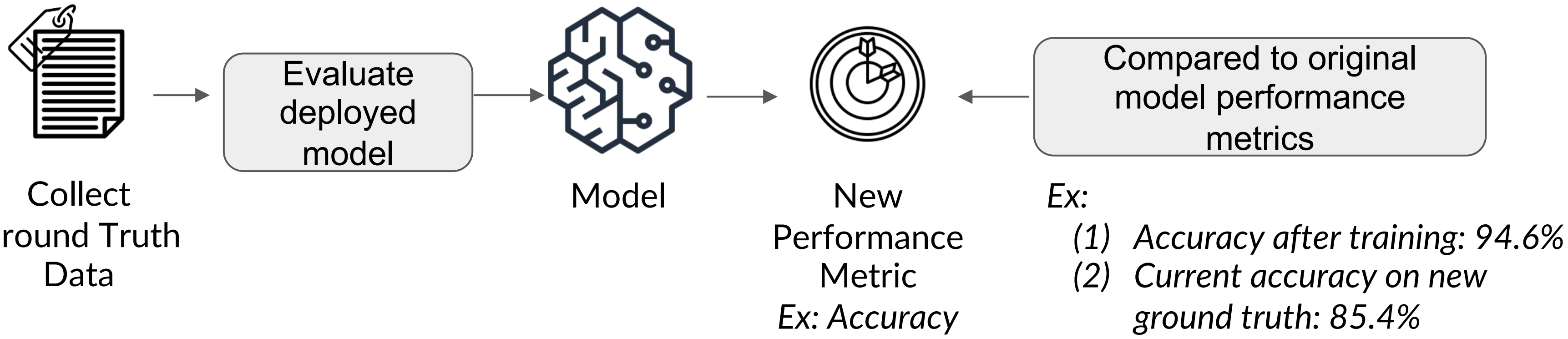
Monitoring Machine Learning Workloads

Concept Drift

What causes **concept drift?**

- Environment changes that impact the context of the predicted target

Methods to detect:



Monitoring Machine Learning Workloads

Data Drift

What causes **Data Drift?**

- Changes in the model input data

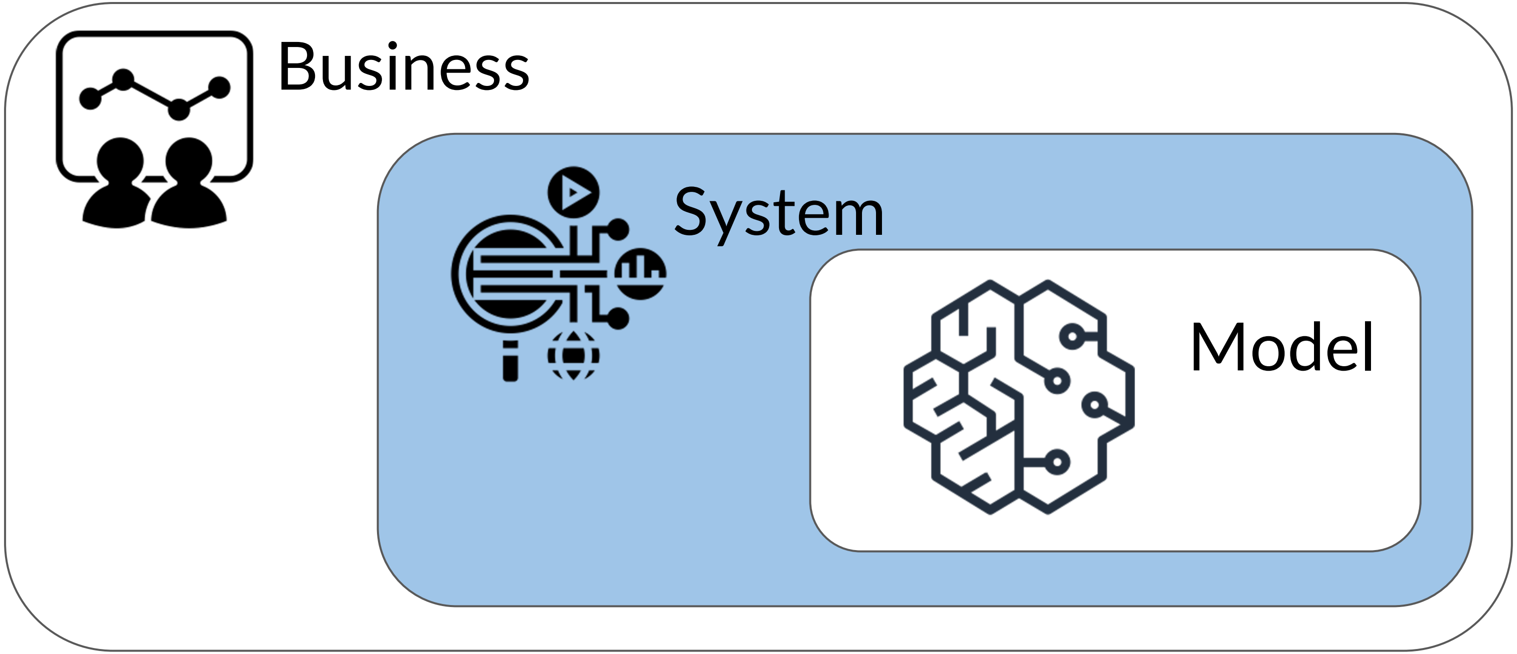
Methods to detect:

