

Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

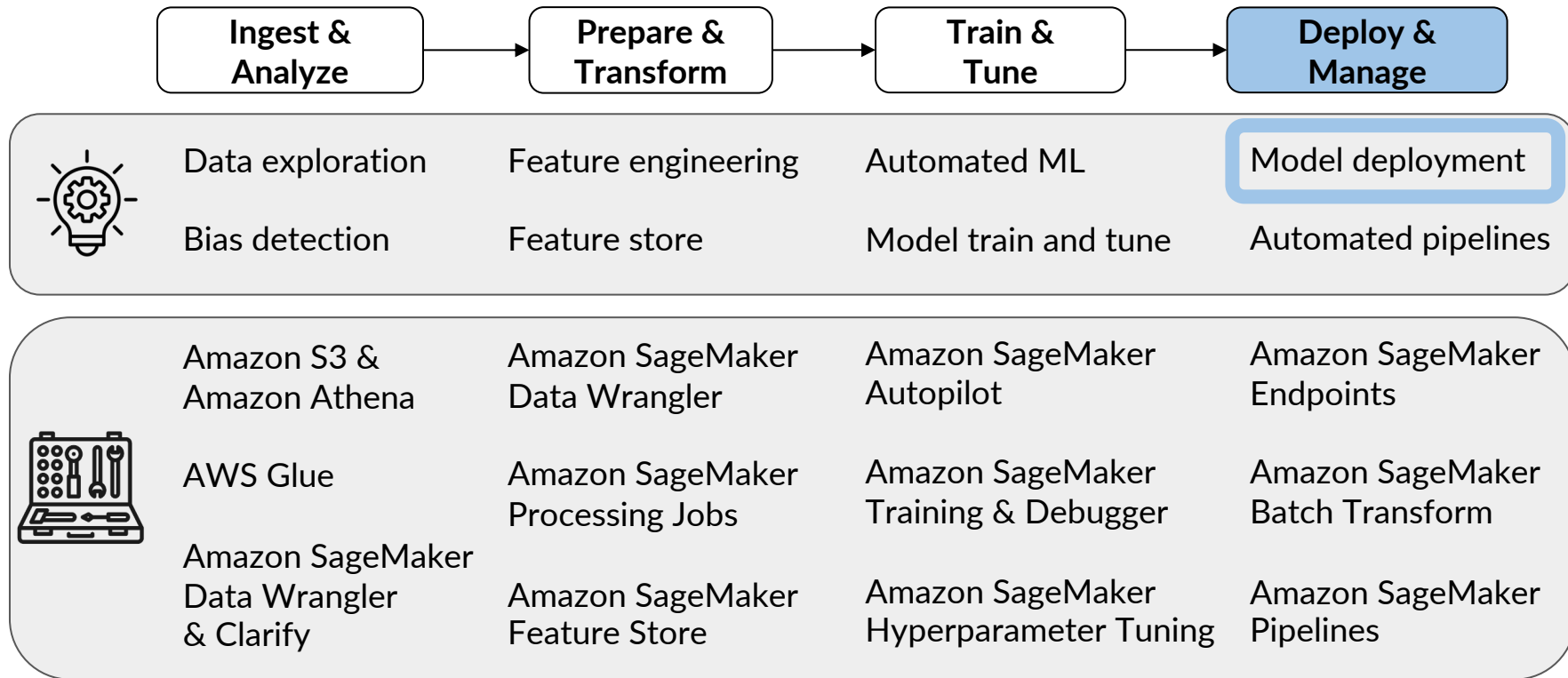
For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>

Model Deployment

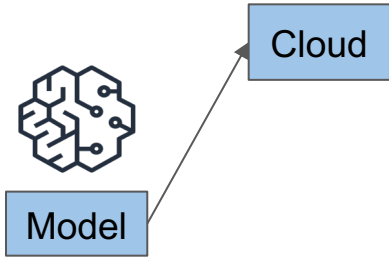
Overview



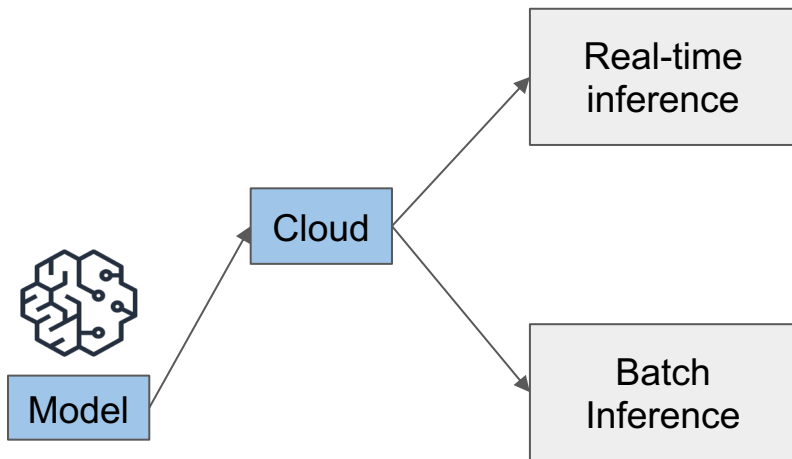
Machine Learning Workflow



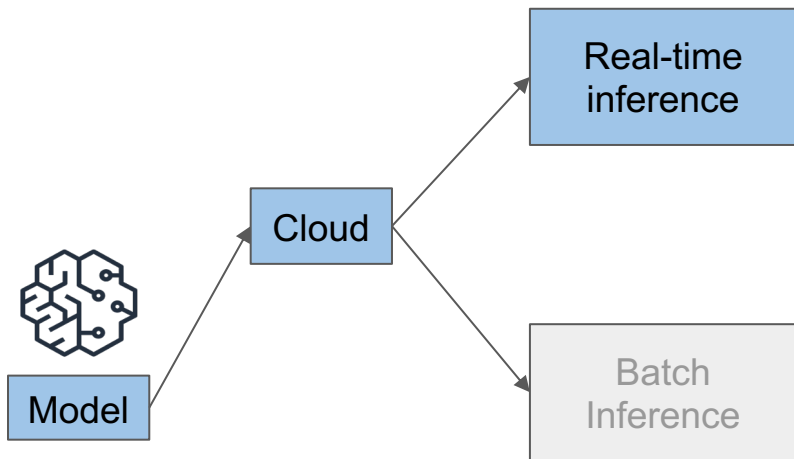
Deployment Options



Deployment Options



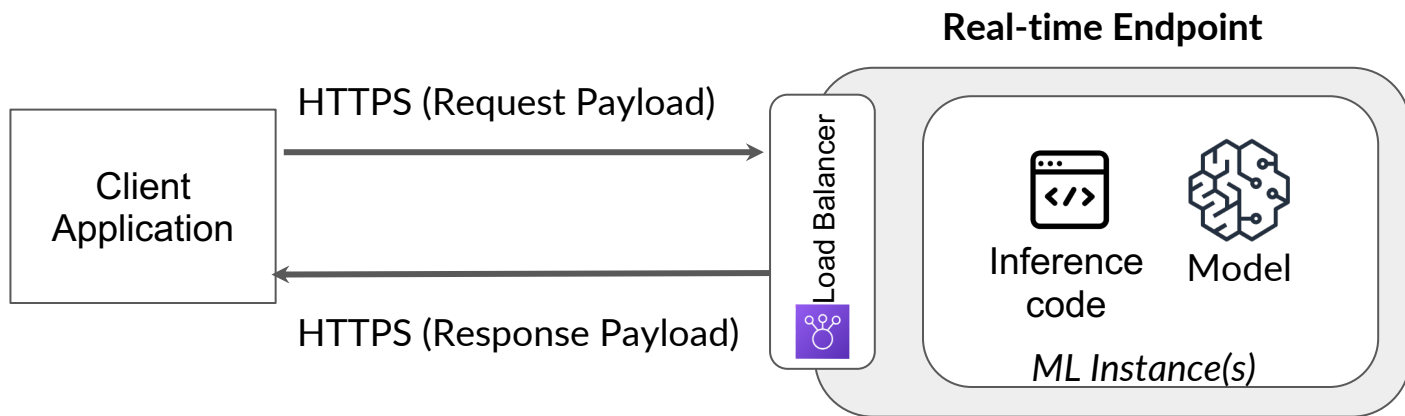
Deployment Options



Deployment Options

Real-Time Inference

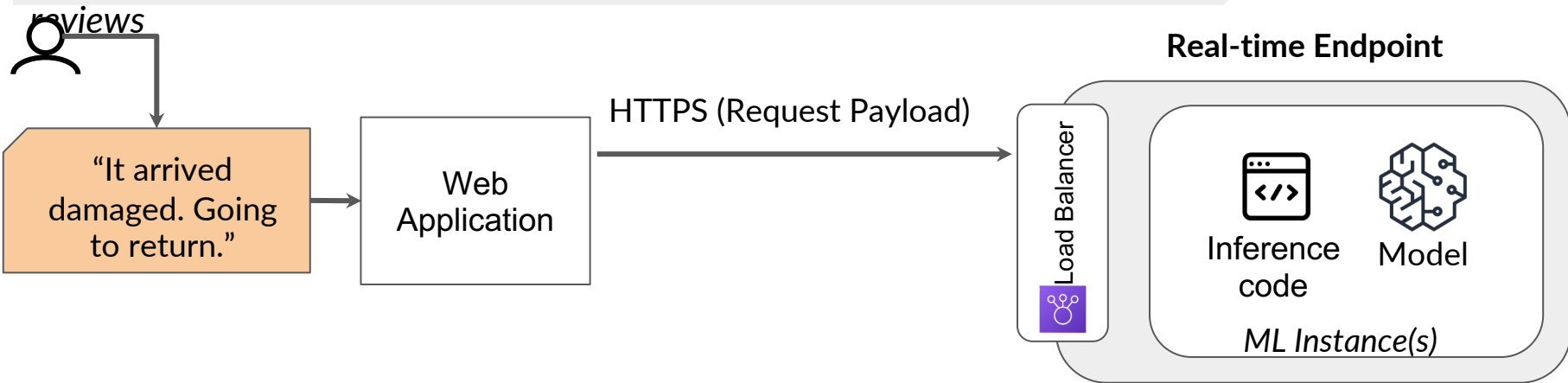
But let's start with deploying a model for real time inference in the cloud deploying a model for real time Inference. This means deploying it to a persistent hosted environment that's able to serve requests for prediction and provide prediction responses back in real time or near real time. This involves 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.



Deployment Options

Real-Time Inference - Product Review Example

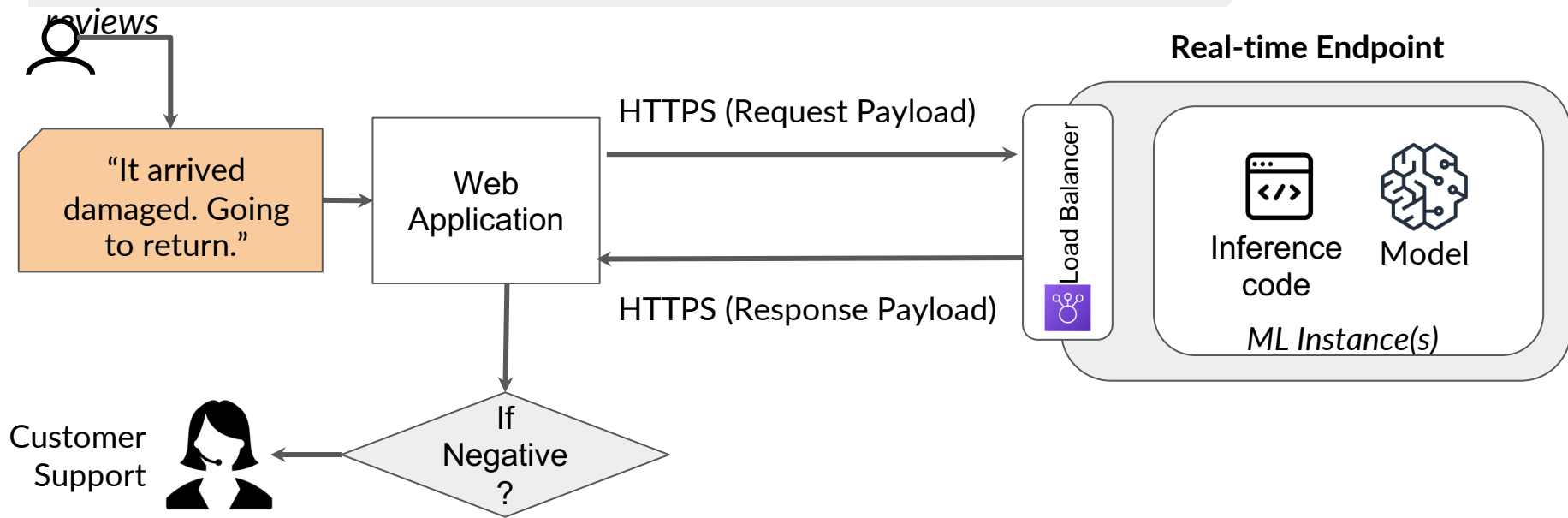
Goal: Improve customer experience with quick responses to negative



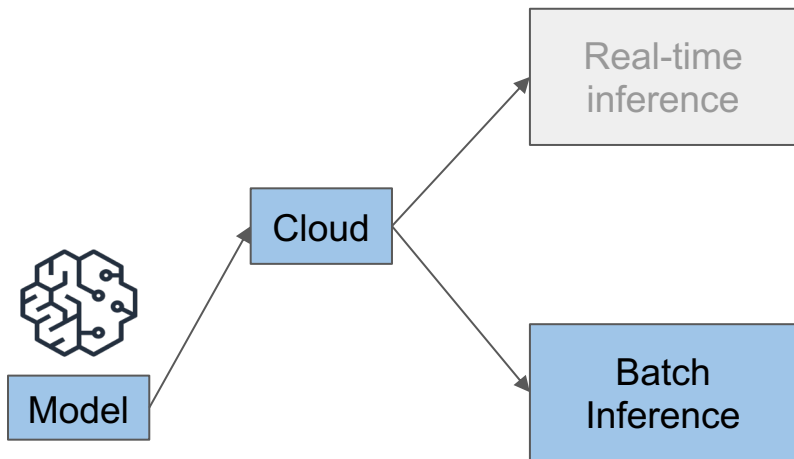
Deployment Options

Real-Time Inference - Product Review Example

Goal: Improve customer experience with quick responses to negative reviews



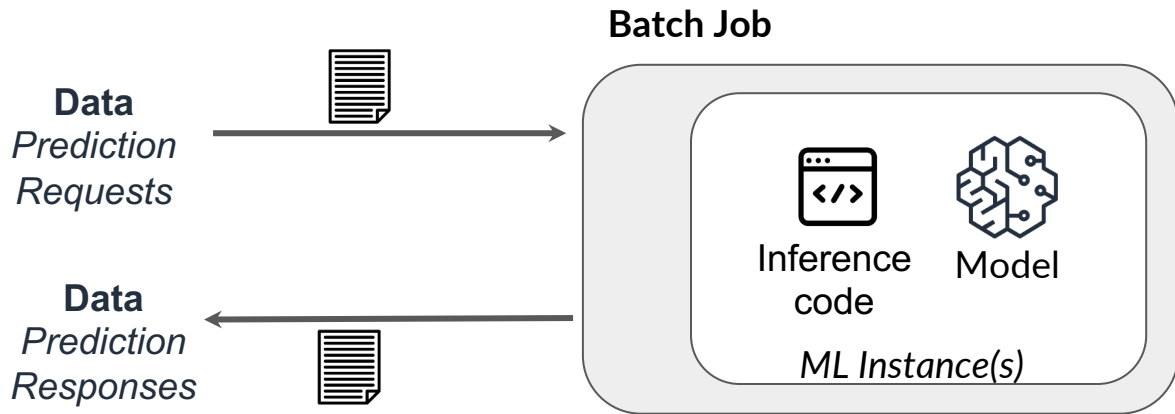
Deployment Options



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 outputting 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

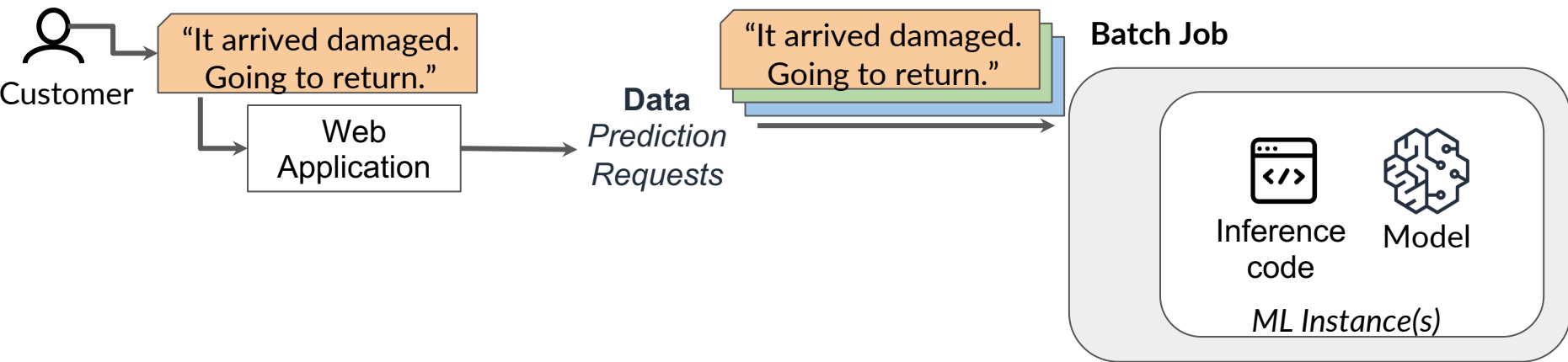
Batch Inference



Deployment Options

Batch Inference - Product Review

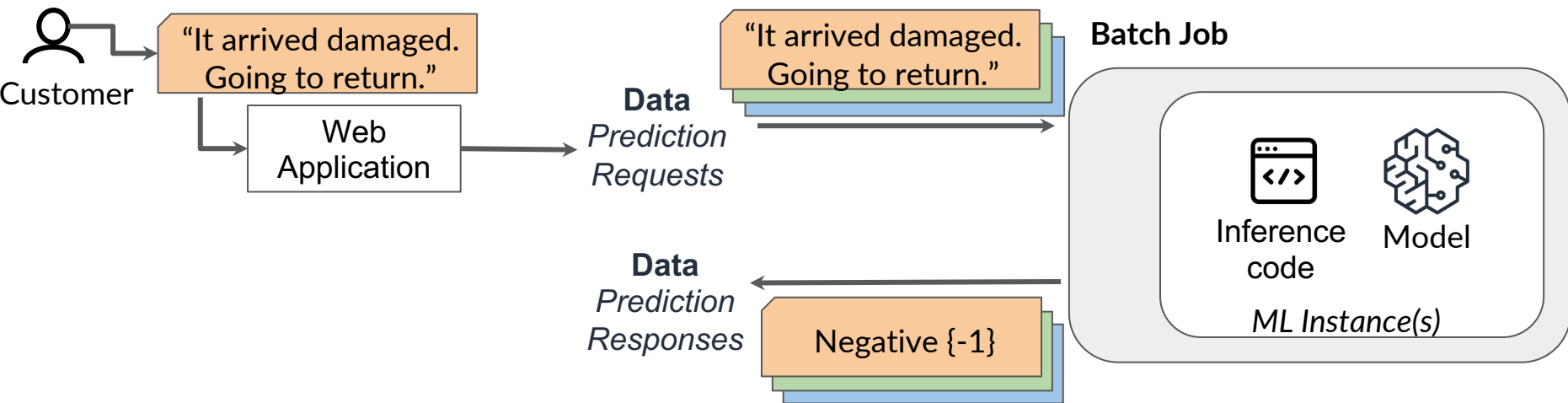
Goal: Identify vendors with potential quality issues



Deployment Options

Batch Inference - Product Review

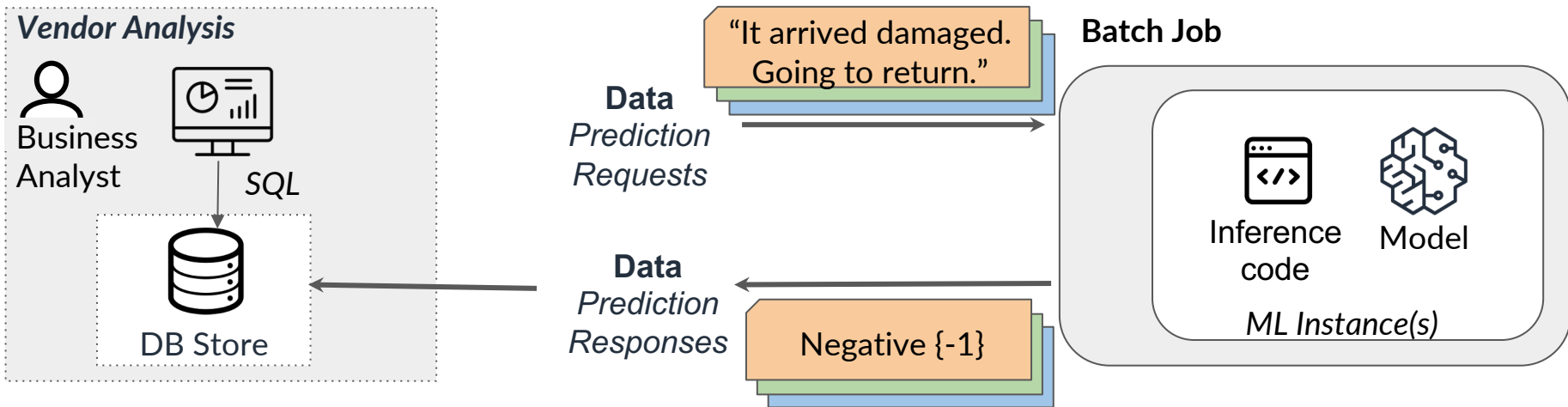
Goal: Identify vendors with potential quality issues



