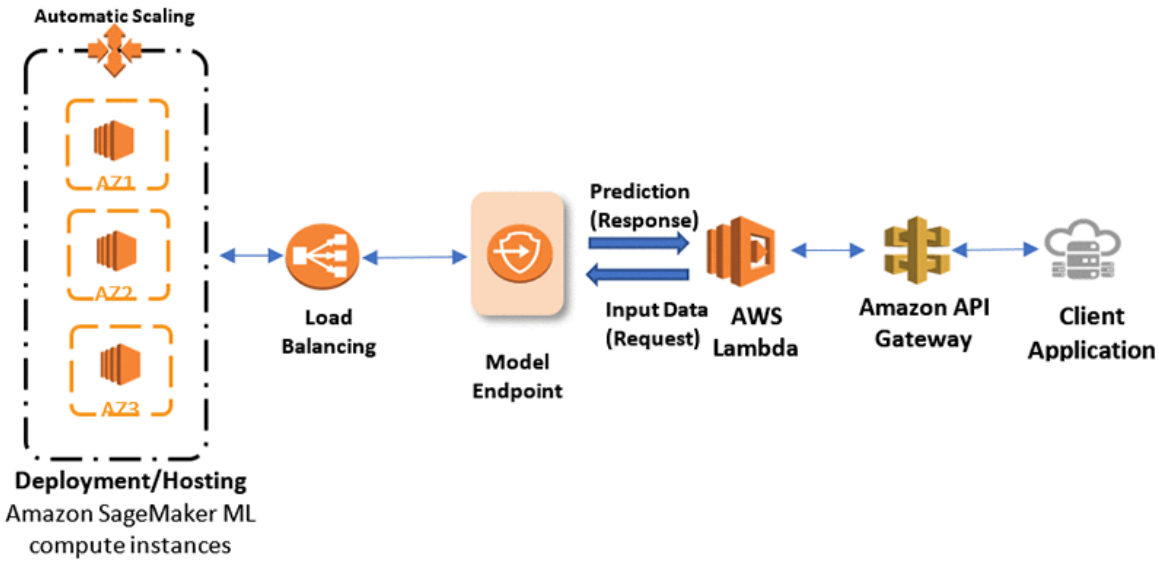
Deployment strategy

The main component of the process of putting the model to work depends on the usage of AWS tools like the sage maker and other components like S3 buckets, API Gateway, and AWS Lambda for deployment and scaling purposes.

The basic strategy for deploying the model in real-time is defined as



Here the hosting or deployment compute instances are several ec2 instances that are distributed over the regions to have high availability so with this we use sagemaker which provides a common platform for deployment

Here as per our need, I have 2 models to deploy for my project keeping it all real-time inference and having a single endpoint API for better scaling and load balancing

But if I had to solve the problem of only sentiment analysis then this architecture is perfectly fine in providing sub-second latency solutions for fast and reliable inference. Here the sagemaker deploys the model in the compute instance to describe in detail.

Through sageMaker hosting we can create endpoints, these are persistent endpoints that can be used for real-time inference. They can be used to serve your models for predictions in real-time with low latency. Serving the predictions in real-time requires a model serving stack that typically includes a web server that can interact code and model. The model can then be consumed by client applications through real-time invoke API requests.

The 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.

SageMaker has three basic scenarios for deployment when you use it to train and deploy your model. You can use prebuilt code, prebuilt serving containers, or a mixture of the two.

In the first option, we use both prebuilt inference codes combined with a prebuilt serving container. The container includes the web proxy and the serving stack combined with the code that's needed to load and serve your model for real-time predictions. This scenario would be valid for some of the SageMaker built-in algorithms where I need only a trained model and the configuration for how I want to host that machine learning instance behind that endpoint. though it’s very straightforward our case is different. Here we have both trained model and inference code

So the other option is bringing your container image and inference code for hosting a model on a SageMaker endpoint. In this case, we have some additional work to do by creating a container that's compatible with SageMaker for inference. But this also offers the flexibility to choose and customize the underlying container that's hosting the model.

We typically want to use smaller instances and more than one machine learning instance. And once your endpoints are deployed, we have to ensure that we canscale up and down to meet the demands. This is where autoscaling comes in. It allows you to scale the number of machine learning instances that are hosting your endpoints up or down based on your workload demands.