Example: Deequ - Open Source Library

- **Data Profiling:** Gather statistics about each feature used to train the model
- **Establish Constraints:** Boundaries on normal/expected data
- **Detect Data Anomalies:** Understand when prediction data violates constraints

Monitoring Machine Learning Workloads

Considerations

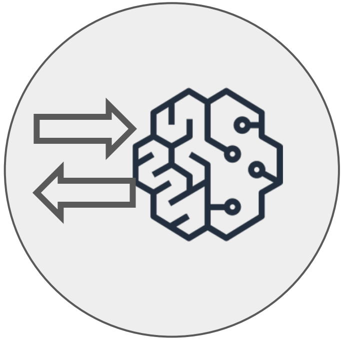


Monitoring Machine Learning Workloads

System Monitoring

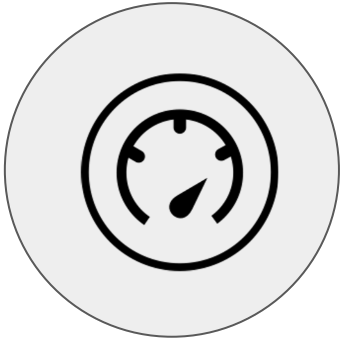
Why? Ensure your model and supporting resources are functioning as expected

Examples



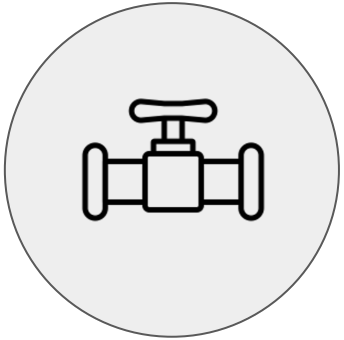
Model Latency

model latency is the time it takes for
a model to respond to a prediction request.