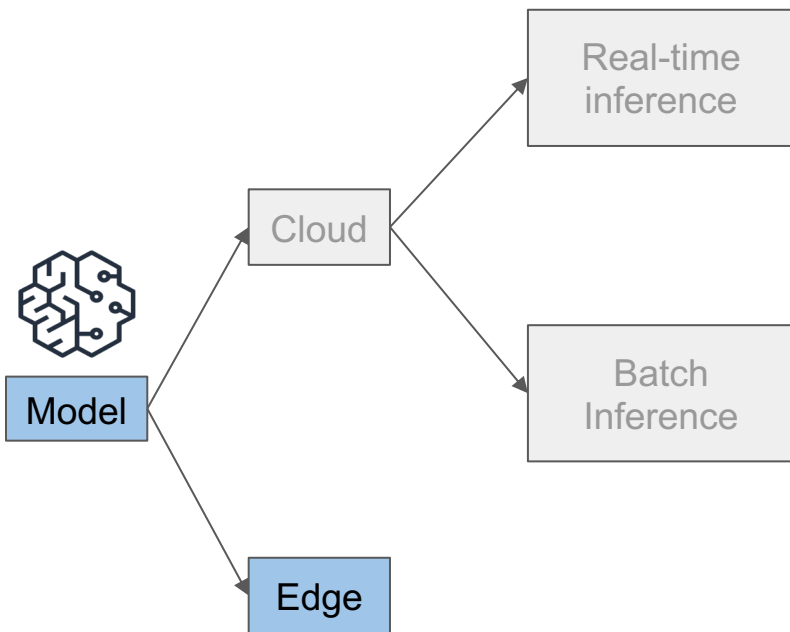
Deployment Options

Batch Inference - Product Review

Goal: Identify vendors with potential quality issues



Deployment Options

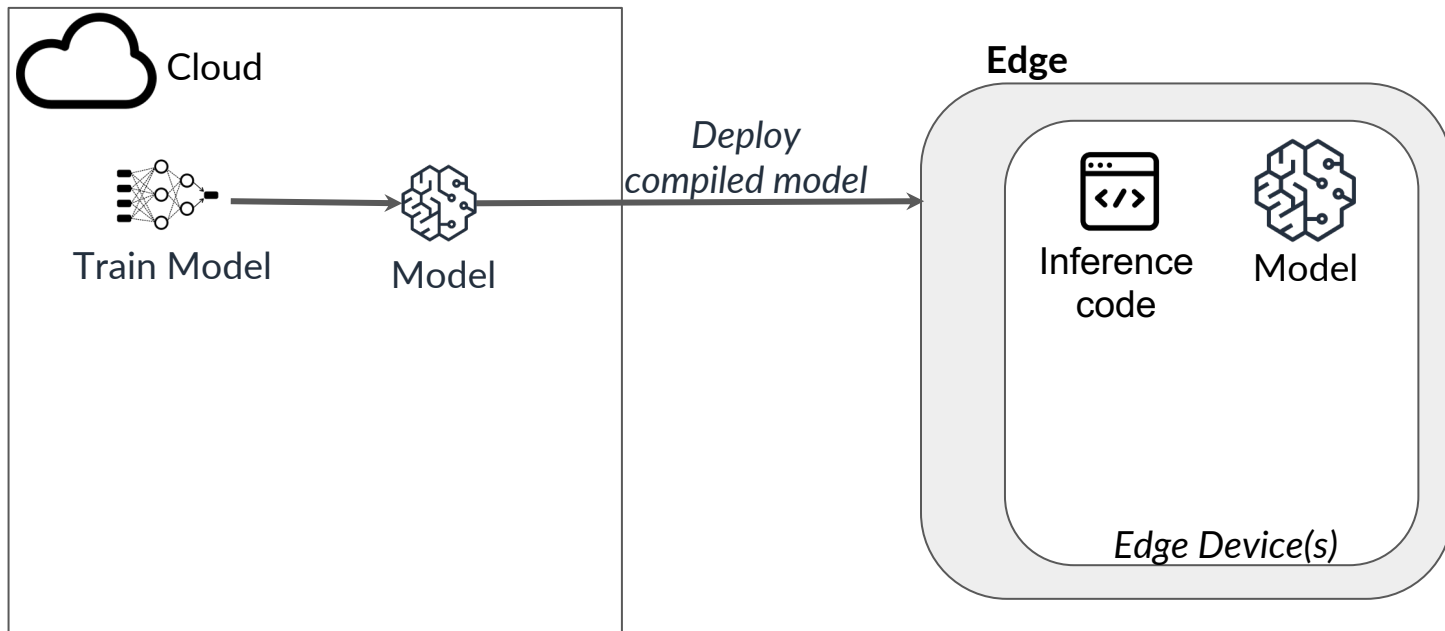


deployment to the edge which is an option that is not a cloud specific. But is a key consideration when deploying models closer to your users or in areas with poor network connectivity. In the case of edge deployments, you train your models in another environment in this case in the cloud and then optimize 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

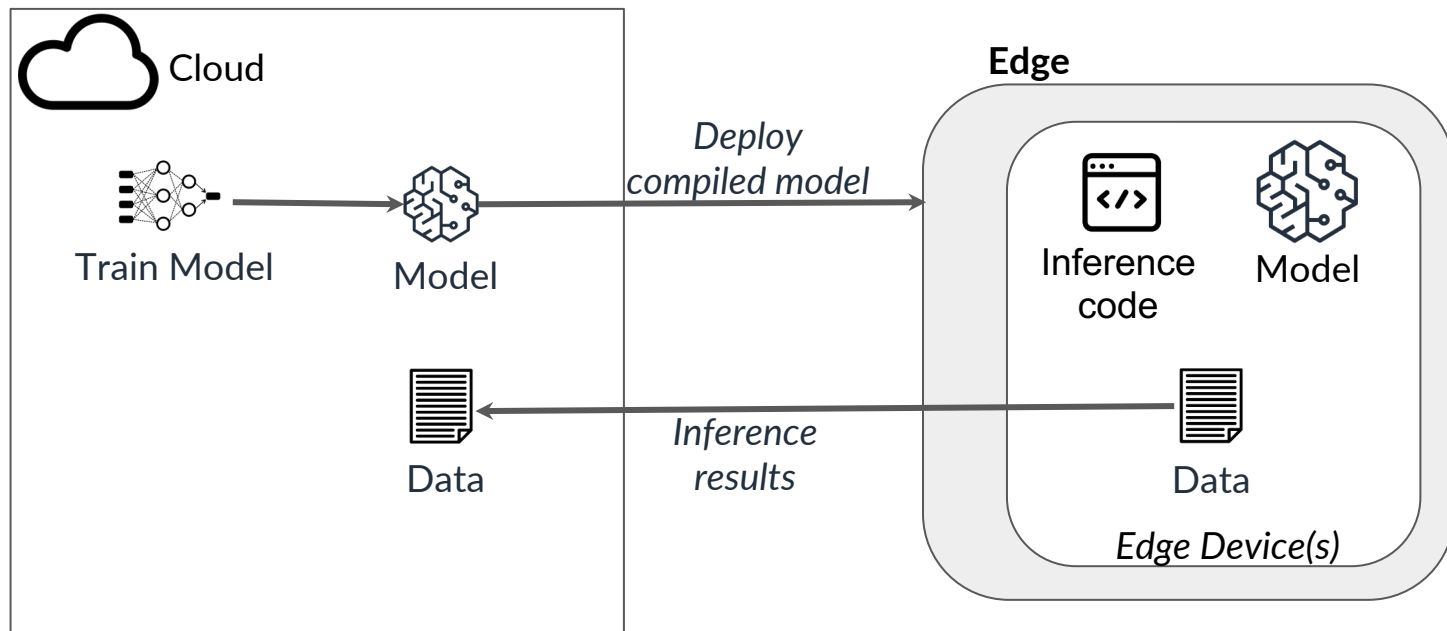
Edge

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 be 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

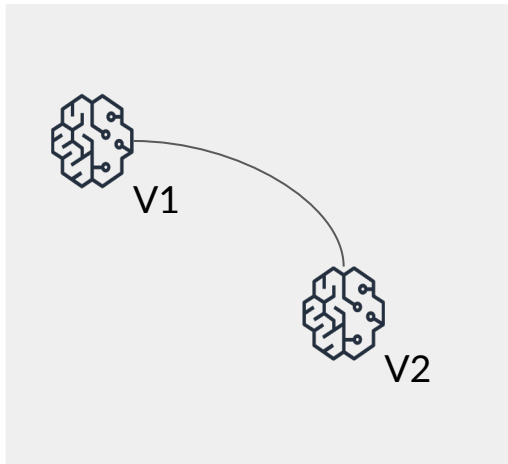
Model Deployment

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

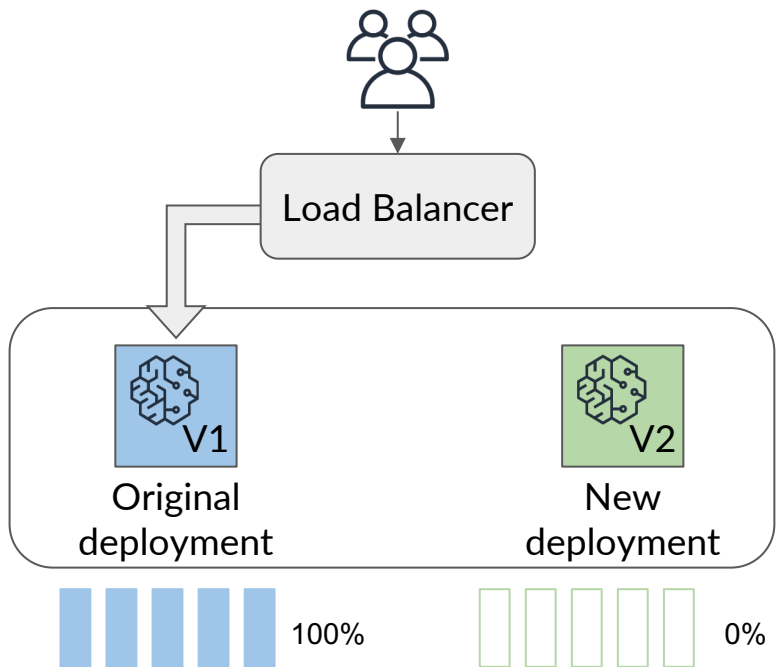
Multi-Armed
Bandits

- 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

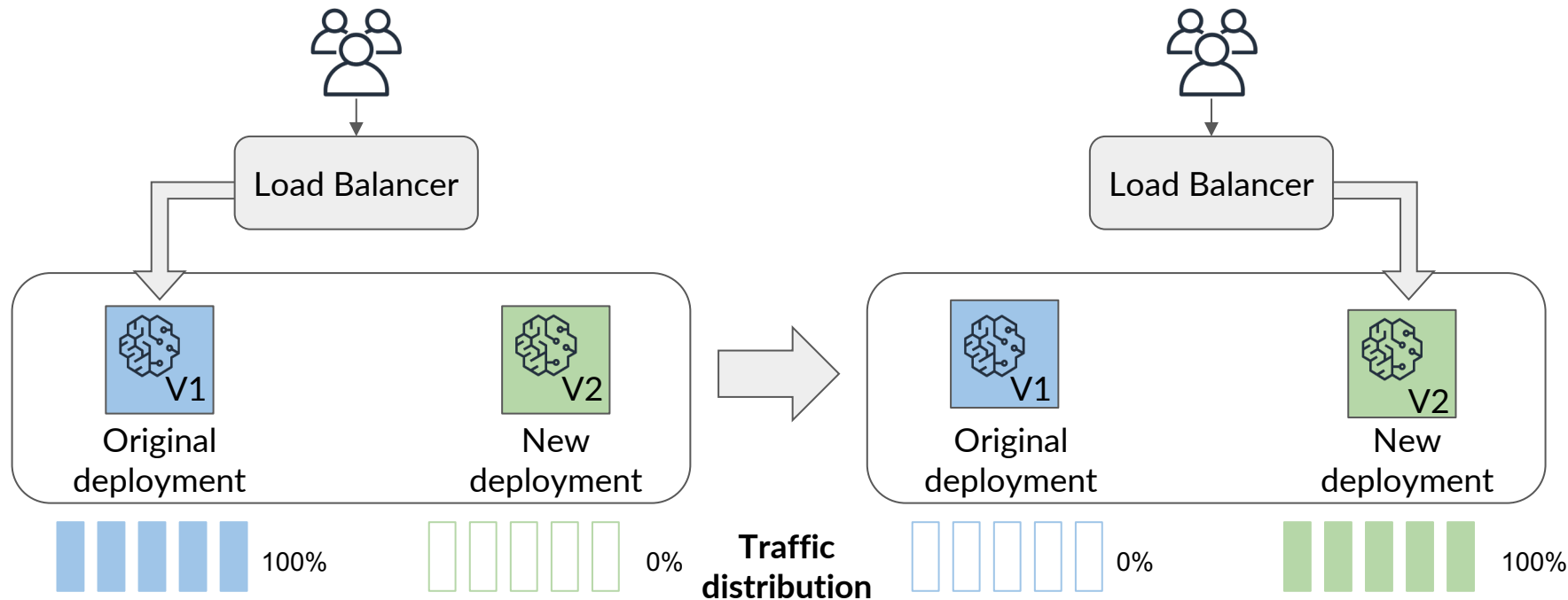
Blue/Green: Shift all traffic to the new model



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

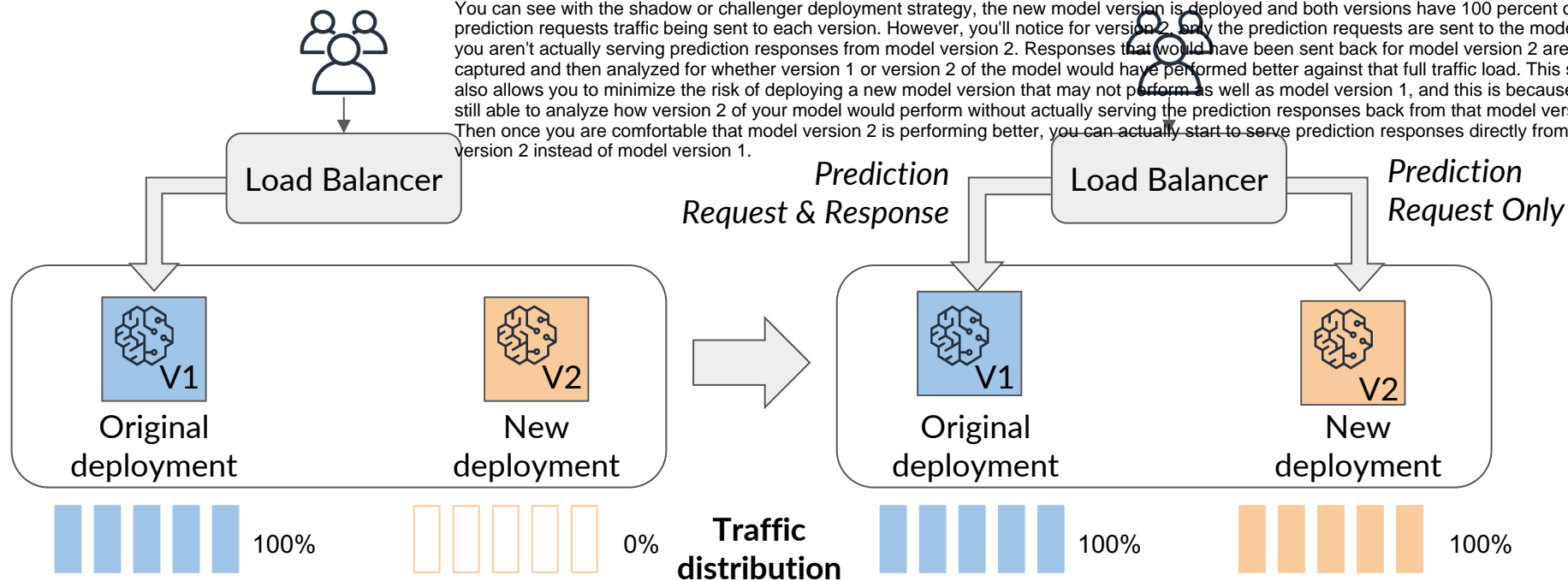
- Parallel prediction request traffic
- Validate new version without impact

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

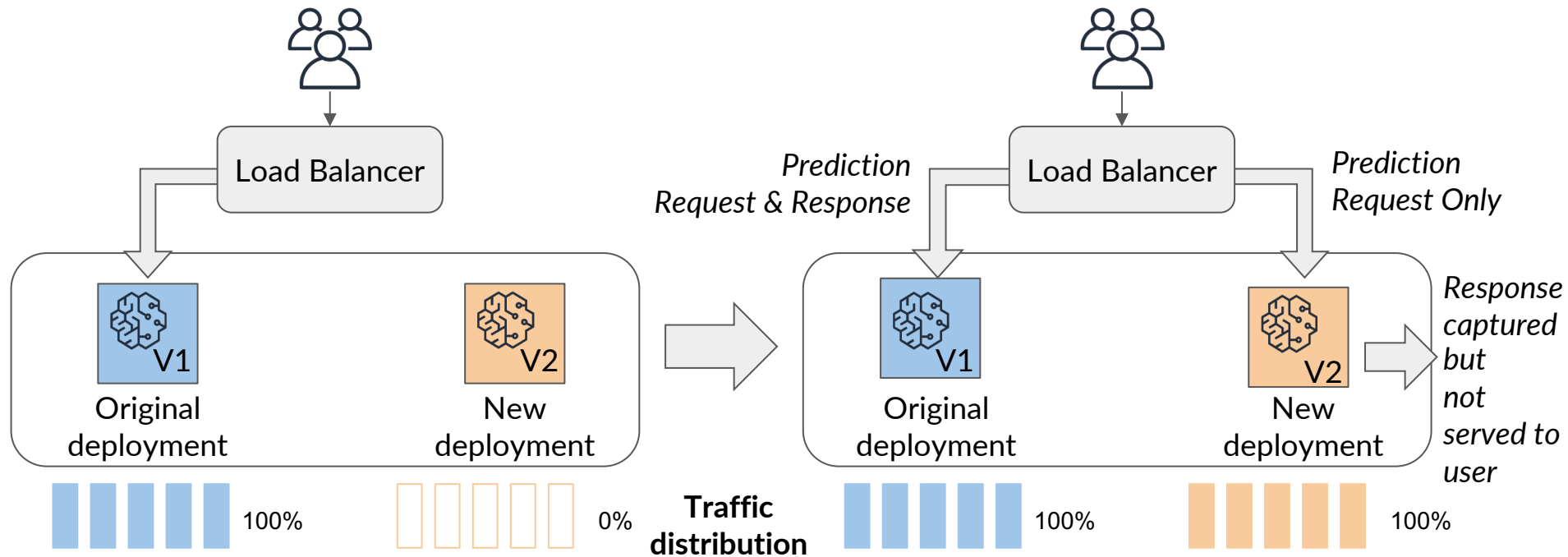
Shadow/Challenger: Run multiple versions in parallel with one serving live traffic