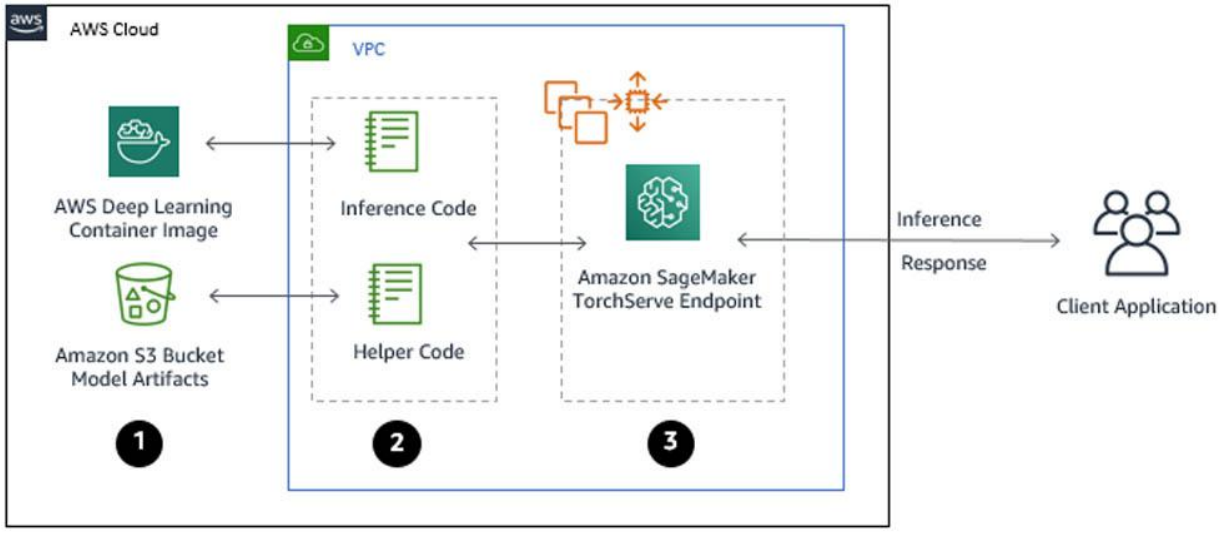
This is important to meet the demands of your workload, which means that you can increase the number of instances that serve your model when you reach a threshold for capacity that you've established. This is also important for cost optimization for two reasons. Not only can you scale your instances up to meet the higher workload demands when we need it, but we can also scale it back down to a lower level of computing when it is no longer needed.

When you deploy your endpoint, the machine learning instances that back that implant will emit several metrics to Amazon CloudWatch. For those that are unfamiliar with it, CloudWatch is the managed AWS service for monitoring your AWS resources. SageMaker emits several metrics about 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.

Now we can start with multi-model endpoints. Until now, we saw about SageMaker endpoints that serve predictions for one model. However, we can also host multiple models behind a single endpoint. Which is our main goal as we have two models with different architectures. SageMaker dynamically loads your models when you invoke them.

All of the models that are hosted on a multi-modal endpoint must share the same serving container image. Multi-model endpoints are an option that can improve endpoint utilization when your models are of similar size and share the same container image and have similar invocation latency requirements.



Here is an example where we can use a multimodal endpoint with its model artifacts in S3 and docker images. These endpoints are passed on to lambda and hence its gets autoscaled with requirements perspective using AWS Lambda.

Here, we'll see another feature called inference pipeline. The inference pipeline allows us to host multiple models behind a single endpoint. But in this case, the models are a sequential chain of models with the steps that are required for inference. This allows you to take our aspect prediction model, sentiment model, and host them so they can be sequentially run behind a single endpoint.

the inference request comes into the endpoint, then the first model is invoked, and that model is the aspect prediction model. The output of that model is then passed to the next step, which is the sentiment model. That output is then passed to the next step, where ultimately in that final step in the pipeline, it provides the final response. Which could be saved into the results S3 bucket.

For reference visit
https://aws.amazon.com/blogs/machine-learning/using-amazon-sagemaker-inference-pipelines-with-multi-model-endpoints/

So to conclude production idea optimized for my approach

I would consider using the sagemaker with a customized model container and serving/inferencing scripts.

Secondary I would consider spinning this customized approach into fully multi-model deployment with a single endpoint using either MME architecture or inference pipelines that have sequential nature of execution

These responses are then passed to the load servers and lambda which handles the target auto-scaling policy as per the need of the requests and cloud watch tool of AWS.

There is then attached to the API gateway which allows for up to 10,000 requests per second. This API accepts GET If you receive a large burst of traffic, both API Gateway and Lambda will scale in response to the traffic.