System Metrics

things like CPU utilization

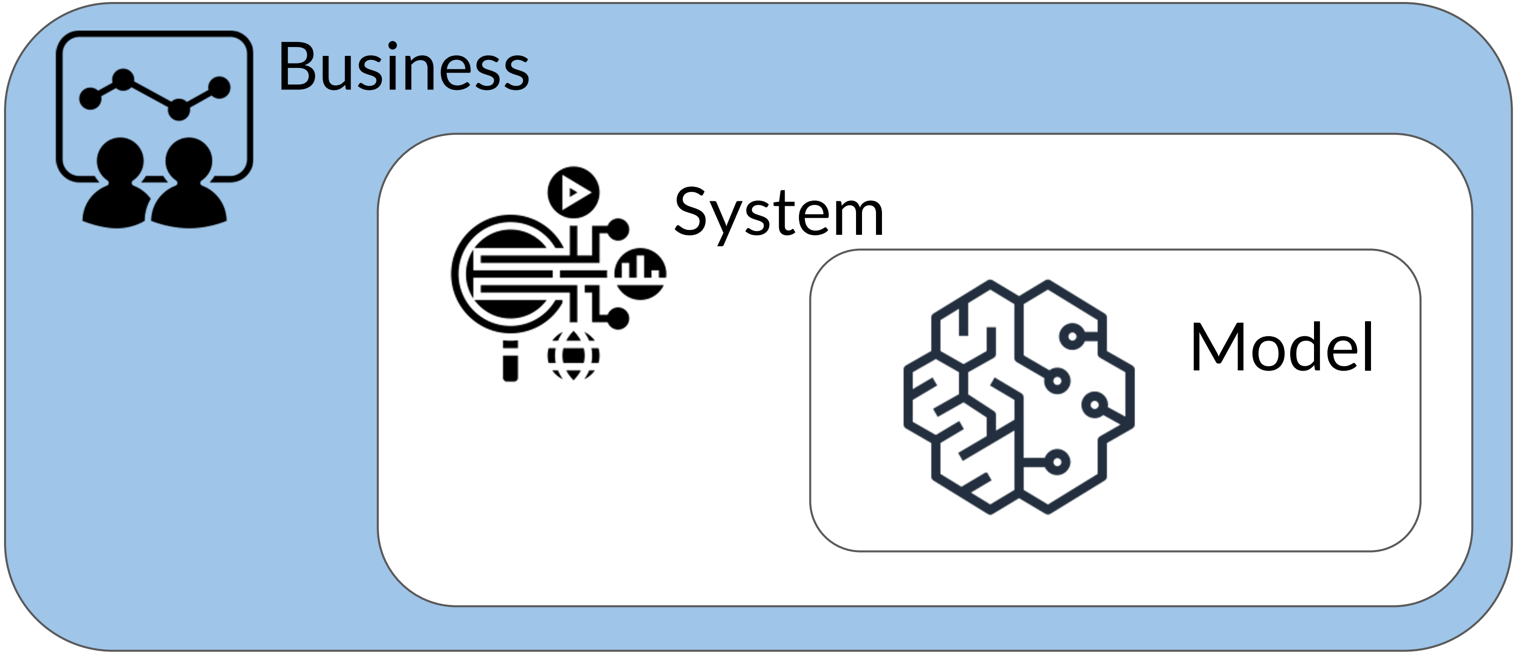


ML Pipeline

monitoring your machine learning pipelines so that you know, if there are any potential issues with model retraining or deploying a new model version.