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

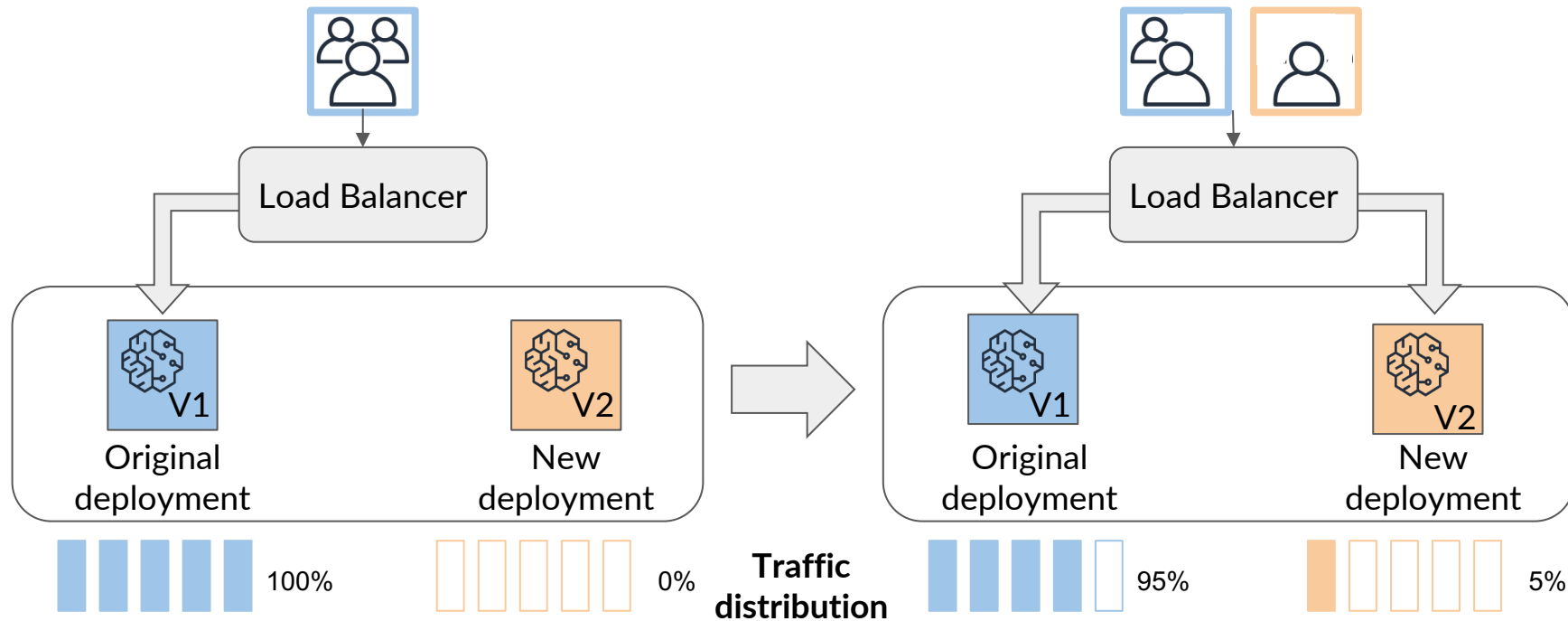
A/B

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

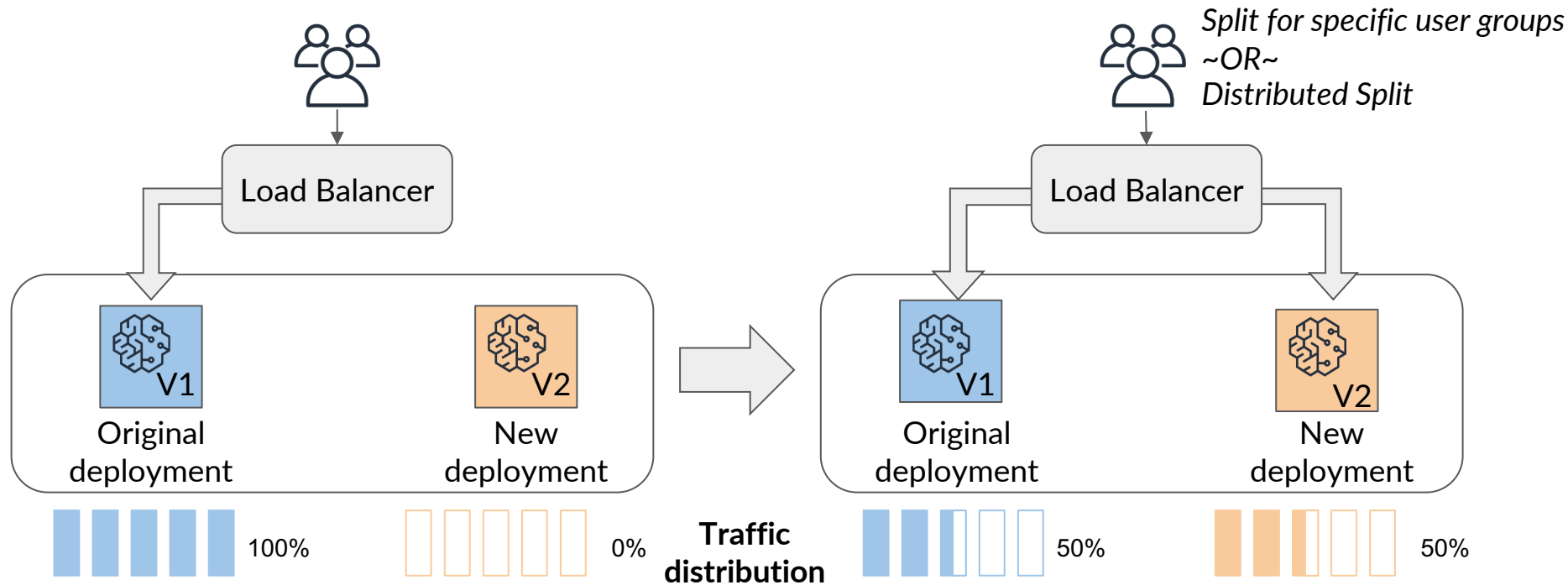
A/B

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



Deployment Strategies

Common Strategies to deploy new and updated models

Blue/Green

Shadow/
Challenger

Canary

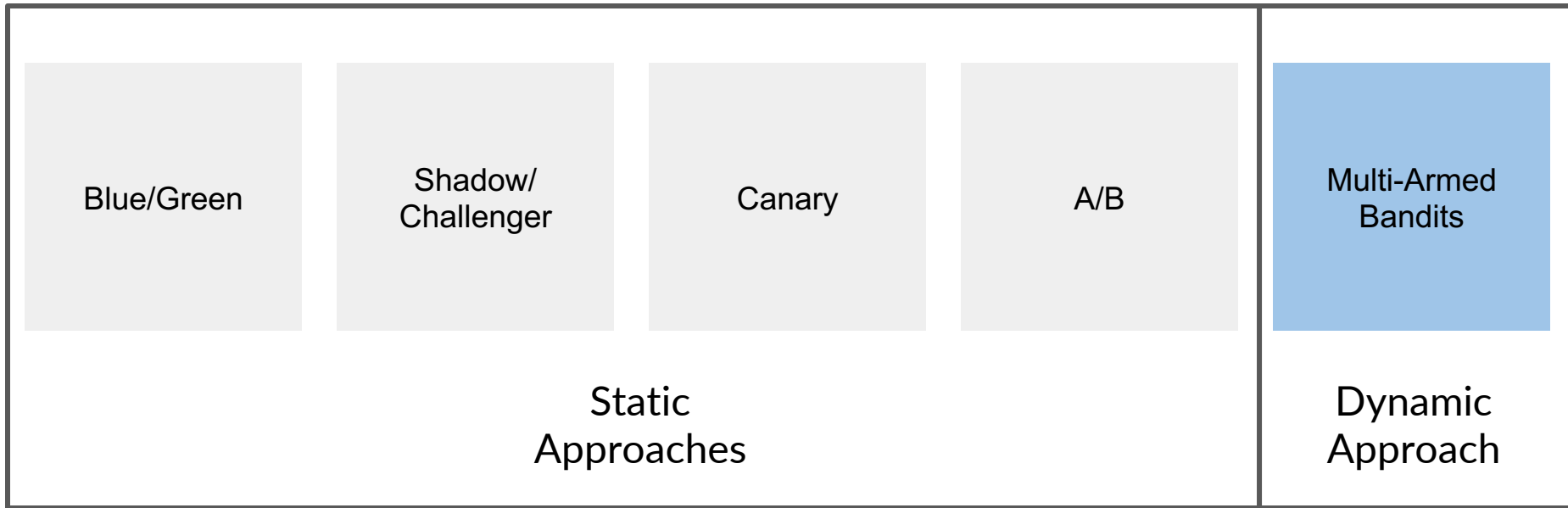
A/B

Multi-Armed
Bandits

Static
Approaches

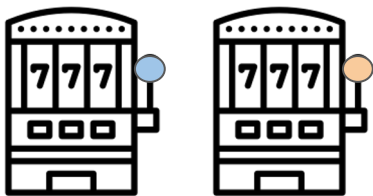
Deployment Strategies

Common Strategies to deploy new and updated models



Deployment Strategies

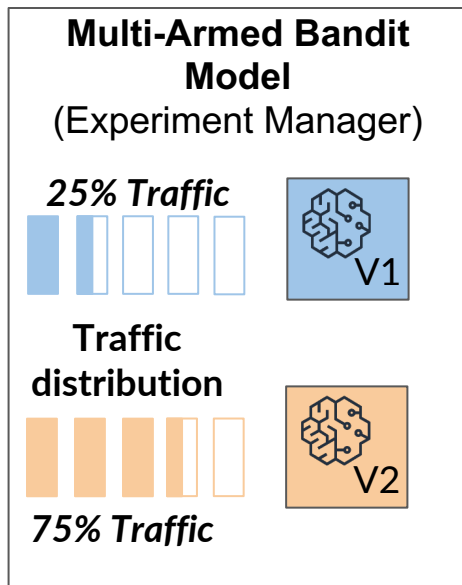
Multi-Armed Bandits (MABs)



- 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



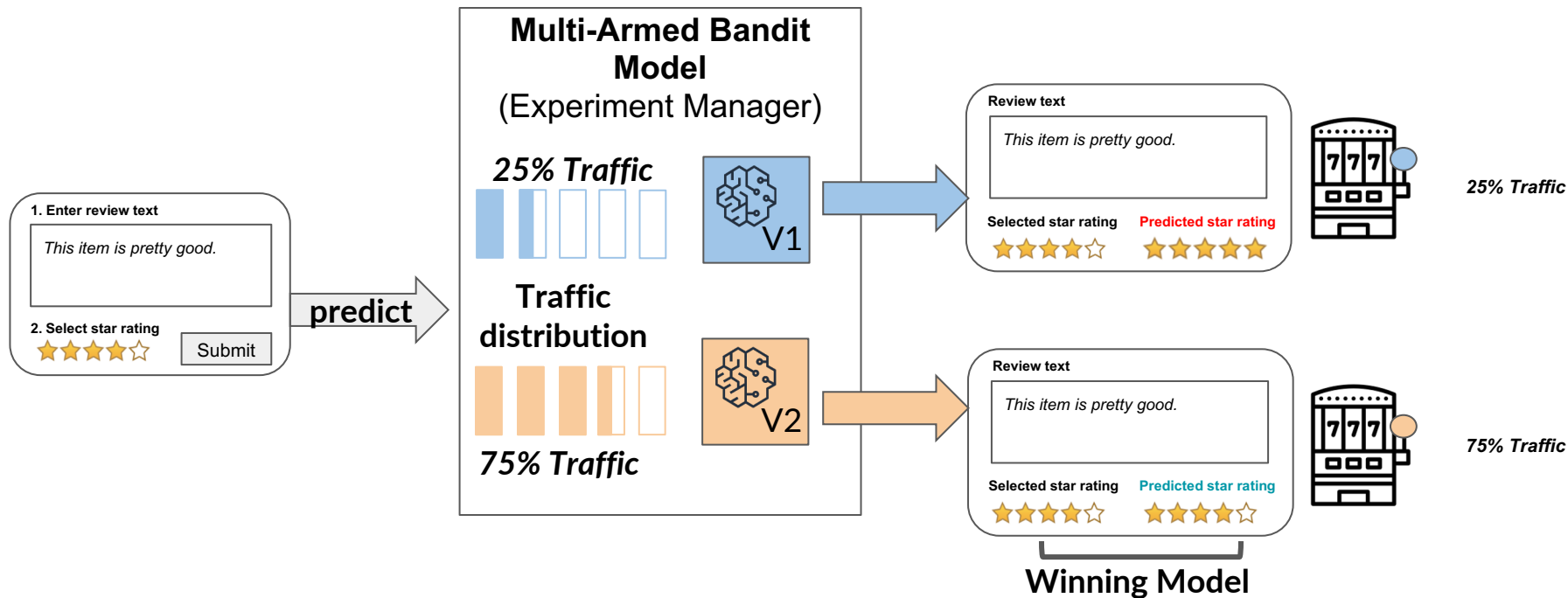
25% Traffic



75% Traffic

Deployment Strategies

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

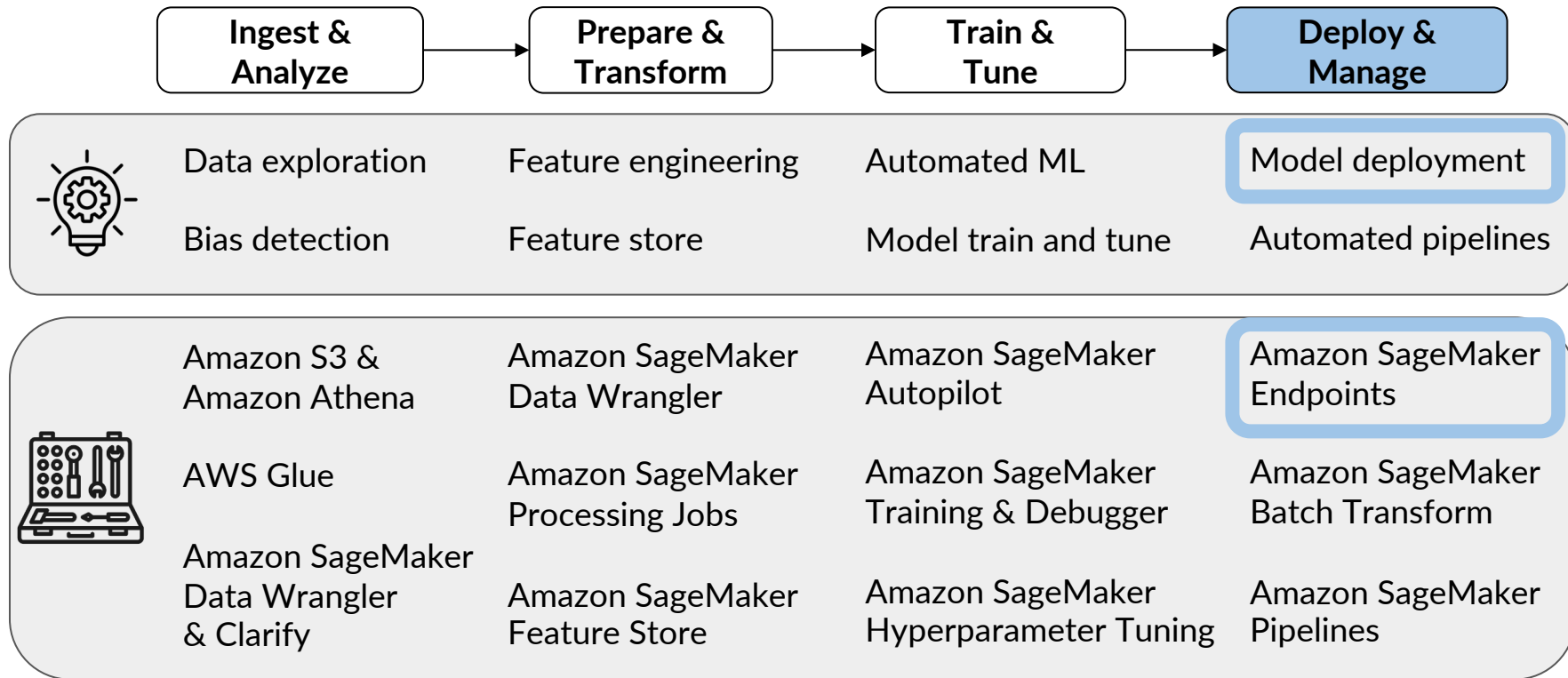


Amazon SageMaker Hosting

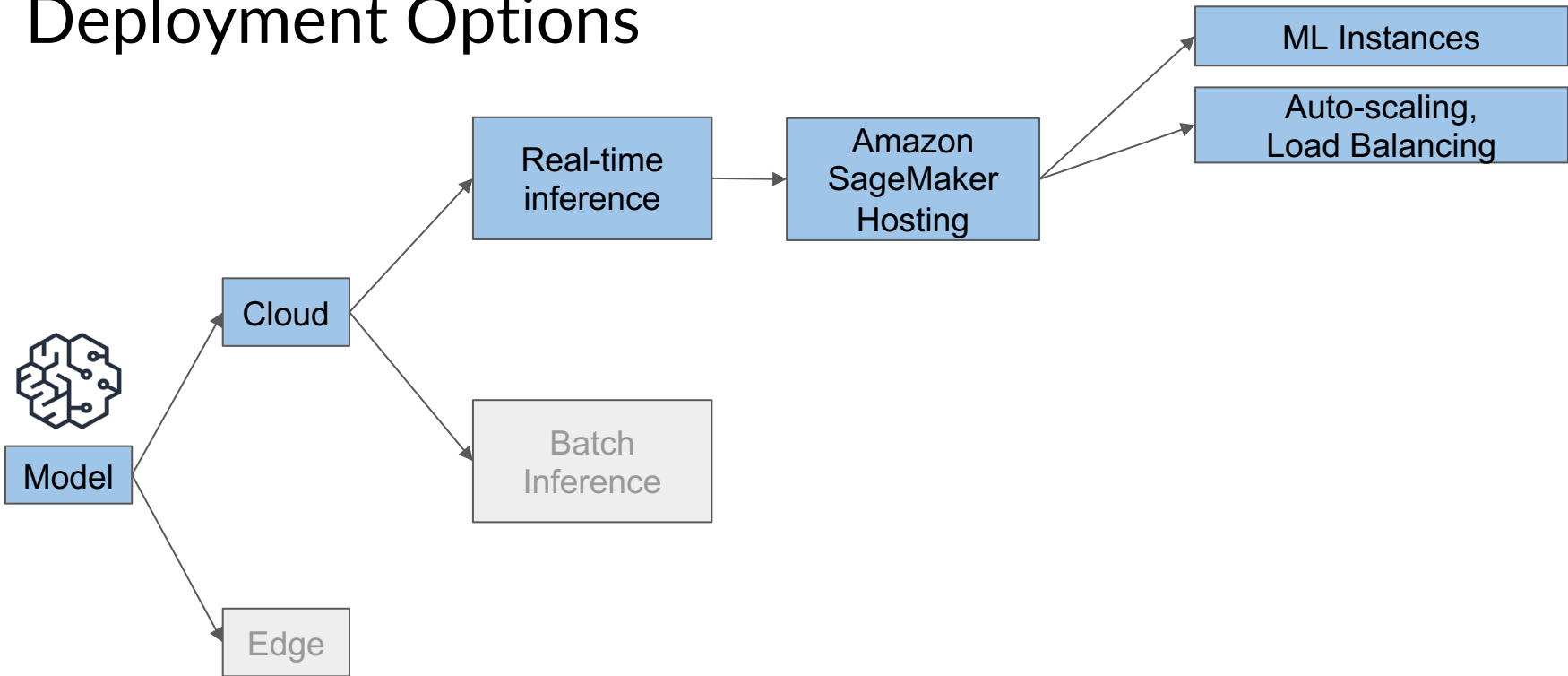
Real-Time Inference



Machine Learning Workflow



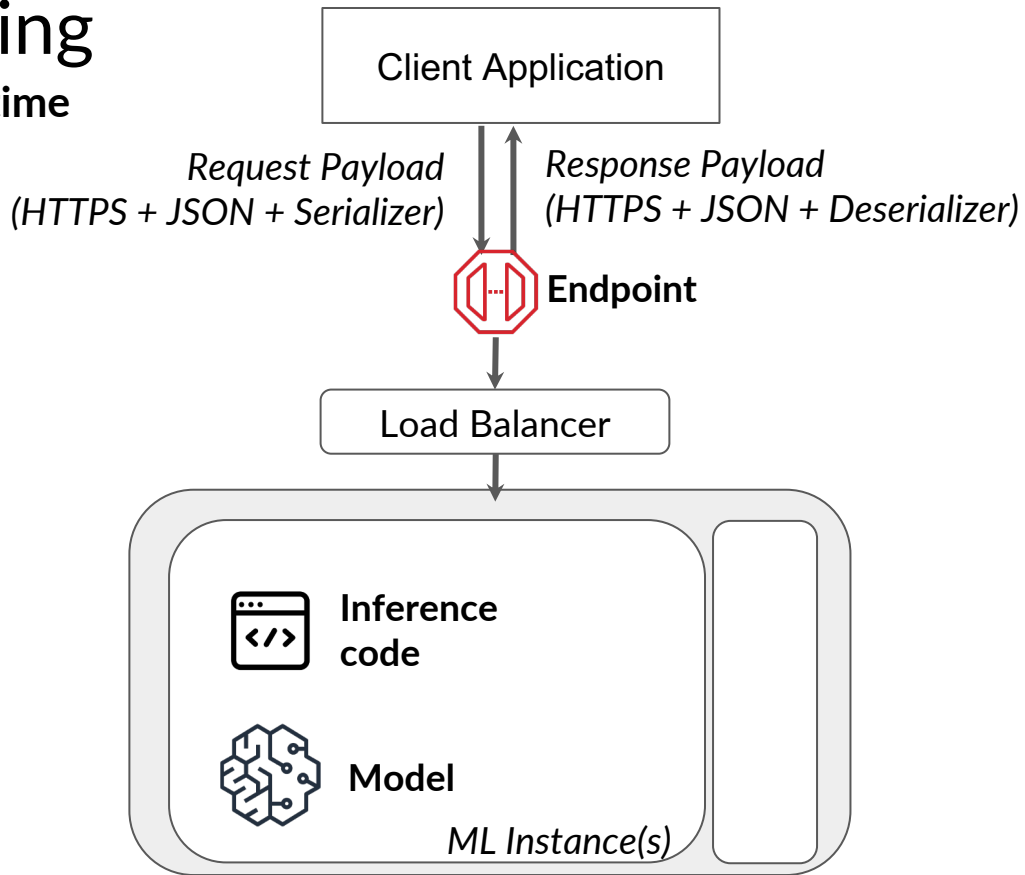
Deployment Options



Amazon SageMaker Hosting

Deploy models to serve predictions in real-time

- 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

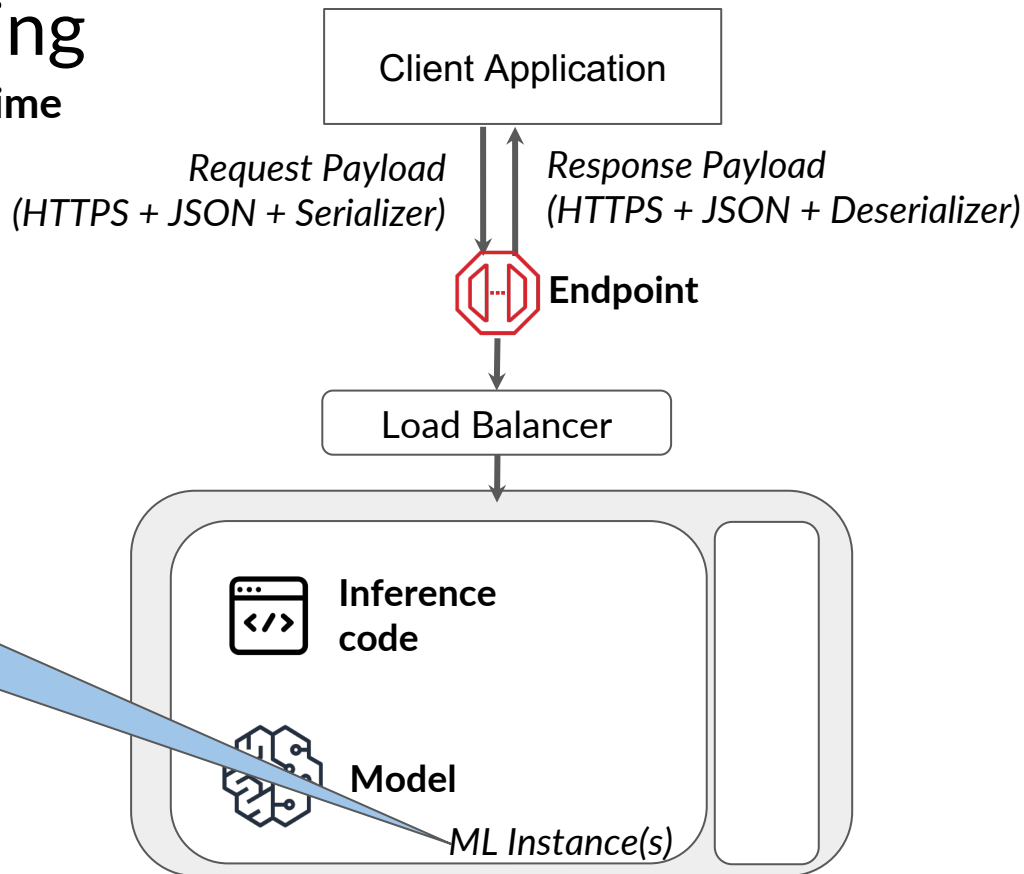


Amazon SageMaker Hosting

Deploy models to serve predictions in real-time

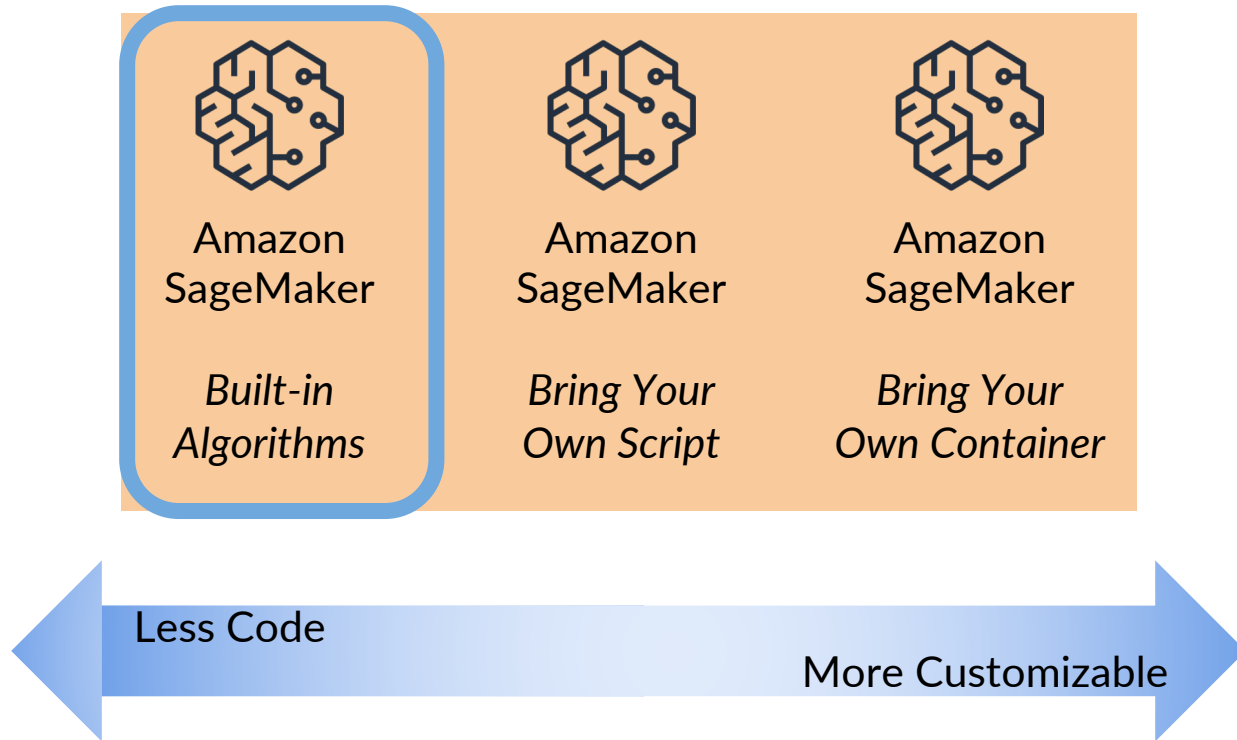
You choose:

- Instance Type
- Instance Size
- Number of Instances
- Autoscaling Options



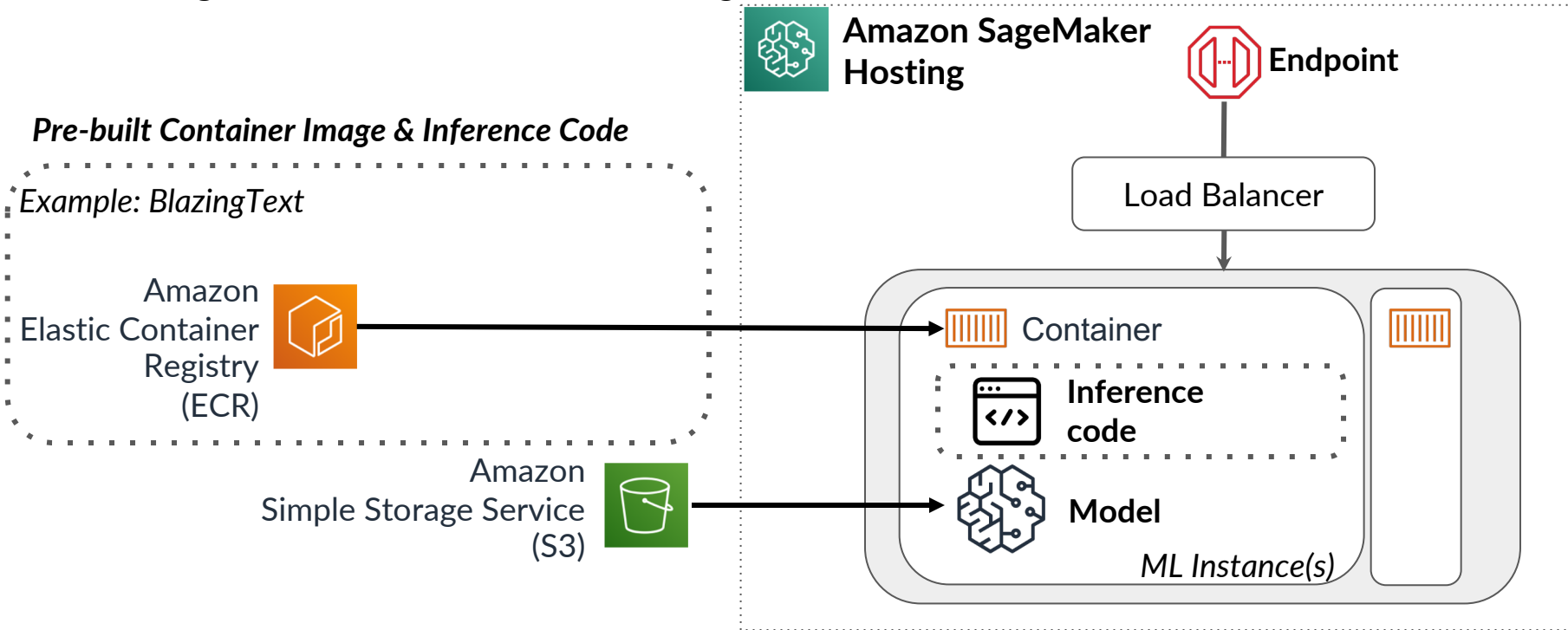
Amazon SageMaker Hosting

Options to deploy models to serve predictions in real-time



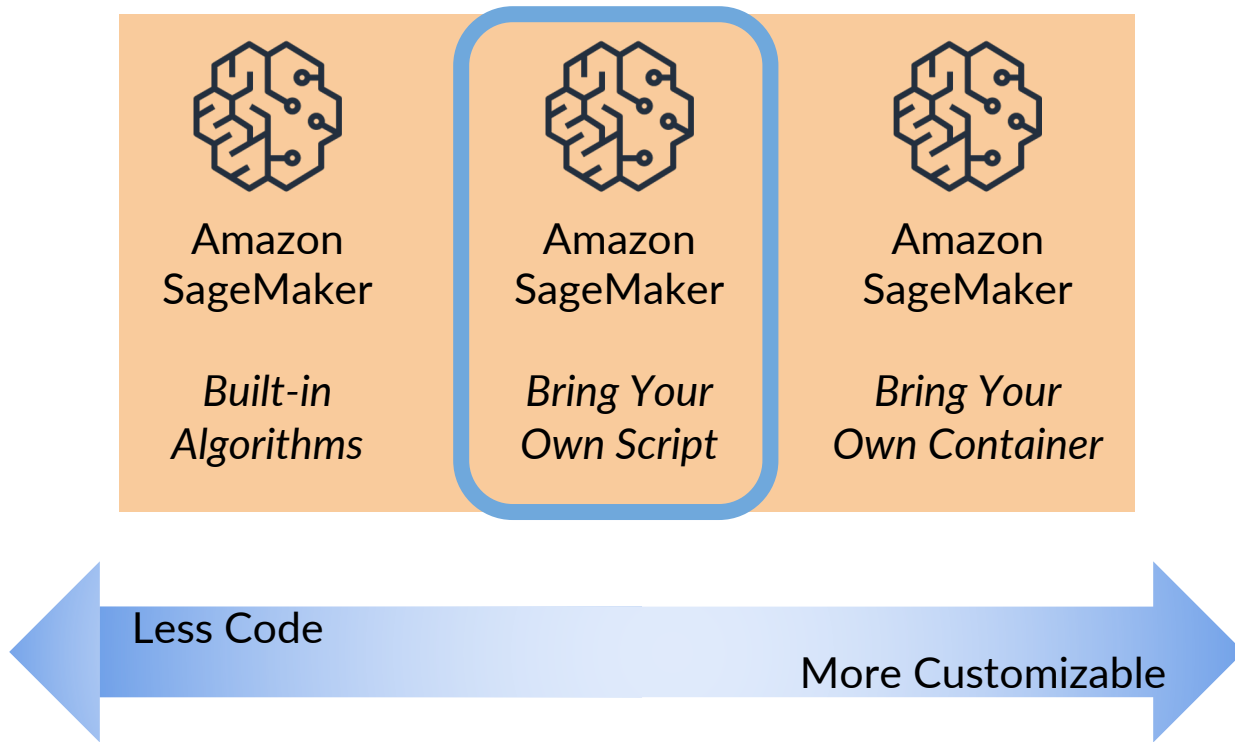
Amazon SageMaker Hosting

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



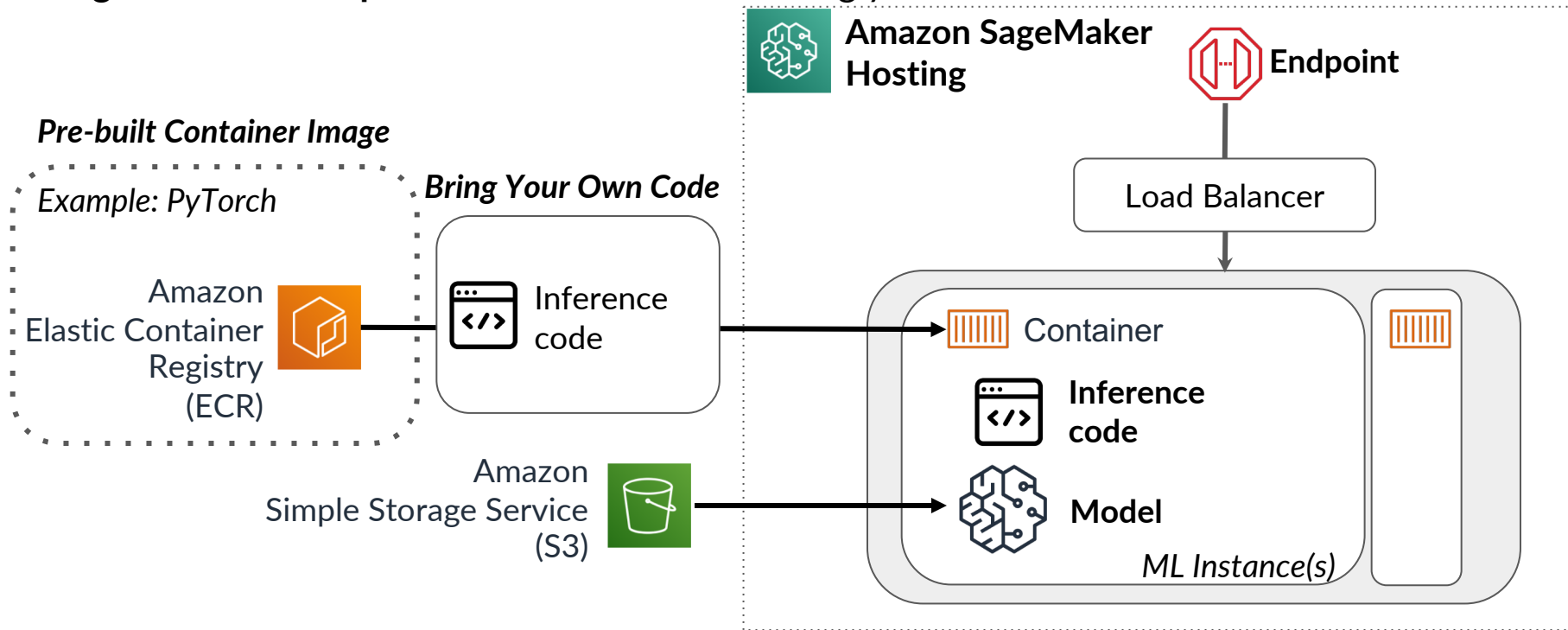
Amazon SageMaker Hosting

Options to deploy models to serve predictions in real-time



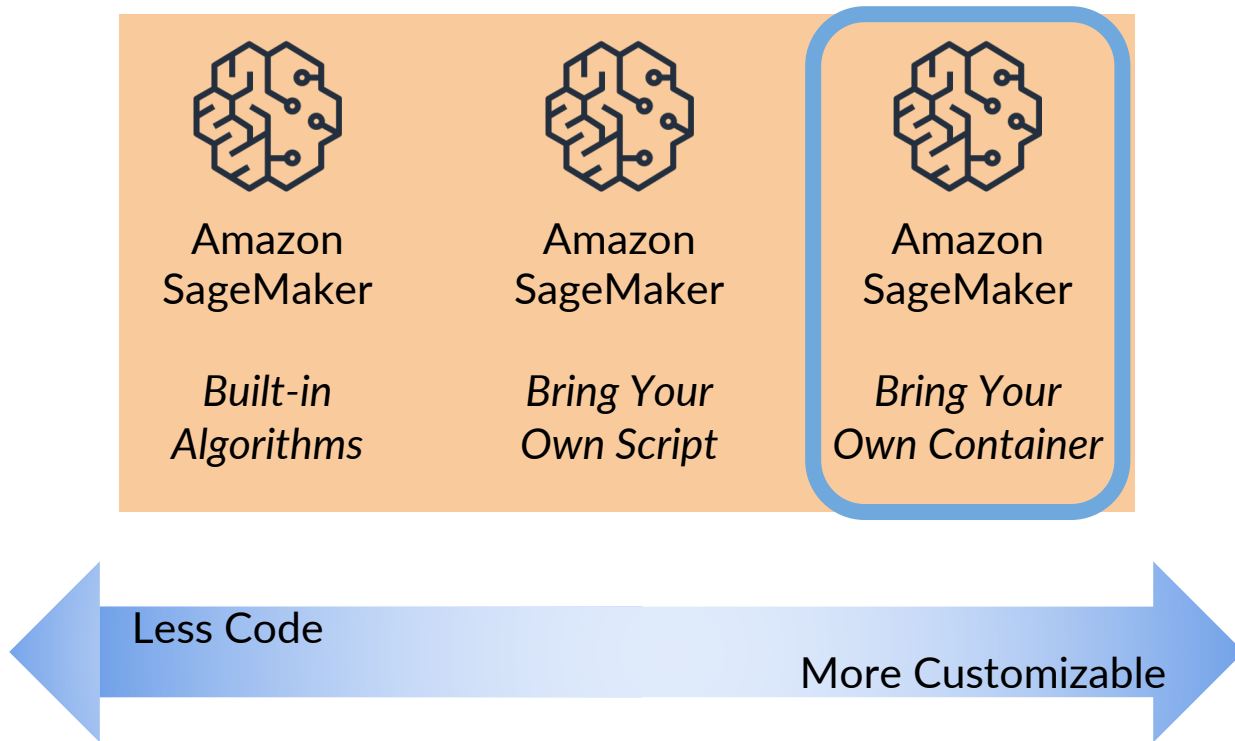
Amazon SageMaker Hosting

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



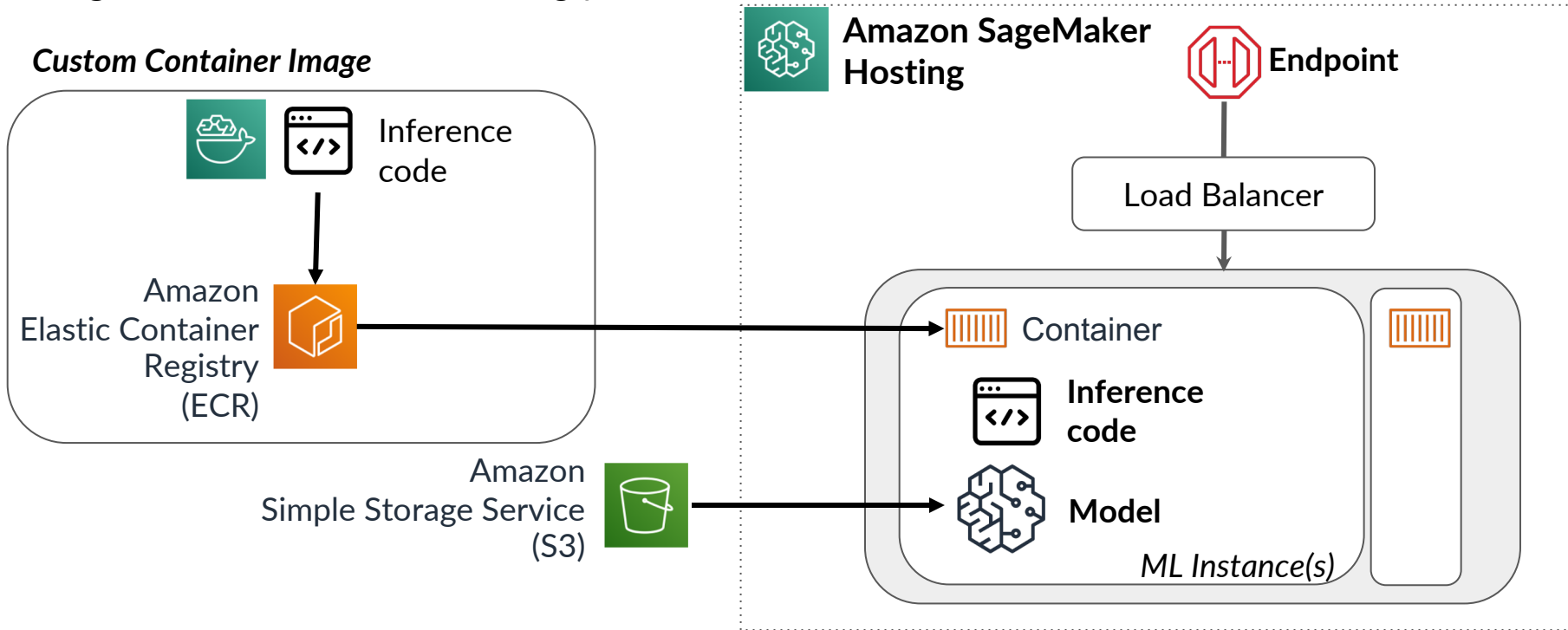
Amazon SageMaker Hosting

Options to deploy models to serve predictions in real-time



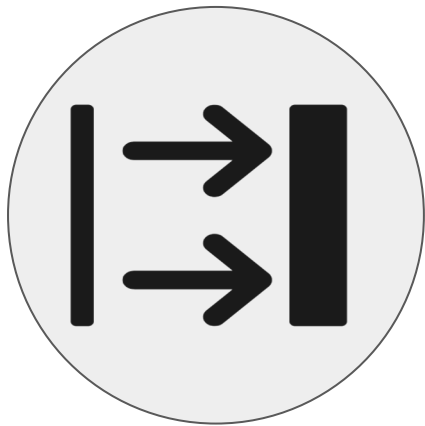
Amazon SageMaker Hosting

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



Amazon SageMaker Hosting

Autoscaling Endpoints

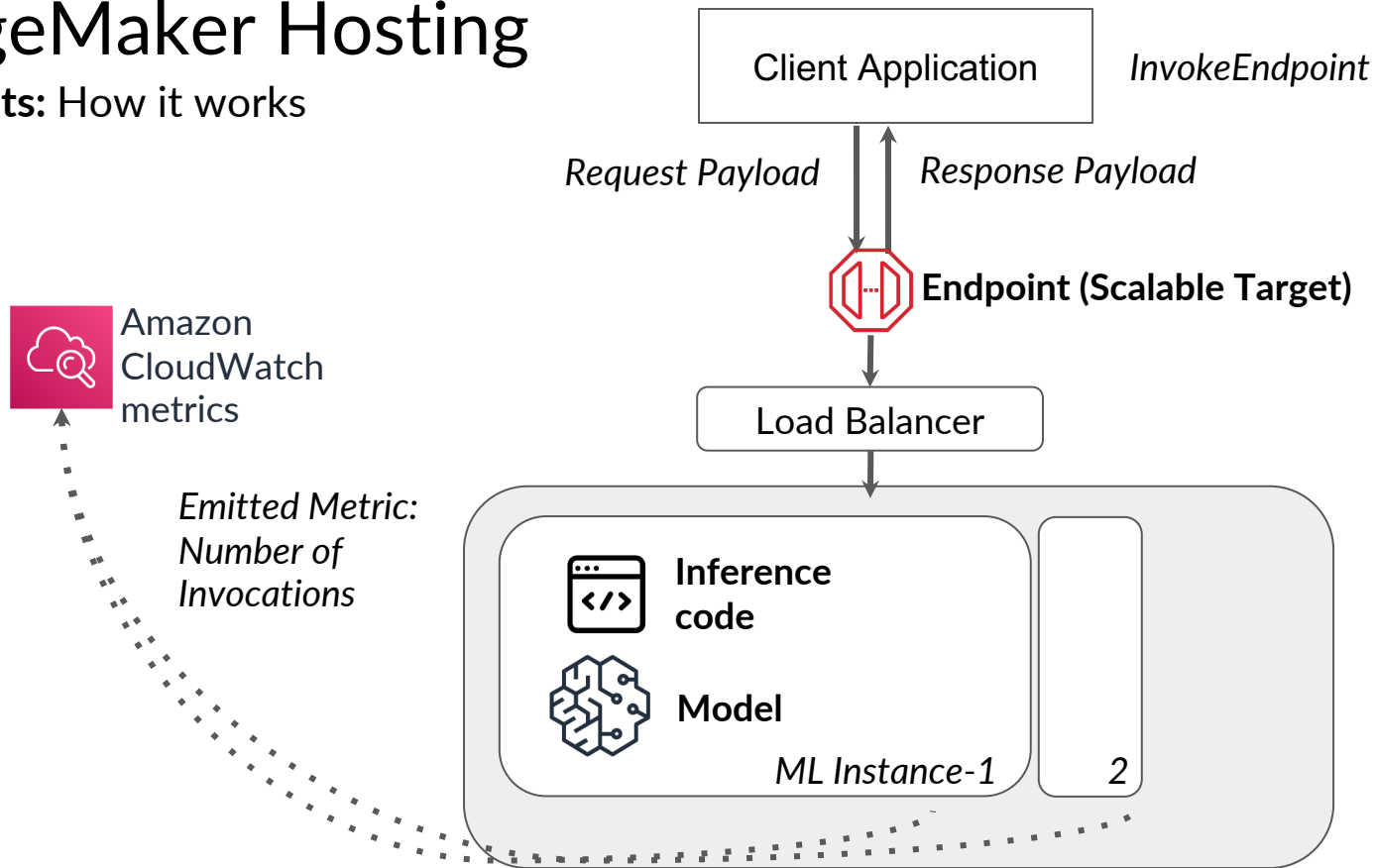


Why?