Monitoring Machine Learning Workloads

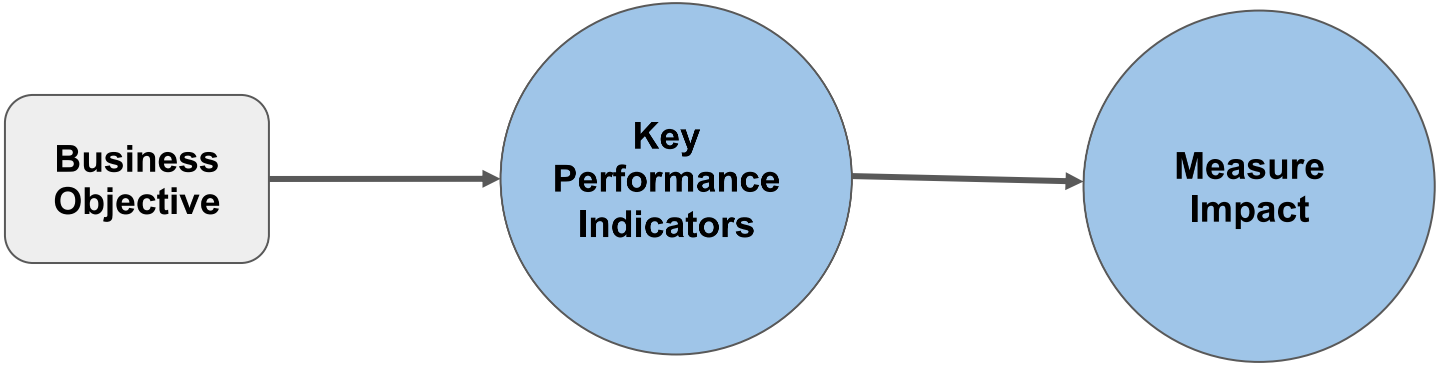
Considerations



Monitoring Machine Learning Workloads

Monitoring Impact on Business Objectives

Why? Ensure your model has impact on the business objective



Model Monitoring

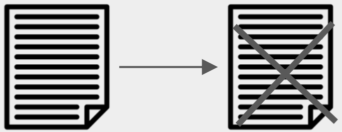
Using Amazon SageMaker Model Monitor

Amazon SageMaker Model Monitor

Monitor Types

Data Drift

Data Quality



Monitor drift in data quality

Concept Drift

Model Quality

Monitor drift in model quality metrics

Concept Drift

Statistical Bias Drift

Monitor statistical bias drift in model predictions

Data Drift

Feature Attribution Drift

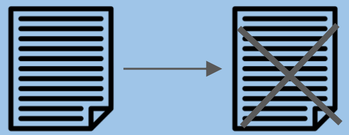
Monitor drift in feature attribution

Amazon SageMaker Model Monitor

Monitor Types

Data Drift

Data Quality



Monitor drift in data quality

Concept Drift

Model Quality

Monitor drift in model quality metrics

Concept Drift

Statistical Bias Drift

Monitor statistical bias drift in model predictions

Data Drift

Feature Attribution Drift

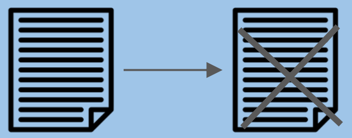
Monitor drift in feature attribution

Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor

Data Drift

Data Quality



Monitor drift in data quality

- Monitor when inference data drifts away from baseline (training) data
- Model Monitor uses, **Deequ**, an open source library built on Apache Spark

Amazon SageMaker Model Monitor uses the open source AWS Deequ library to monitor when inference input data drifts away from the baseline training input data.

Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor

Deploy Endpoint → Enable Data Capture

Inference requests, prediction results

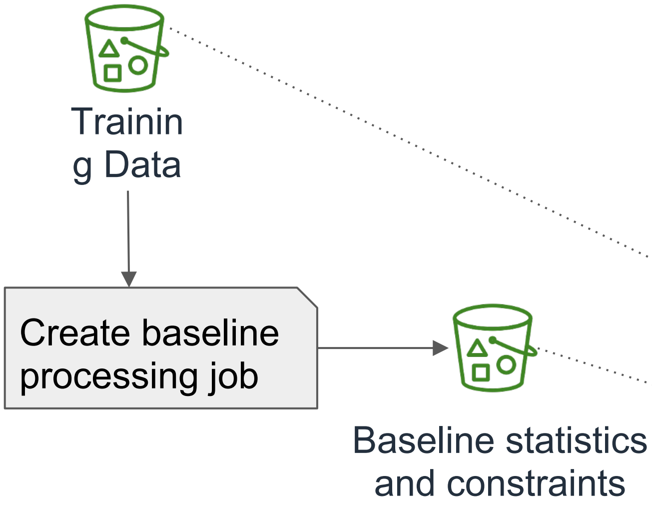
Amazon SageMaker Endpoint

% of data to capture

```
data_capture_config=DataCaptureConfig(
                        enable_capture = True,
                        sampling_percentage=100,

destination_s3_uri=s3_capture_upload_path)

predictor = model.deploy(initial_instance_count=1,
                instance_type='ml.m4.xlarge',
                endpoint_name=endpoint_name
                data_capture_config=data_capture_config)
```

Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor




```
data_monitor = DefaultModelMonitor(
    #baseline processing job configuration - data
    ...
)

data_monitor.suggest_baseline(
    baseline_dataset=s3train_data_uri,
    dataset_format=DatasetFormat.csv(header=True),
    output_s3_uri=baseline_results_uri,
    wait=True
)
```

Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor

Deploy Endpoint → Enable Data Capture → Create Baseline

Baseline statistics and constraints

statistics.json

- Columnar statistics for each feature
- Examples:
  - Numeric → missing values, mean, min, max, distribution
  - String → missing values, distinct values, categorical distribution

Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor

Deploy Endpoint → Enable Data Capture → Create Baseline

Baseline statistics and constraints

statistics.json

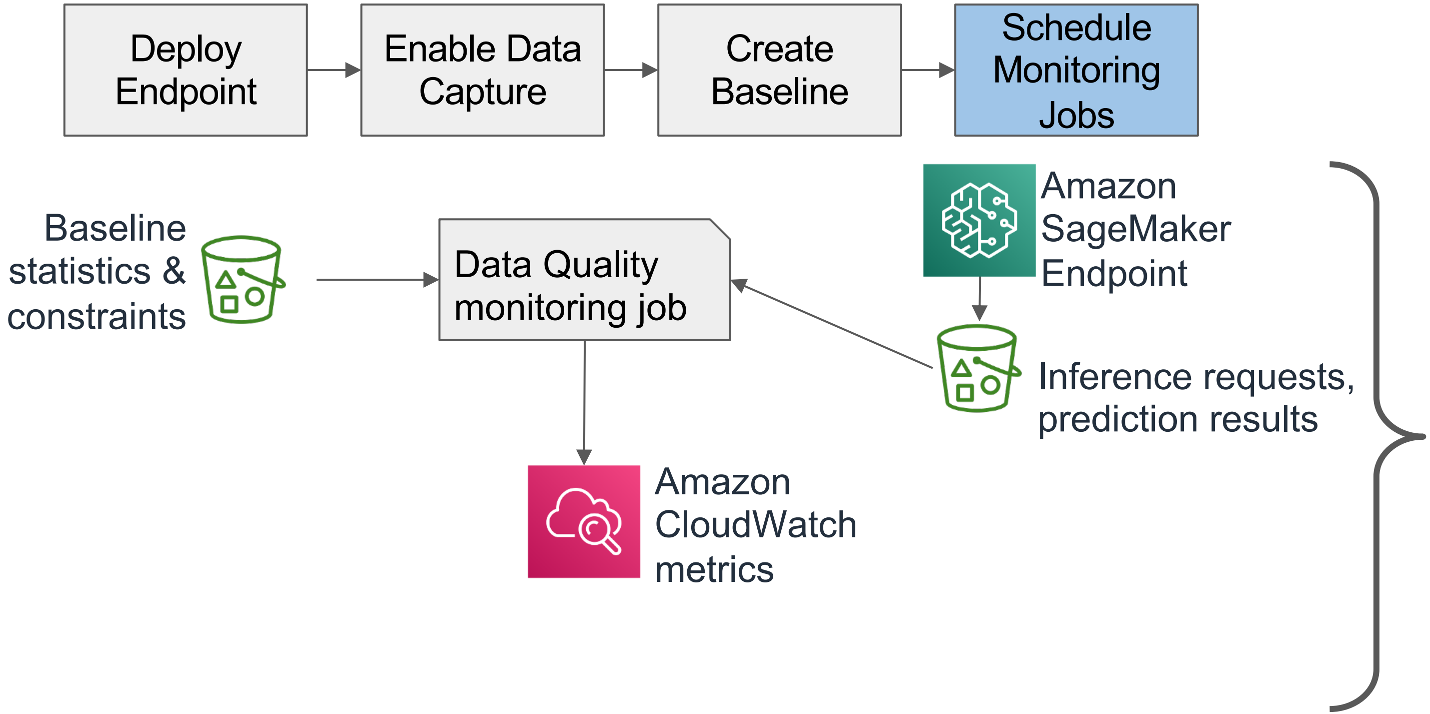
- Columnar statistics for each feature
- Examples:
  - Numeric → missing values, mean, min, max, distribution
  - String → missing values, distinct values, categorical distribution

constraints.json

- Constraints that are used to evaluate potential data drift
- Examples:
  - Numeric → non-negative
  - String → observed values