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

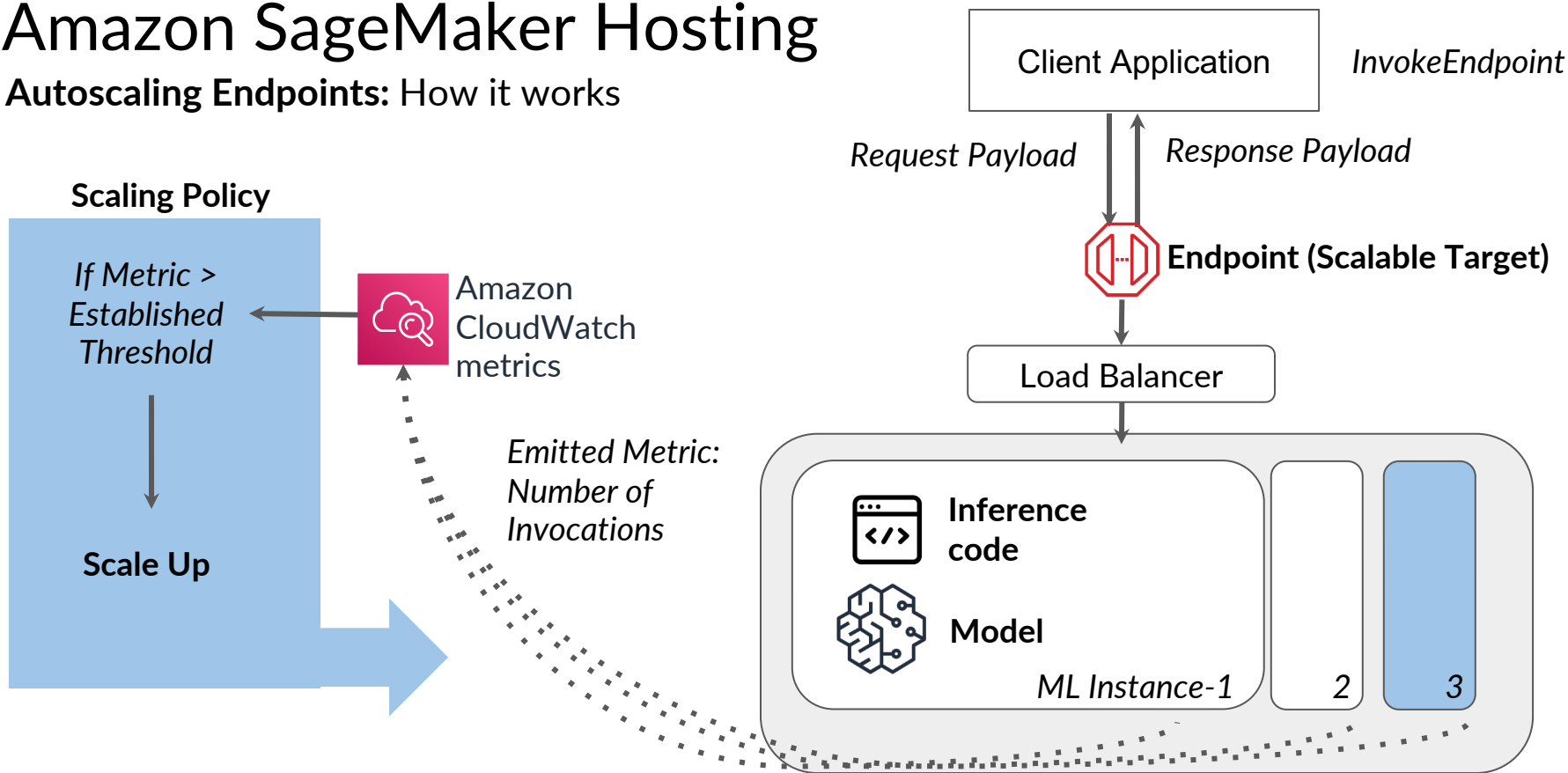
Amazon SageMaker Hosting

Autoscaling Endpoints: How it works



Amazon SageMaker Hosting

Autoscaling Endpoints: How it works

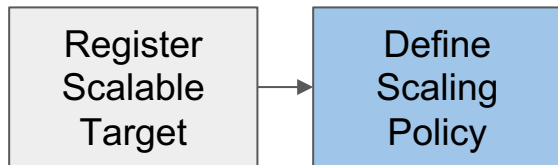


Autoscale Amazon SageMaker Endpoints

Register
Scalable
Target

```
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



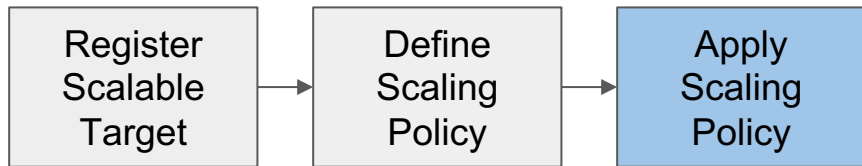
```
scaling_policy = {  
    "TargetValue": 2.0,  
    "PredefinedMetricSpecification": {  
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance",  
    },  
    "ScaleOutCooldown": 60,  
    "ScaleInCooldown": 300,  
},  
}
```

Scaling Metric

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

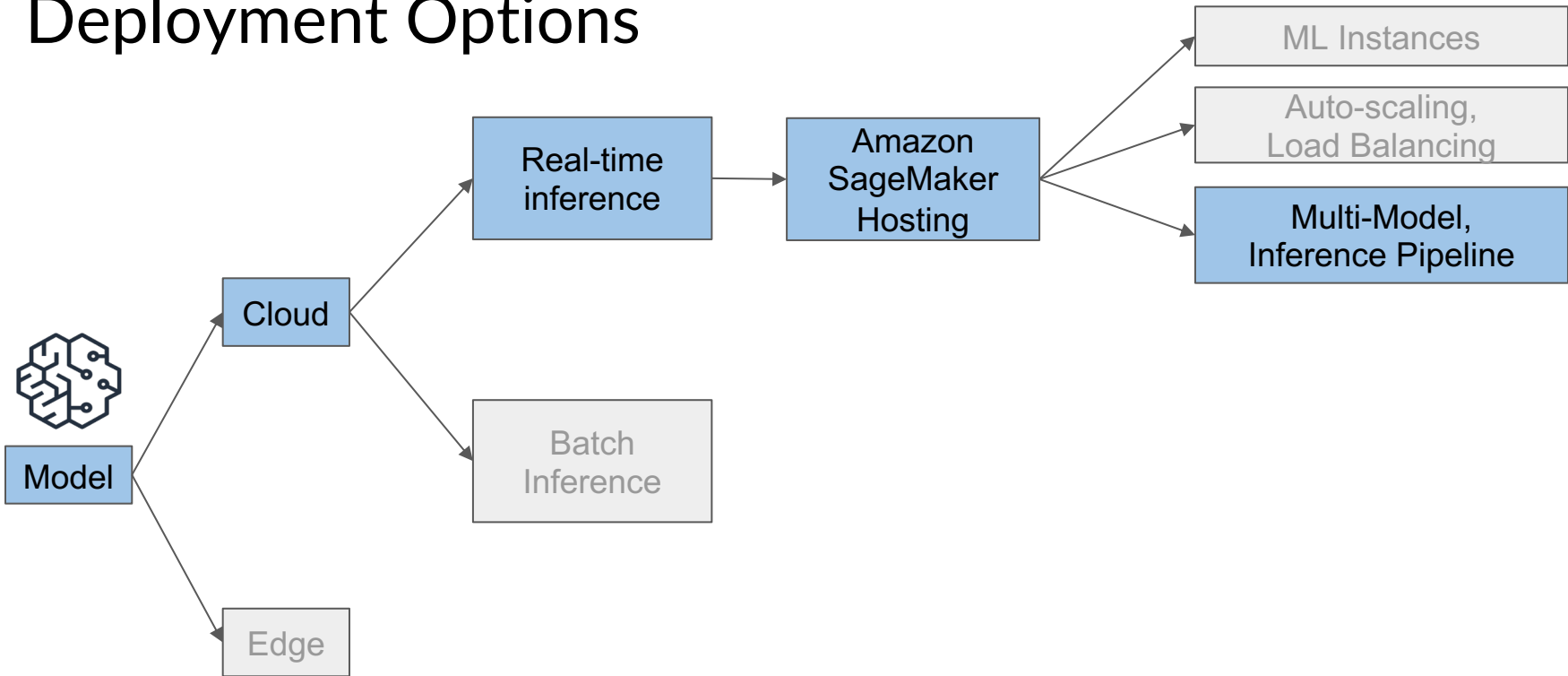
Wait time, in seconds, before beginning another **scale in** activity after last one completes

Autoscale Amazon SageMaker Endpoints



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

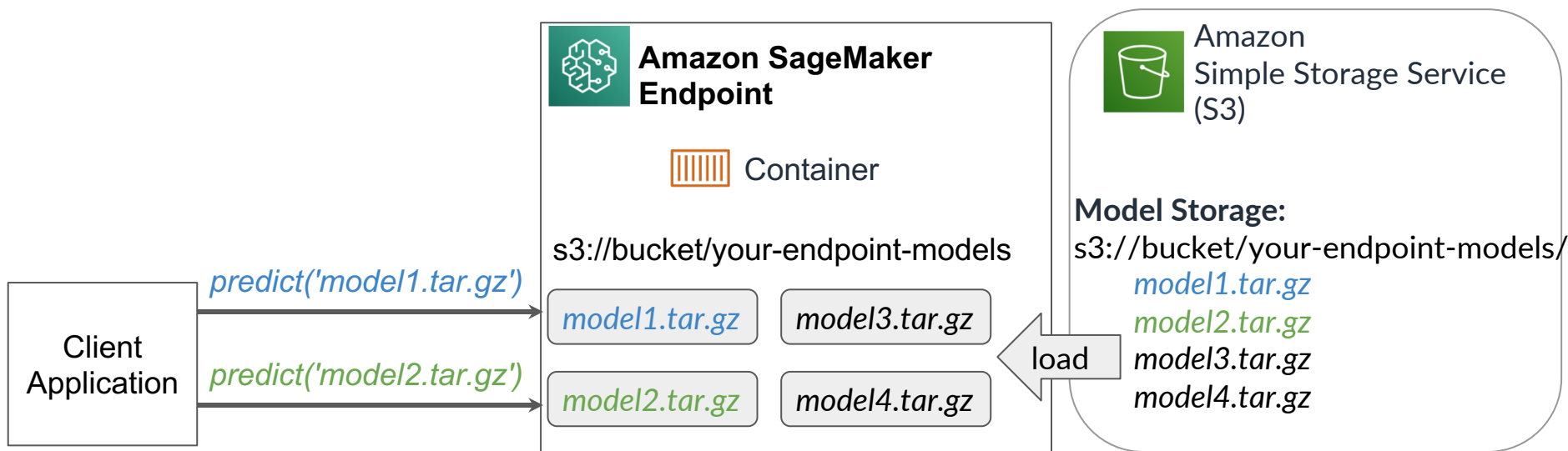
Deployment Options



Advanced Deployment Options

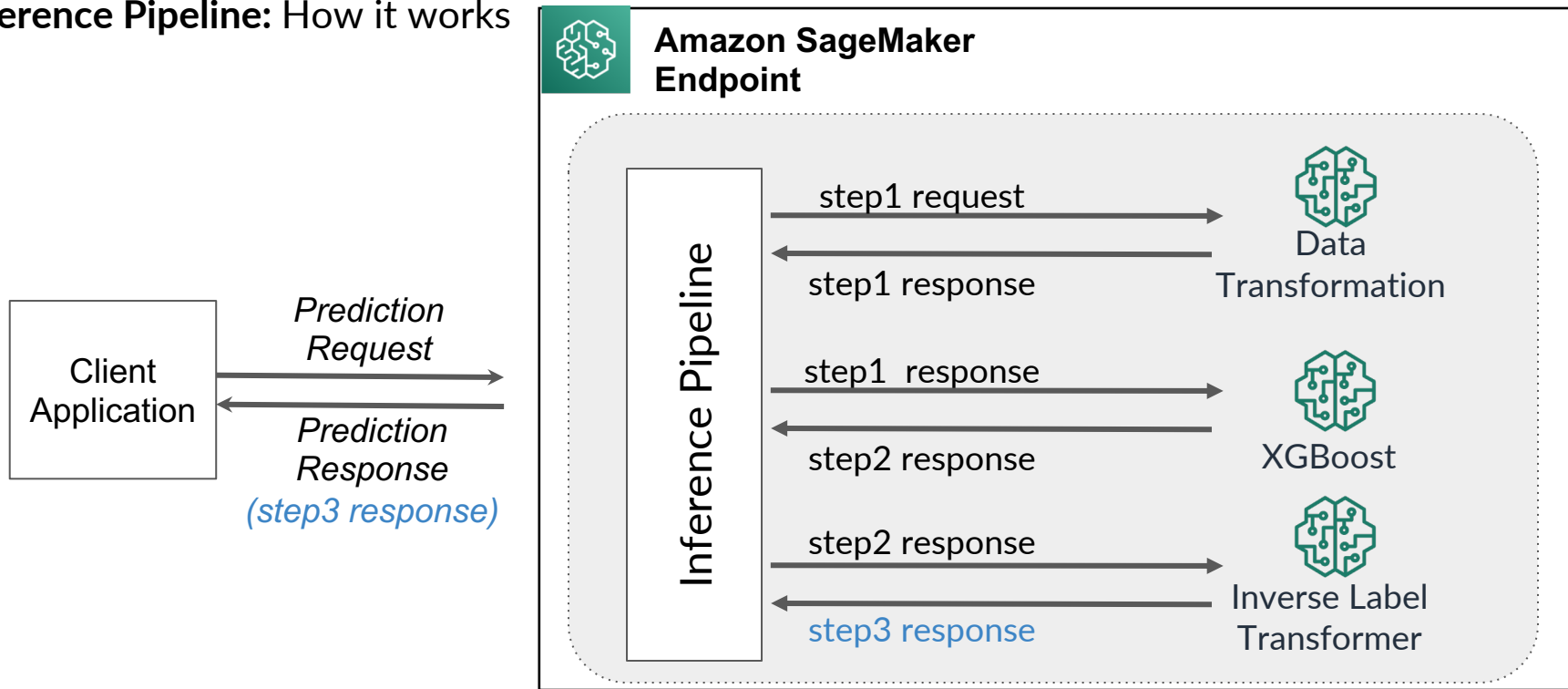
Multi-Model Endpoints: How it works

Deploy Multiple Models to a Single Endpoint



Advanced Deployment Options

Inference Pipeline: How it works

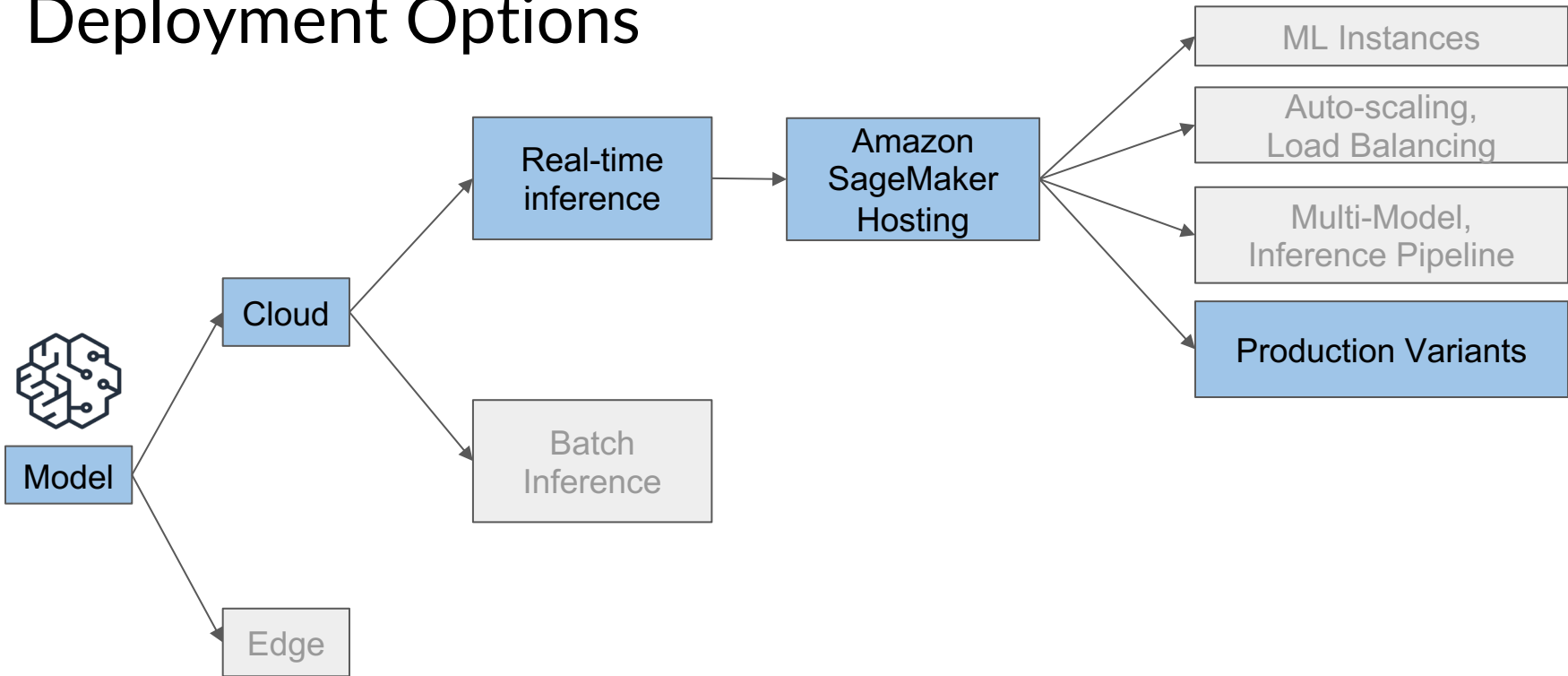


Amazon SageMaker Hosting

Real-Time Inference
Production Variants

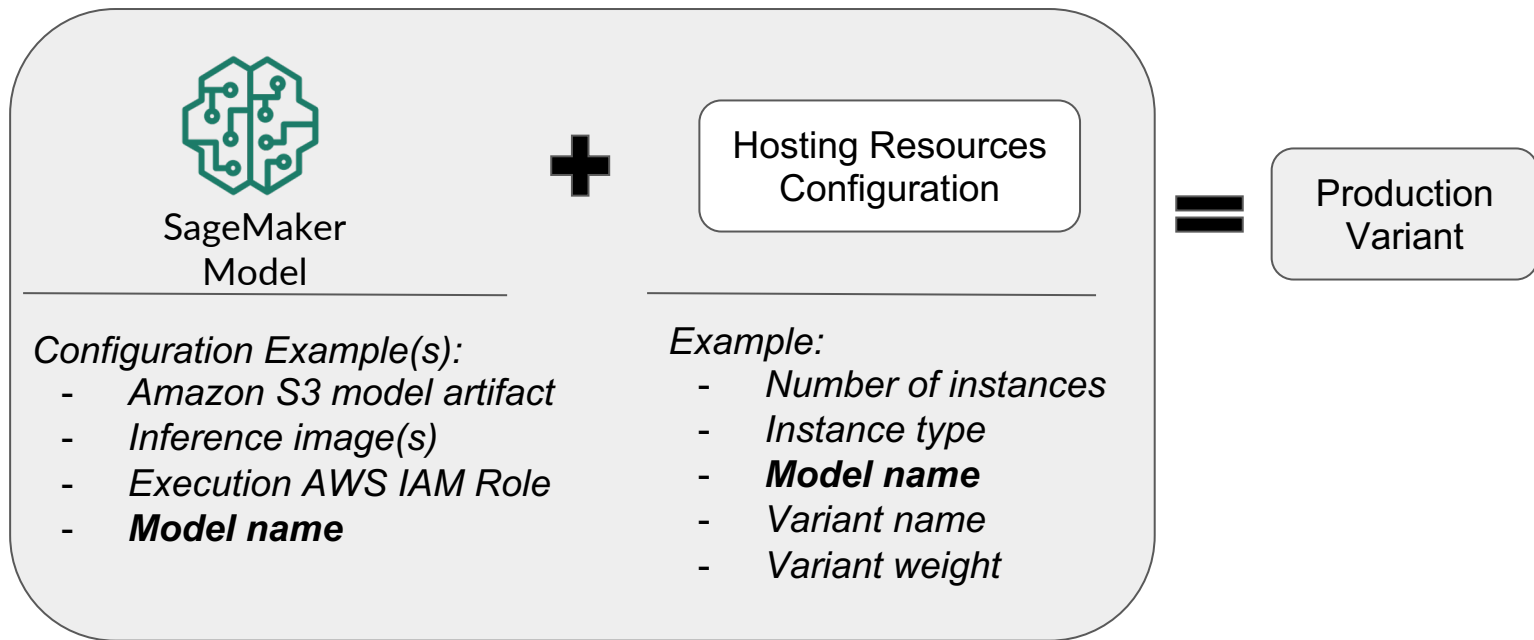


Deployment Options



Amazon SageMaker Production Variants

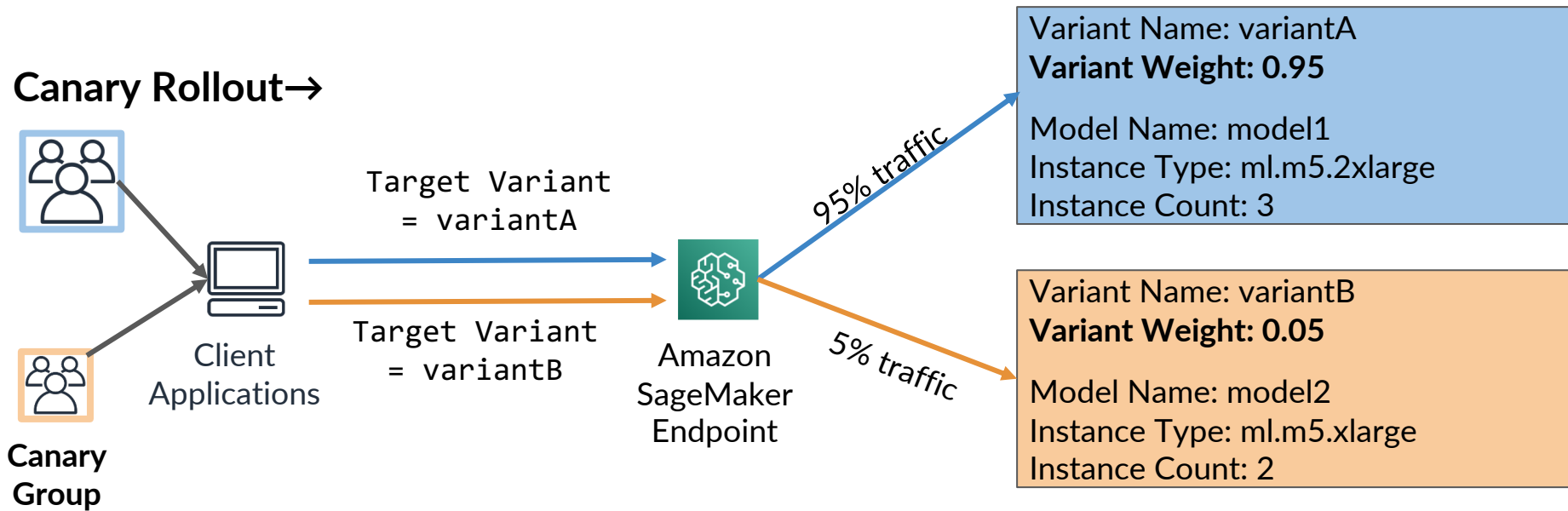
What is a Production Variant?



Amazon SageMaker Production Variants

Using Production Variants for a Canary Rollout

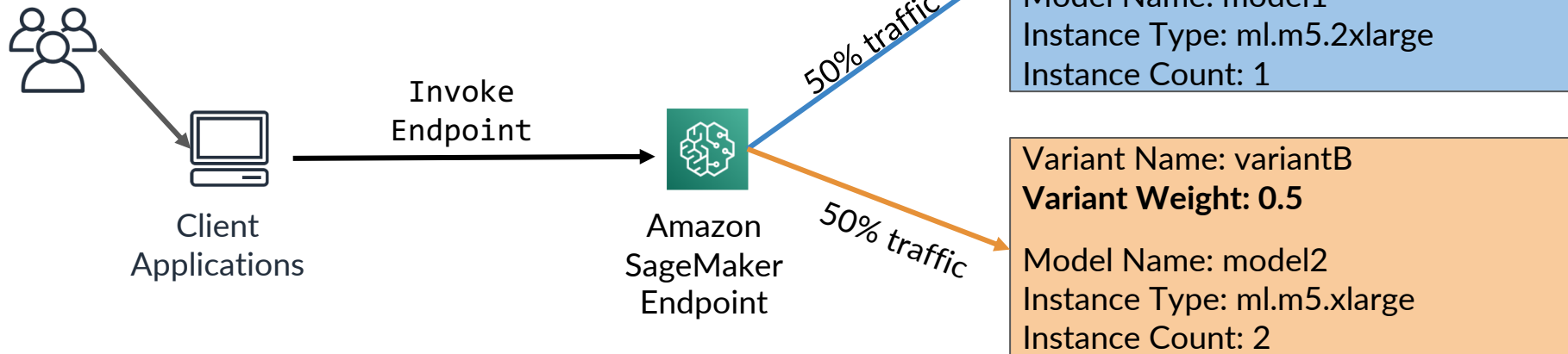
Canary Rollout→



Amazon SageMaker Production Variants

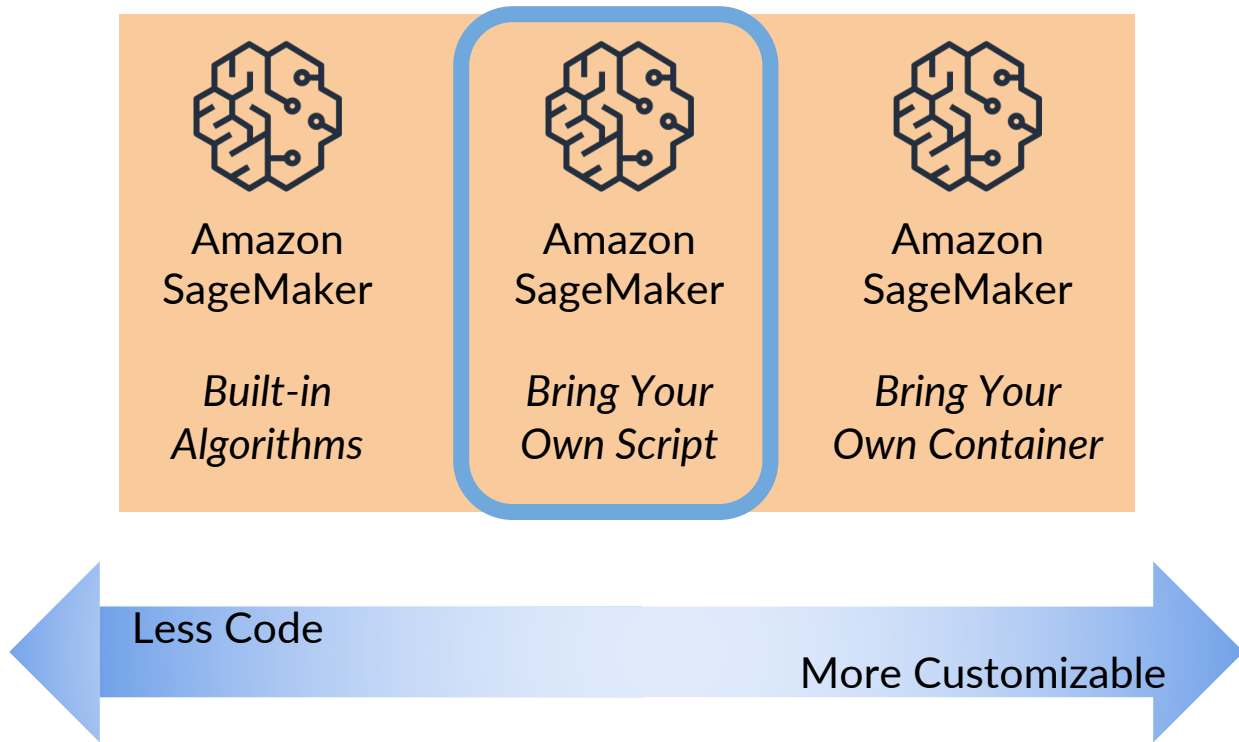
Using Production Variants for A/B Testing

A/B Testing →



Amazon SageMaker Hosting

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



Using Production Variants for A/B Testing

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

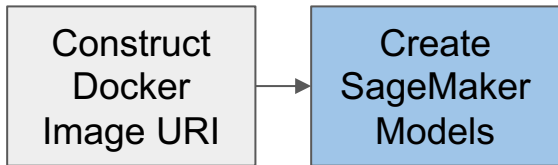
Construct
Docker
Image URI

```
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



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



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



Using Production Variants for A/B Testing

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



```
from sagemaker.session \
    import production_variant

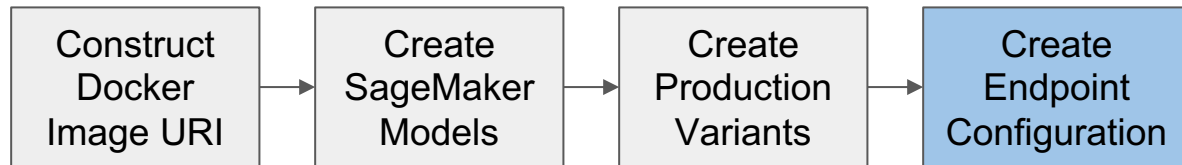
variantA = production_variant(
    model_name=...,
    instance_type=...,
    initial_instance_count=1,
    variant_name='VariantA',
    initial_weight=50,
)
```

```
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



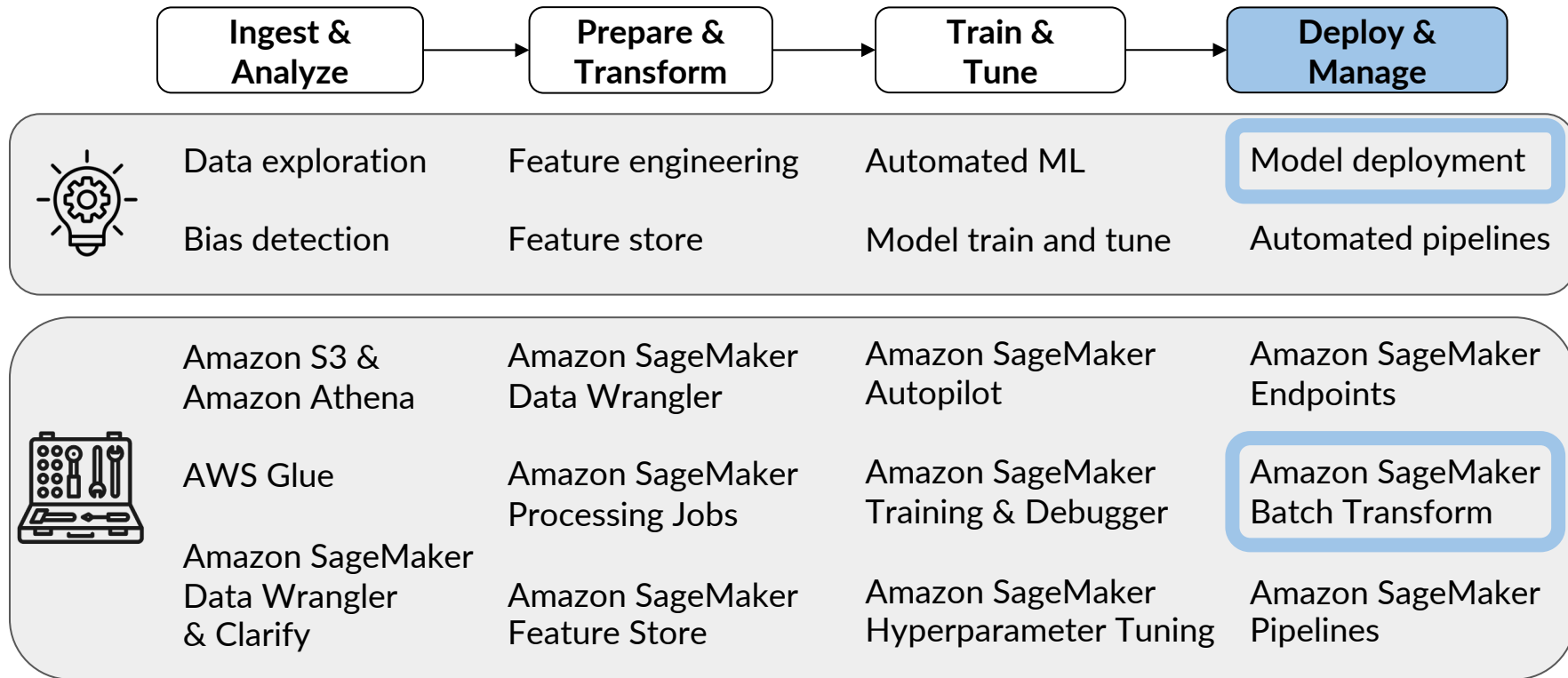
```
endpoint_response = sm.create_endpoint(EndpointName=...,  
                                       EndpointConfigName=...)
```

Amazon SageMaker Batch Transform

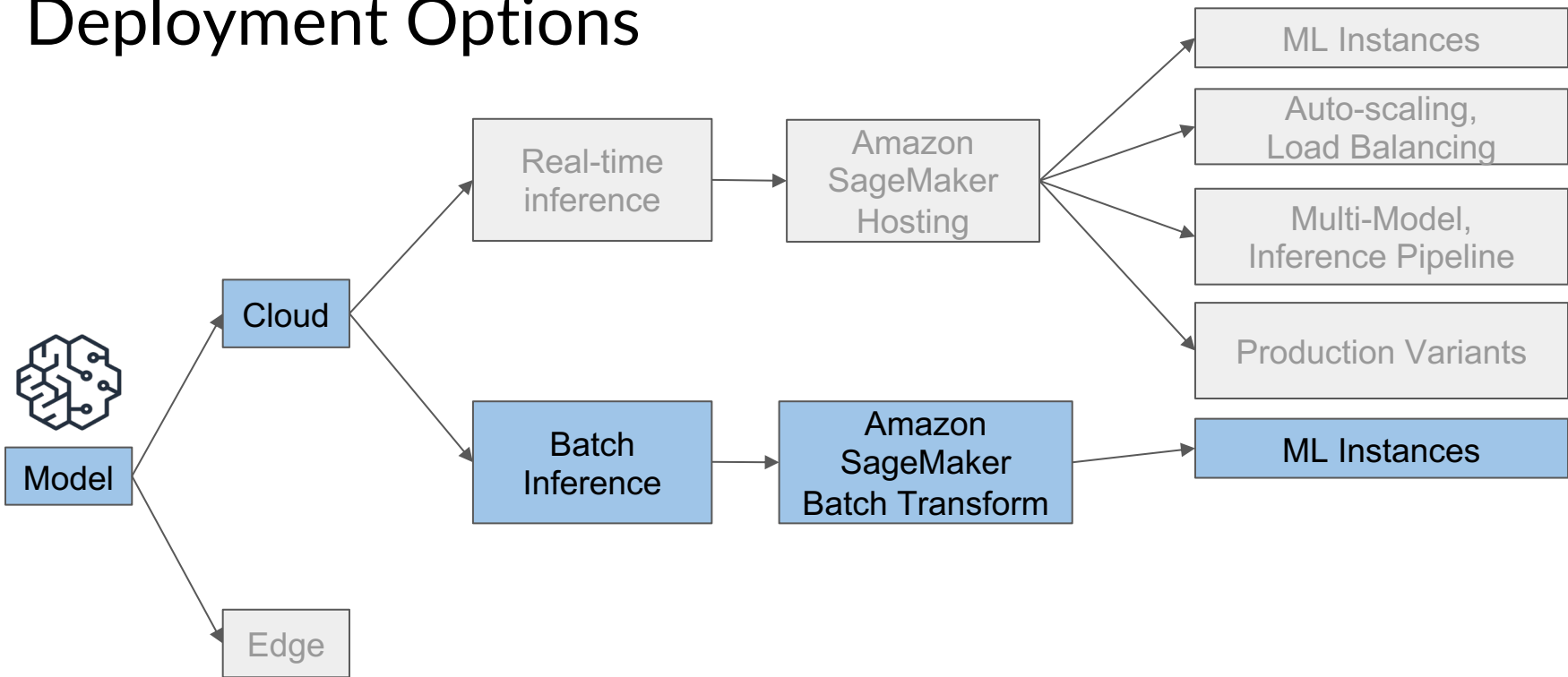
Batch Inference



Machine Learning Workflow



Deployment Options



Amazon SageMaker Batch Transform

Deploy Model For Batch Inference

```
create_model(  
    ...  
)
```

Package model
for
deployment

Amazon
Elastic Container
Registry
(ECR)



Amazon
S3



Amazon SageMaker
Batch Transform Job



Container



Inference
code



Model

ML Instance(s)



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(  
    ...  
)
```

Start
Batch
Transform Job



Amazon SageMaker
Batch Transform Job



Amazon
S3



Prediction
Request Data

Transient Compute



Container



Inference
code



Model

ML Instance(s)



Amazon SageMaker Batch Transform

Run Batch Transform Job For Batch Inference

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

Start
Batch
Transform Job



Amazon SageMaker
Batch Transform Job



Amazon
S3



Prediction
Request Data



Amazon
S3



Prediction
Response Data

Transient Compute



Container



Inference
code

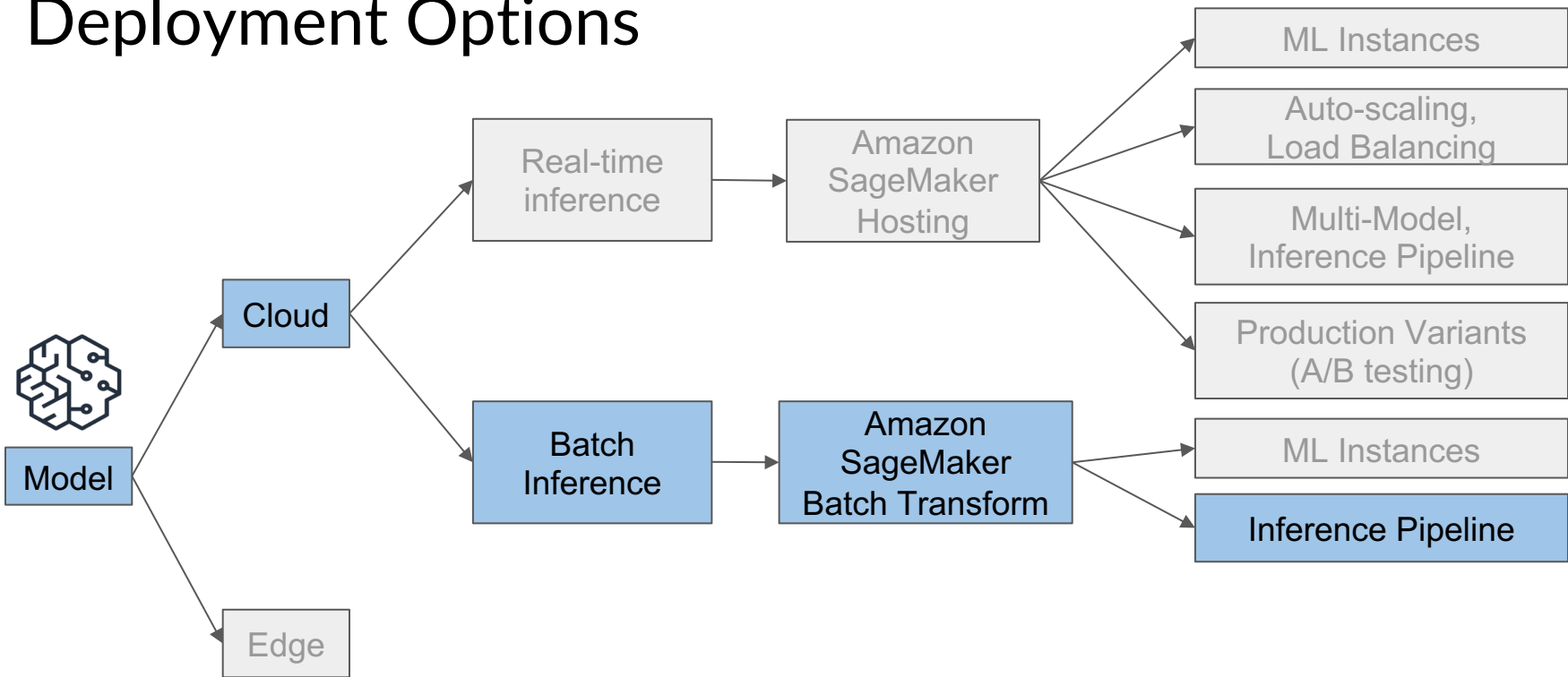


Model

ML Instance(s)