Amazon SageMaker Model Monitor

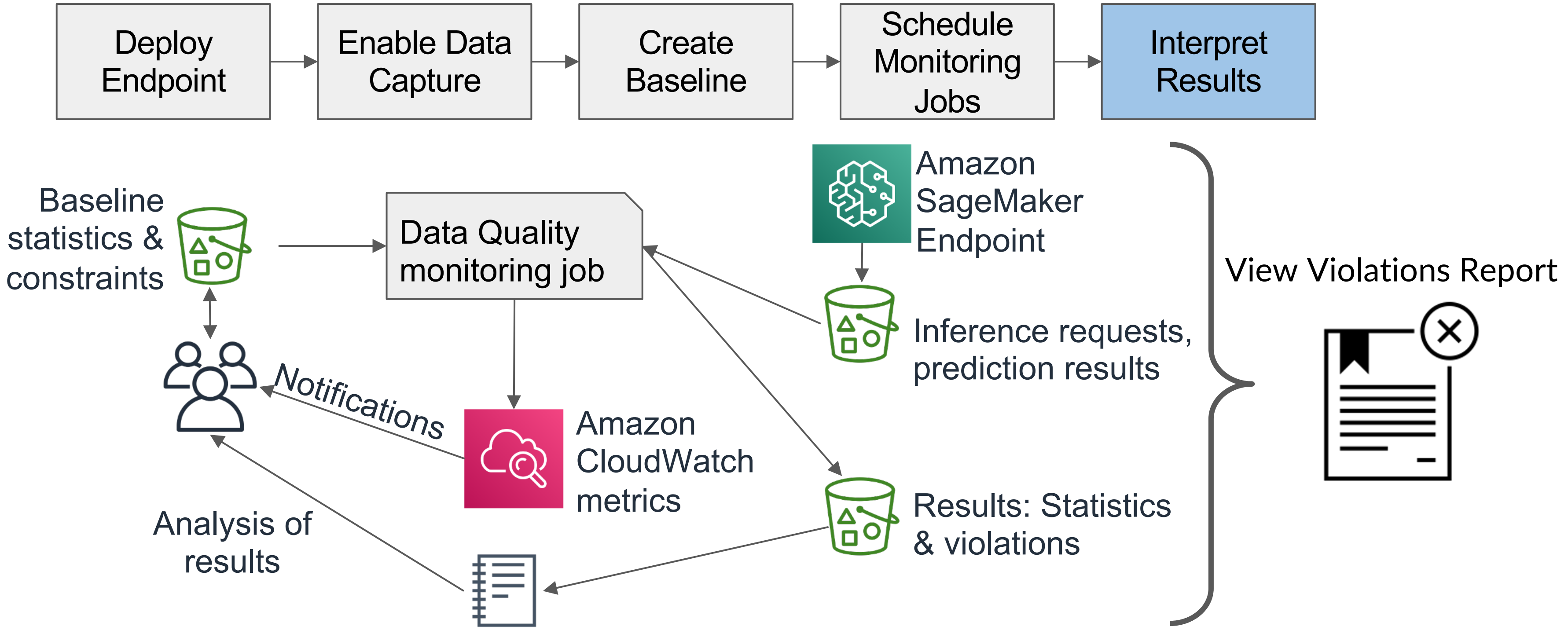
Monitor Type: Data Quality Monitor



Continuously monitor the data collected on a schedule

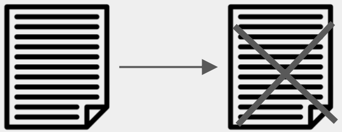
Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor



Amazon SageMaker Model Monitor

Monitor Types

Data Drift	*Concept Drift*	*Concept Drift*	*Data Drift*
Data Quality	**Model Quality**	**Statistical Bias Drift**	**Feature Attribution Drift**
			
Monitor drift in data quality	Monitor drift in model quality metrics	Monitor statistical bias drift in model predictions	Monitor drift in feature attribution

Amazon SageMaker Model Monitor

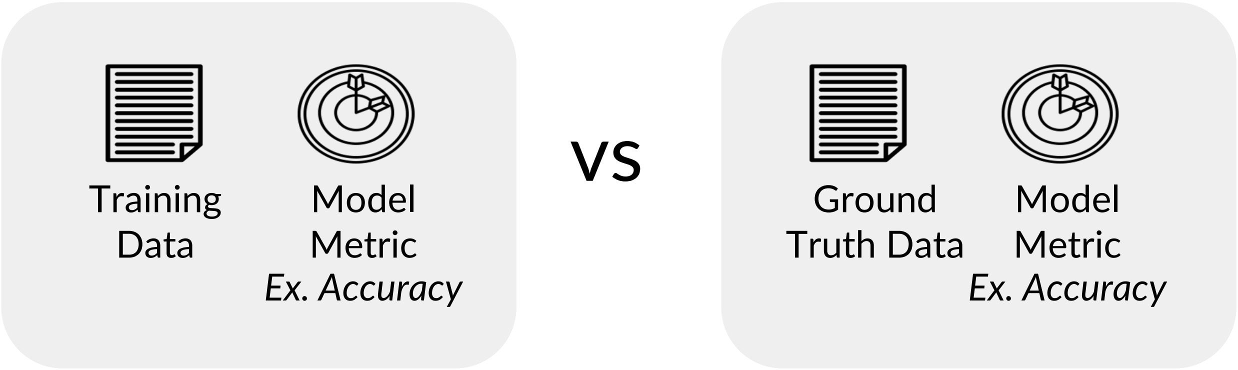
Monitor Type: Model Quality Monitor

Concept Drift

Model Quality

Monitor drift in model quality metrics

- Monitor model quality by comparing model predictions with ground truth labels

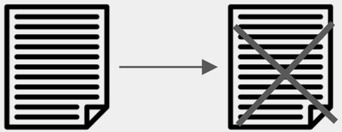


Amazon SageMaker Model Monitor

Monitor Types

Data Drift

Data Quality



Monitor drift in data quality

Concept Drift

Model Quality

Monitor drift in model quality metrics

Concept Drift

Statistical Bias Drift

Monitor statistical bias drift in model predictions

Data Drift

Feature Attribution Drift

Monitor drift in feature attribution

Amazon SageMaker Model Monitor

Monitor Type: Statistical Bias Drift

Concept Drift

Statistical Bias Drift

Monitor statistical bias drift in model predictions

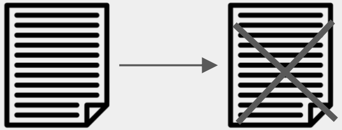
- Monitor predictions for statistical bias
- Amazon SageMaker Clarify integrates with Amazon SageMaker Model Monitor to detect statistical bias drift

Amazon SageMaker Model Monitor

Monitor Types

Data Drift

Data Quality



Monitor drift in data quality

Concept Drift

Model Quality

Monitor drift in model quality metrics

Concept Drift

Statistical Bias Drift

Monitor statistical bias drift in model predictions

Data Drift

Feature Attribution Drift

Monitor drift in feature attribution

Amazon SageMaker Model Monitor

Monitor Type: Feature Attribution Drift

Data Drift

Feature Attribution Drift

Monitor drift in feature attribution

- Monitor features contributing to predictions over time
- Amazon SageMaker Clarify integrates with Amazon SageMaker Model Monitor to detect feature attribution drift
- Utilizes SHAP for baselining

SHAP or shapely additive explanations is a common technique used to explain the output of a machine learning model.