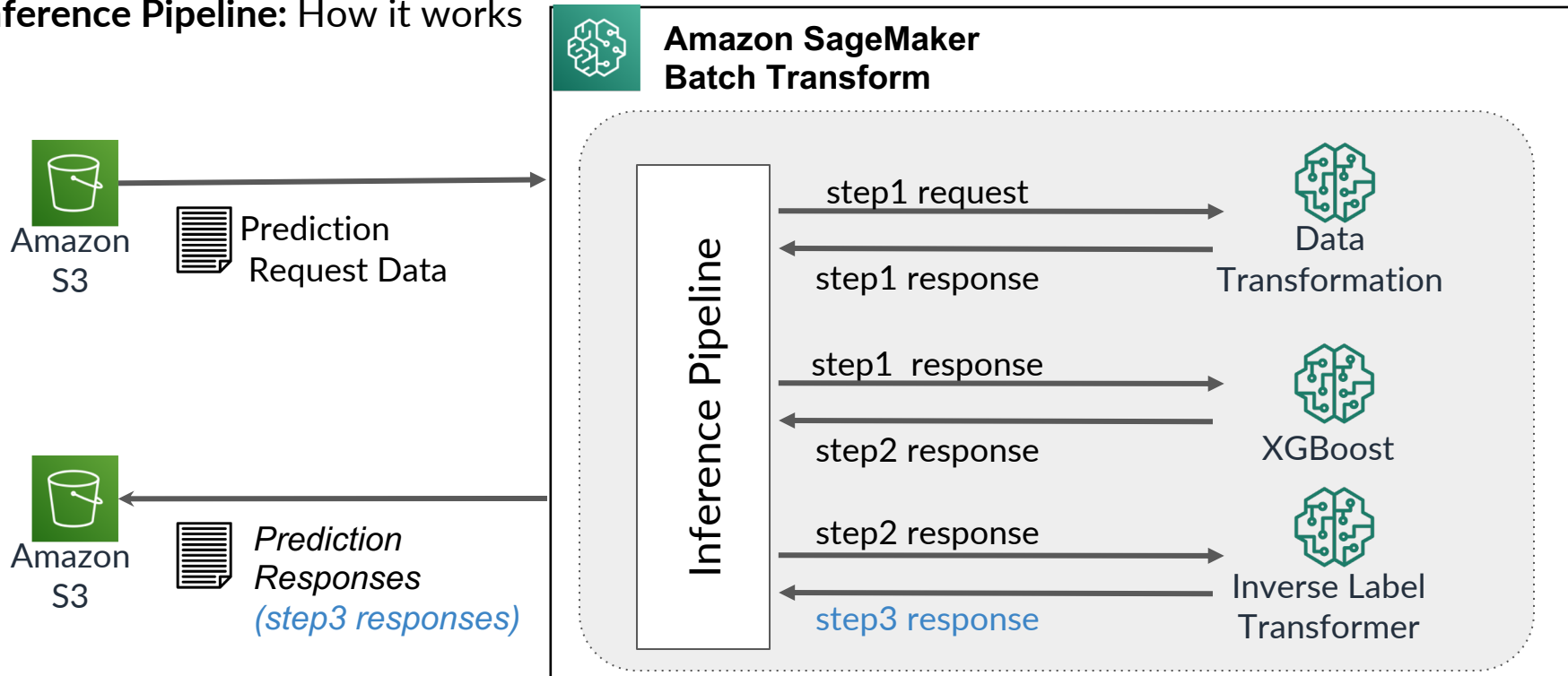


Deployment Options



Advanced Deployment Options

Inference Pipeline: How it works

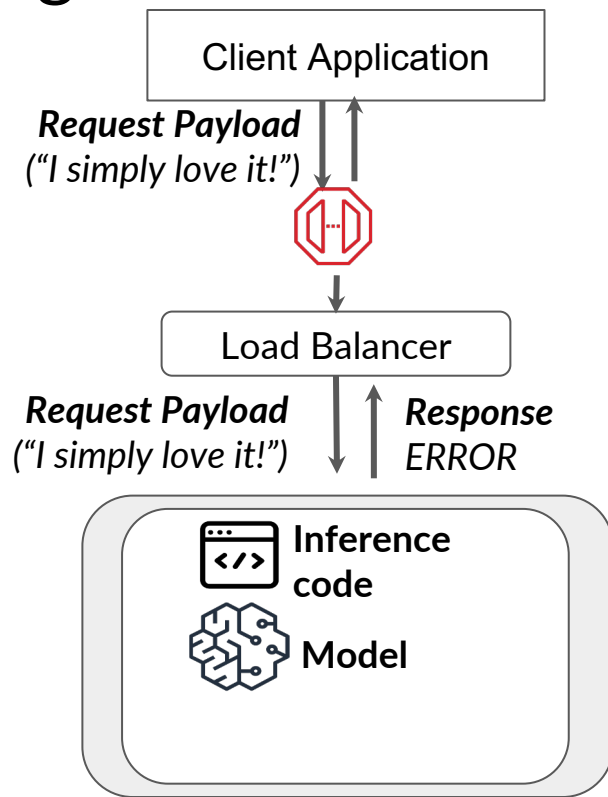


Model Integration



Amazon SageMaker Model Integration

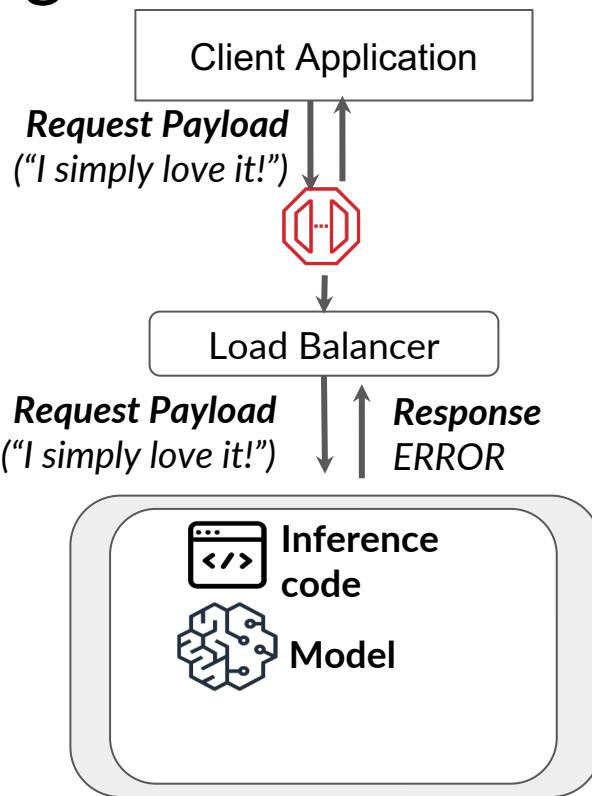
Integrating Models with ML Applications



Amazon SageMaker Model Integration

Integrating Models with ML Applications

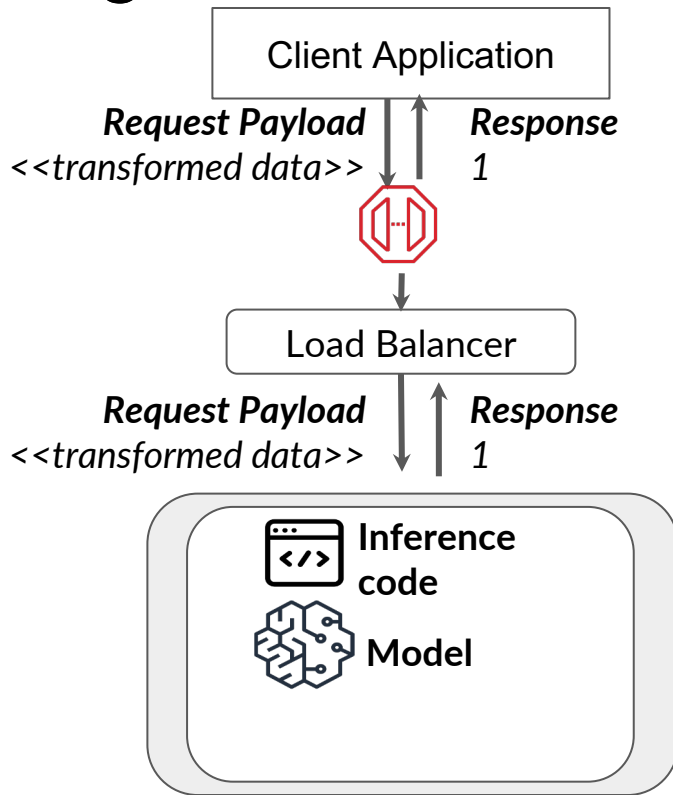
Need to apply the
same data
transformations used
during training



Amazon SageMaker Model Integration

Prepare Data for Inference in Client Application

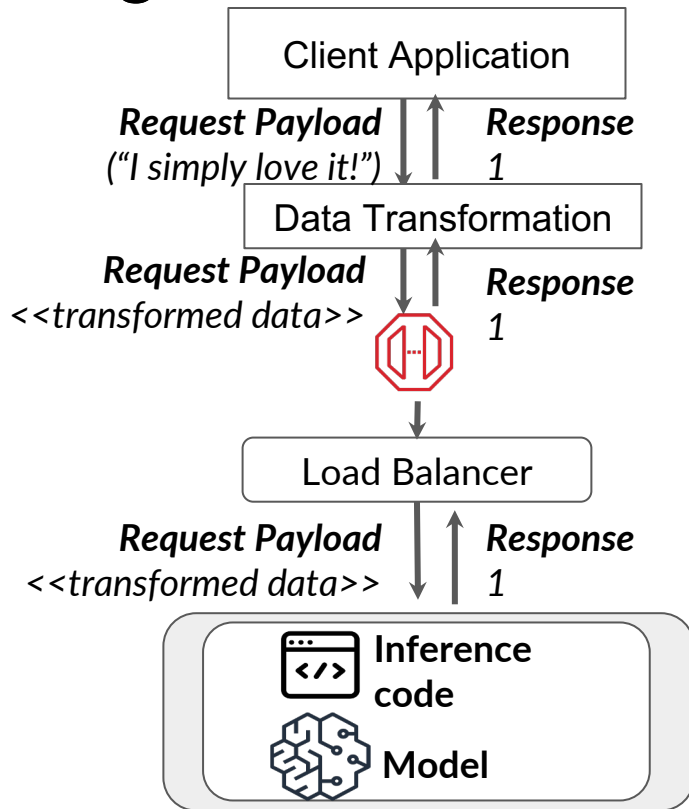
- **Implement data transformations in Client Application**
 - **Challenge:** Difficult to scale & manage
 - **Consideration:** Response may need to be transformed (1 = Positive)



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

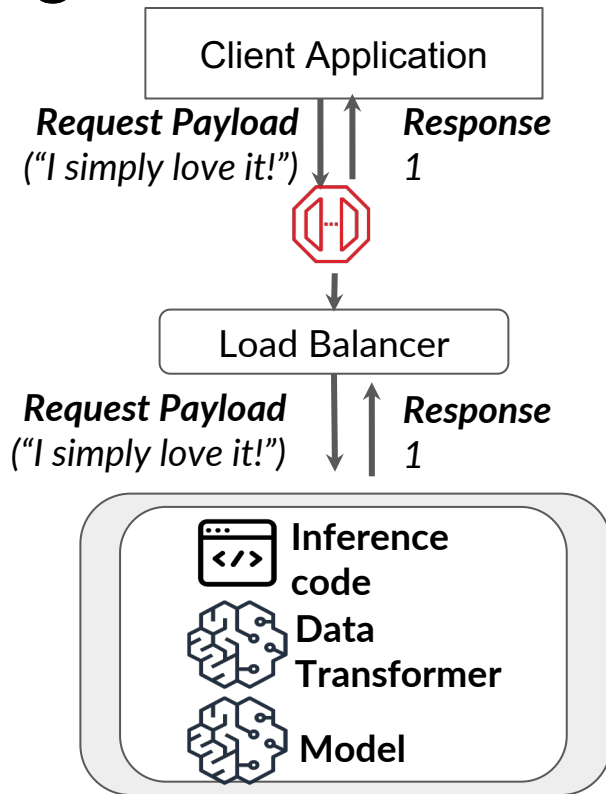


Amazon SageMaker Model Integration

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)

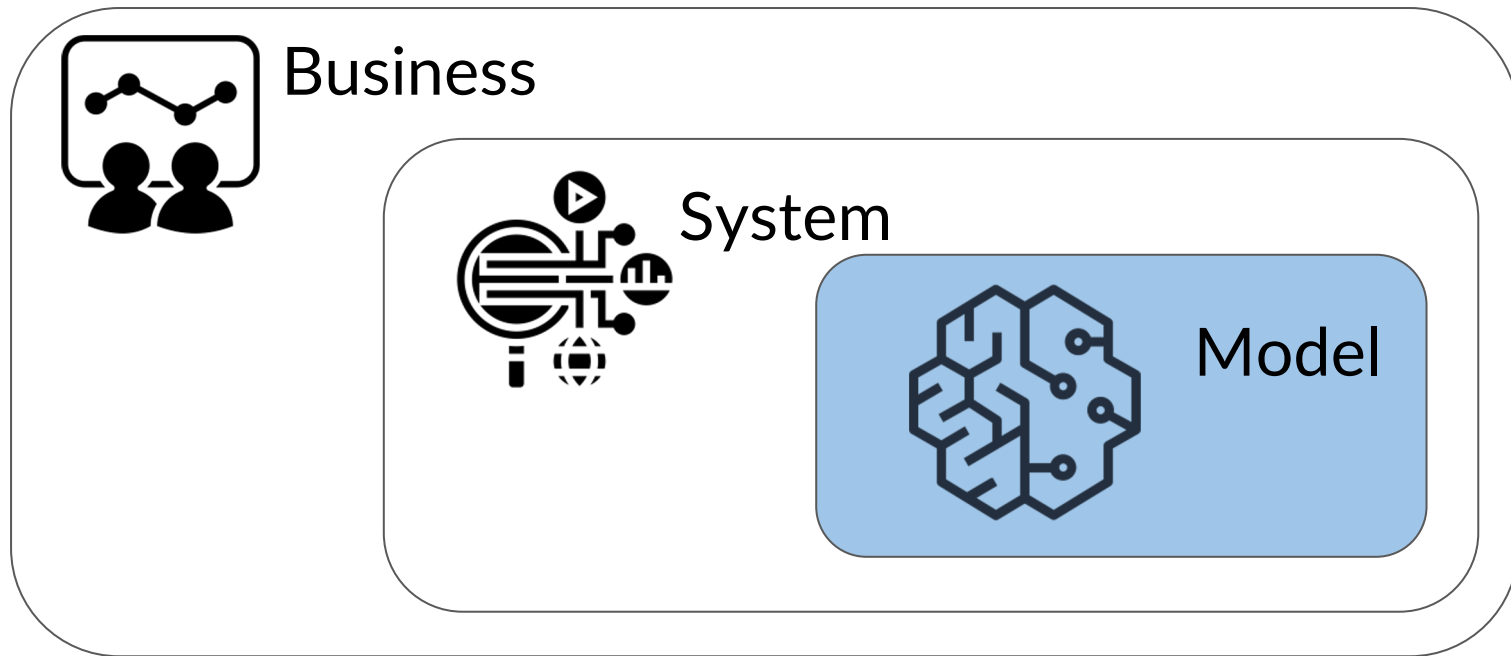


Monitoring ML Workloads



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

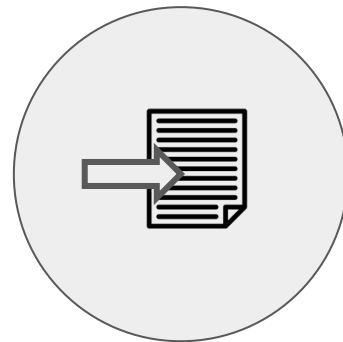
Examples



Customer behavior change
*Ex. Product change,
Demand change*



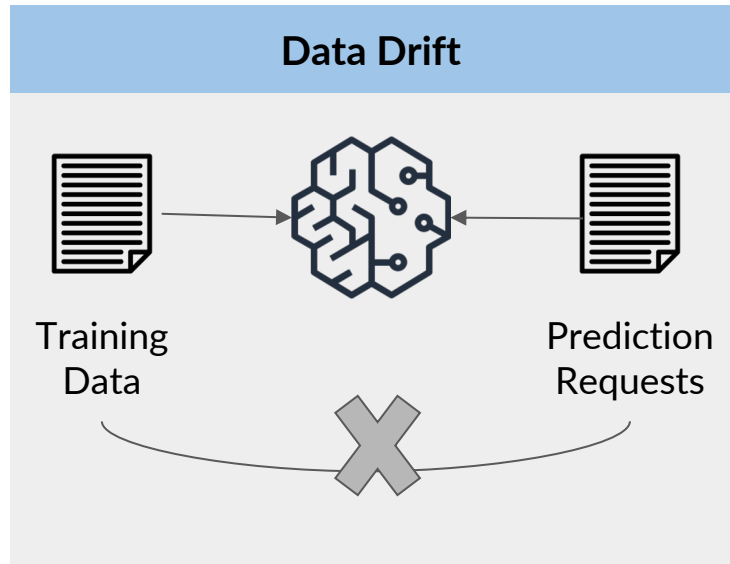
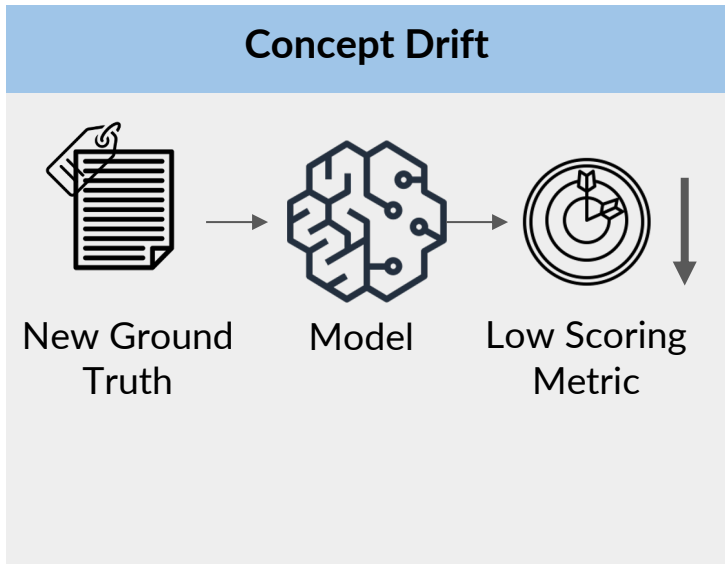
Changing business environment
Ex. New products



Changing data pipeline
Ex. Feature data suddenly missing

Monitoring Machine Learning Workloads

Model Monitoring



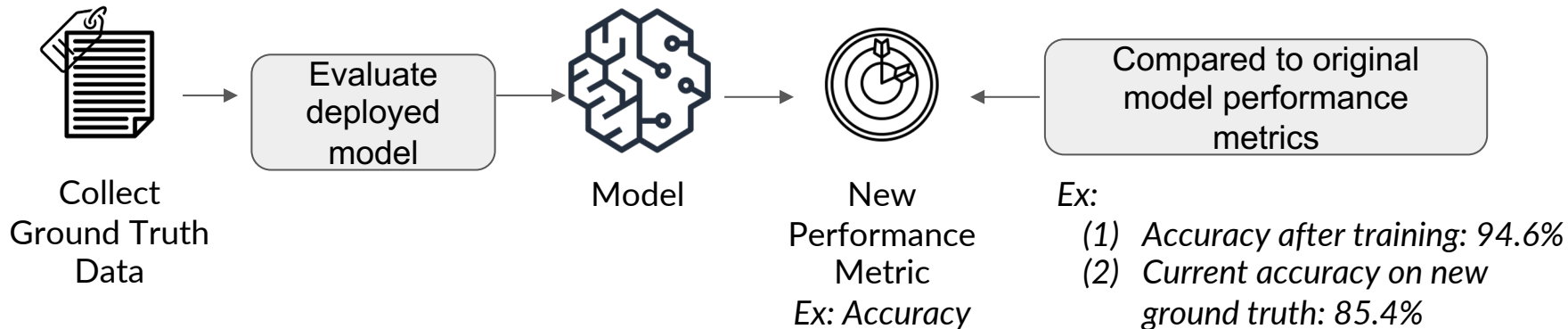
Monitoring Machine Learning Workloads

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

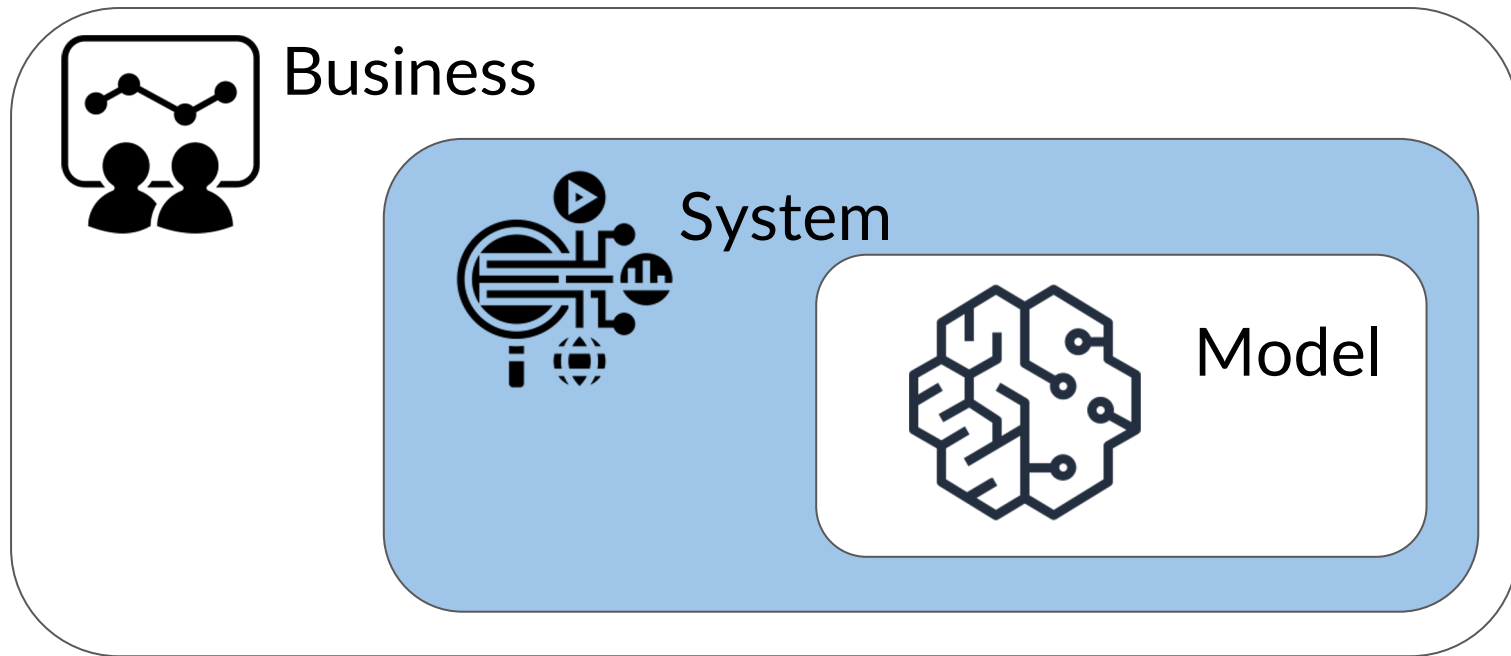
Methods to detect:

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

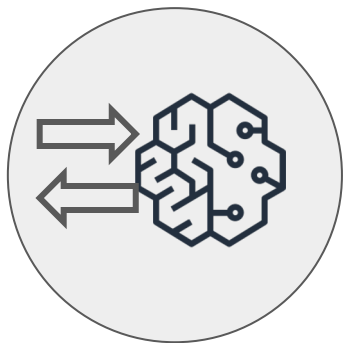


Monitoring Machine Learning Workloads

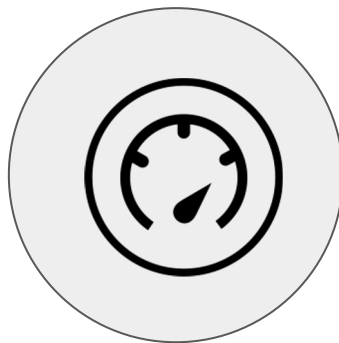
System Monitoring

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

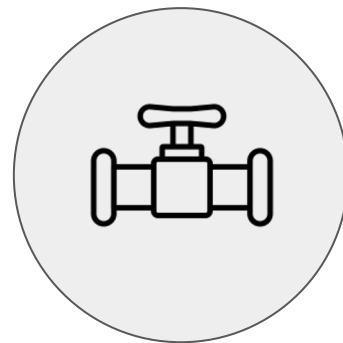
Examples



Model Latency



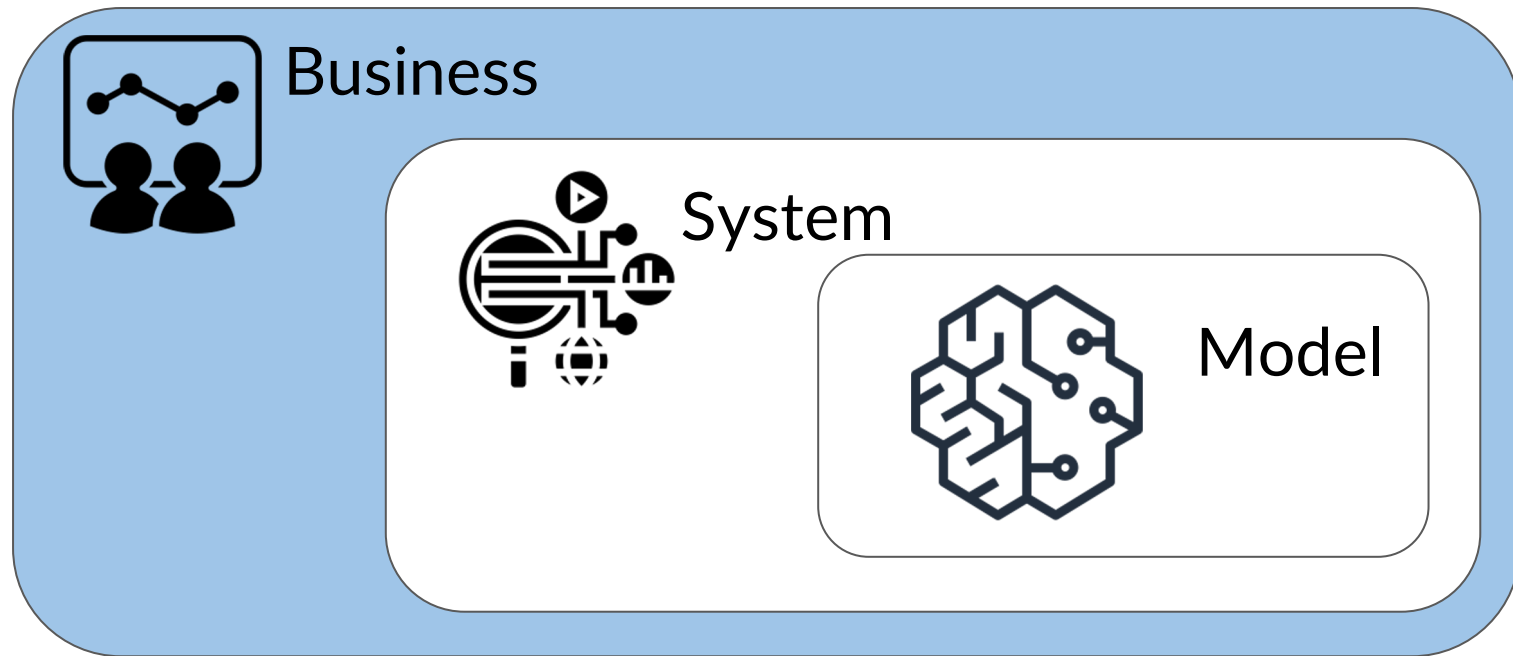
System Metrics



ML Pipeline

Monitoring Machine Learning Workloads

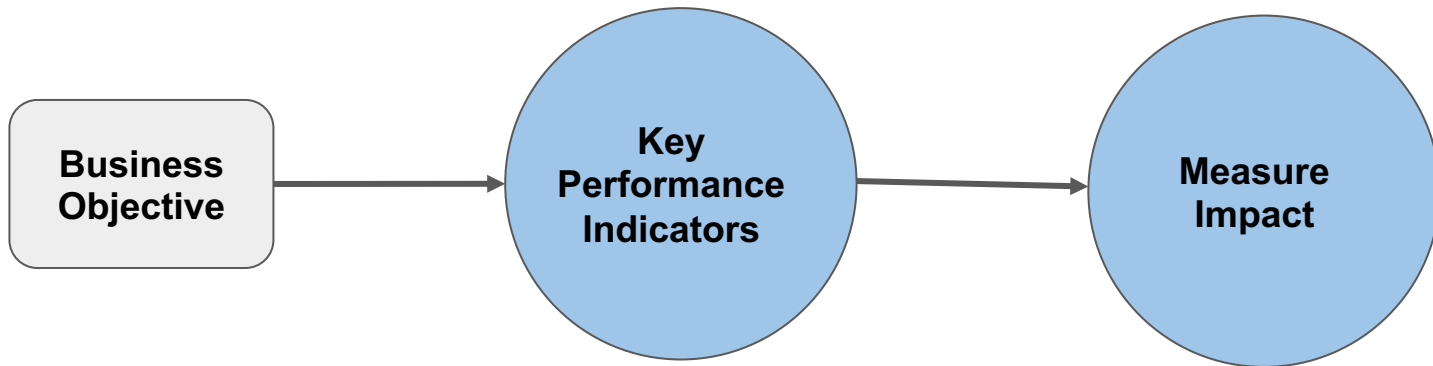
Considerations



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

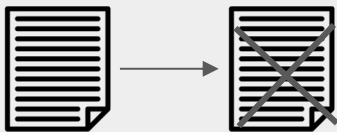


Amazon SageMaker Model Monitor

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



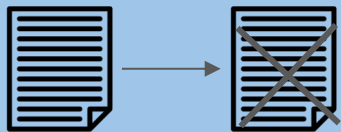
Monitor drift in feature attribution

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

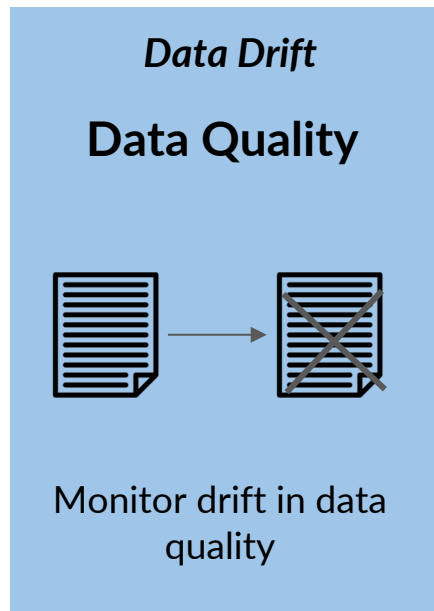
Feature Attribution Drift



Monitor drift in feature attribution

Amazon SageMaker Model Monitor

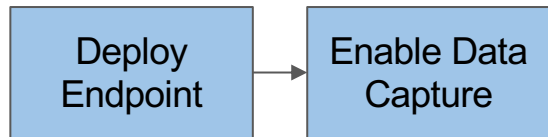
Monitor Type: Data Quality Monitor



- 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

Monitor Type: Data Quality Monitor



Inference requests,
prediction results



Amazon
SageMaker
Endpoint

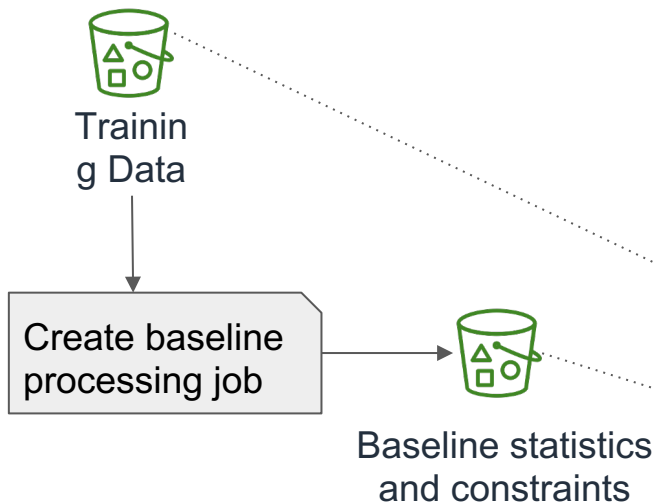
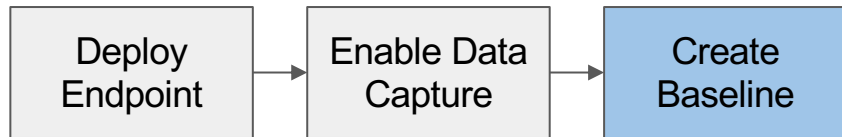


```
data_capture_config=DataCaptureConfig(  
    enable_capture = True,  
    sampling_percentage=100,  
    destination_s3_uri=s3_capture_upload_path)  
  
predictor = model.deploy(initial_instance_count=1,  
    instance_type='ml.m4.xlarge',  
    endpoint_name=endpoint_name,  
    data_capture_config=data_capture_config)
```

% of data to
capture

Amazon SageMaker Model Monitor

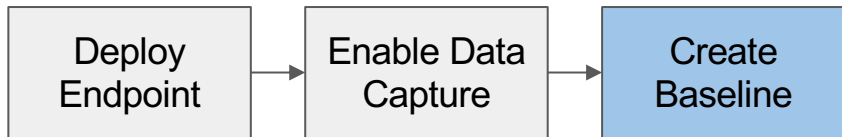
Monitor Type: Data Quality Monitor



```
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



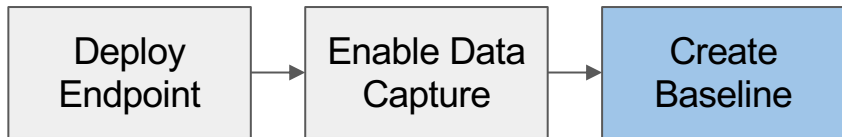
Baseline statistics
and constraints

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



Baseline statistics
and constraints

statistics.json

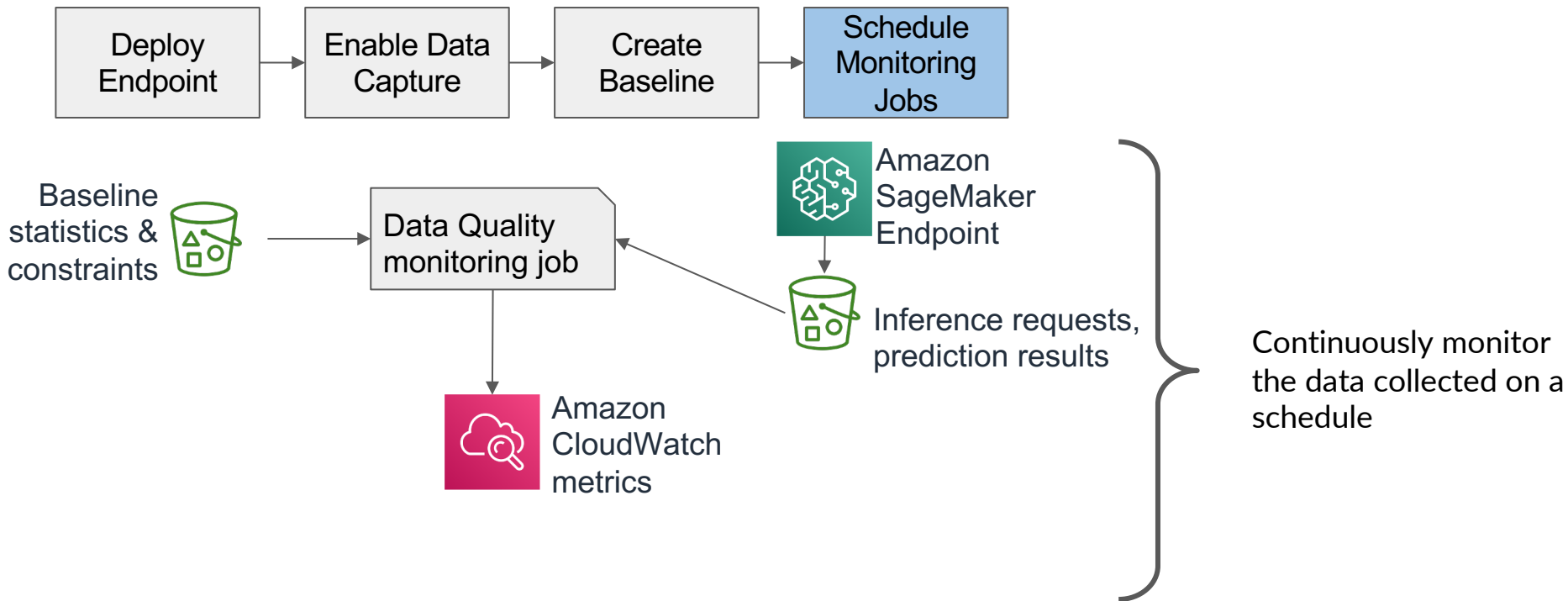
- Columnar statistics for each feature
- Examples:
 - Numeric → missing values, mean, min, max, distribution
 - String → missing values, distinct values, categorical distribution

constraints.json

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

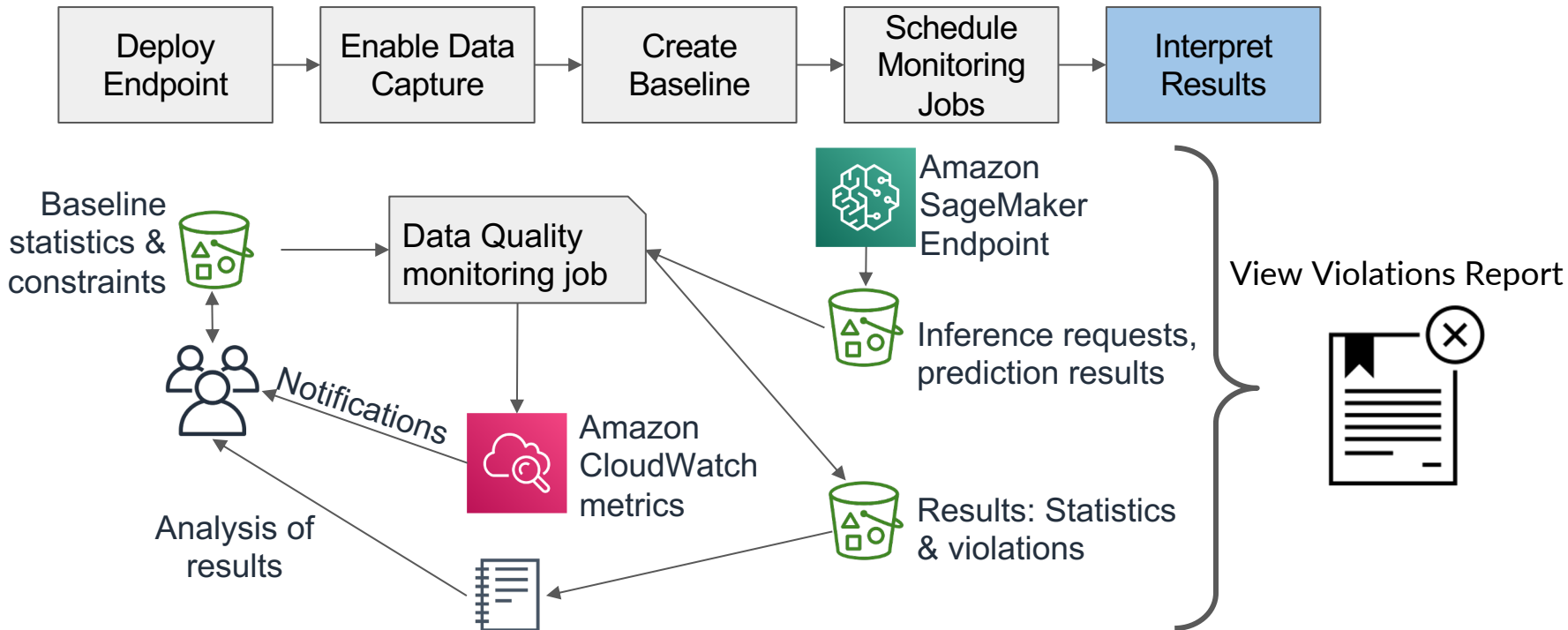
Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor



Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor

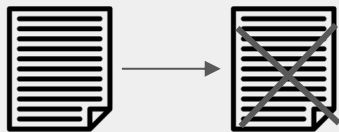


Amazon SageMaker Model Monitor

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 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



Training
Data



Model
Metric
Ex. Accuracy

VS



Ground
Truth Data



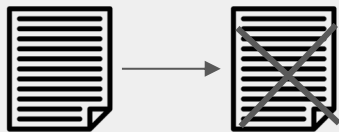
Model
Metric
Ex. Accuracy

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

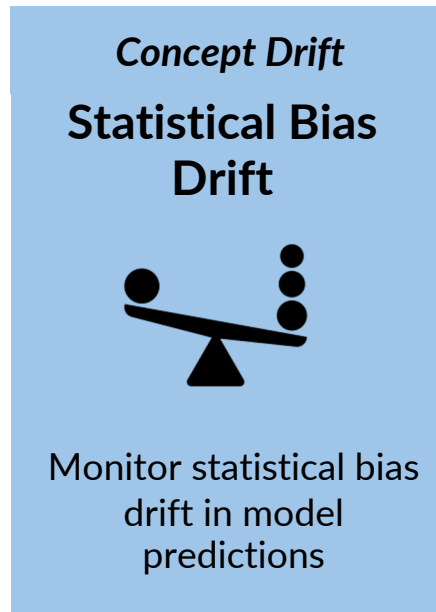
Feature Attribution Drift



Monitor drift in feature attribution

Amazon SageMaker Model Monitor

Monitor Type: Statistical Bias Drift



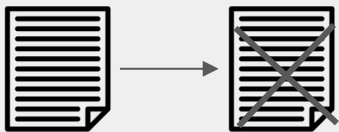
- Monitor predictions for statistical bias
- Amazon SageMaker Clarify integrates with Amazon SageMaker Model Monitor to detect statistical bias drift

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

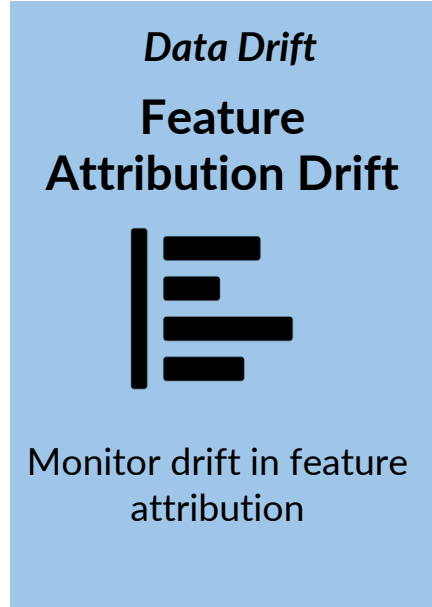
Feature Attribution Drift



Monitor drift in feature attribution

Amazon SageMaker Model Monitor

Monitor Type: Feature Attribution Drift



- 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