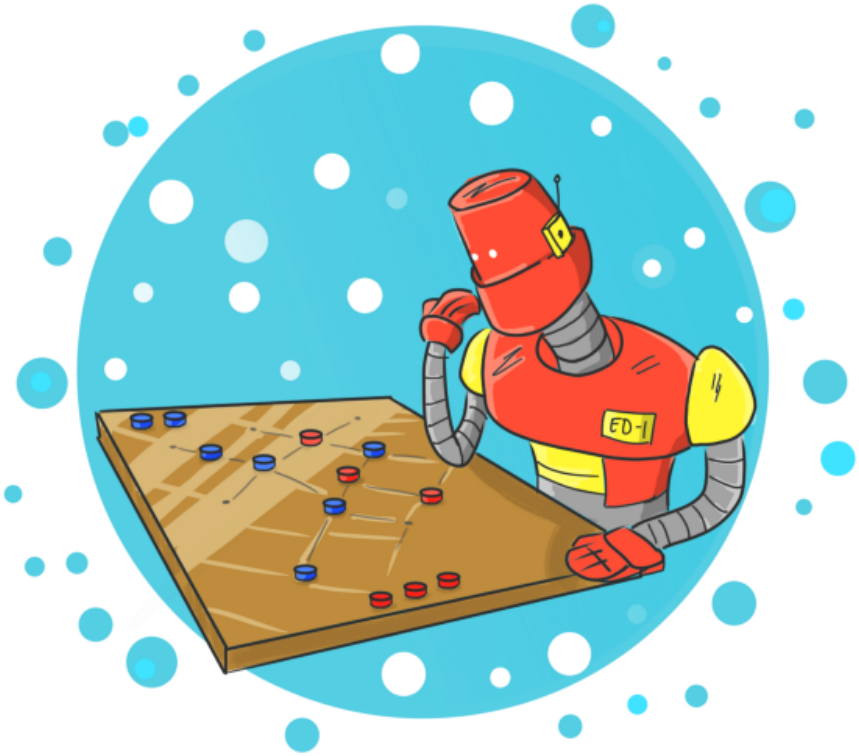
ML Model Deployment Strategies



Let’s Strategize (Image by Author)

An illustrated guide to deployment strategies for ML Engineers

Hello There!

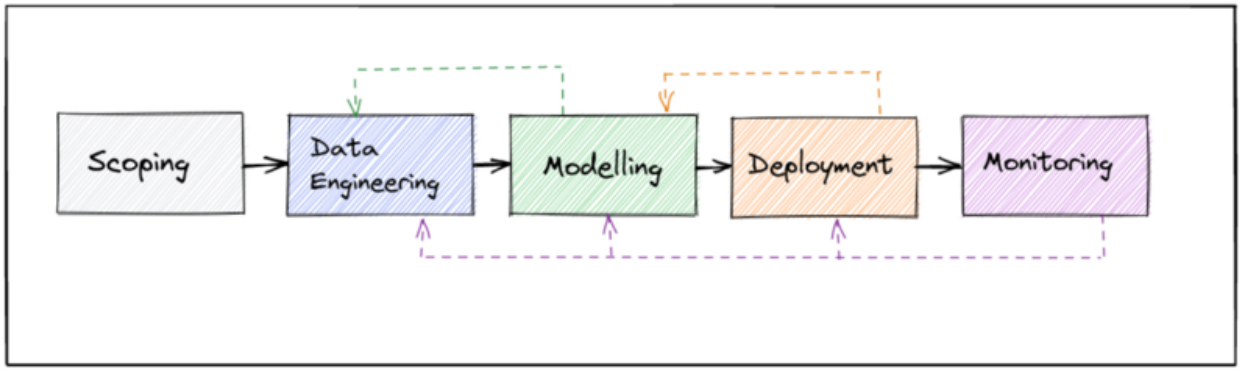
This article is for anyone who wants to understand how ML models are deployed in production and what strategies can be used in deploying those models.

I will illustrate the general approach to deploying ML models, different strategies that can be adopted for deploying, and where these are generally implemented. Each data science team will have a different set of requirements, so take it with a grain of salt.

Let’s begin.

Understanding ML model deployment

Unlike software or application deployment, model deployment is a different beast. A simple ML model lifecycle would have stages like Scoping, Data Collection, Data Engineering, Model Training, Model Validation, Deployment, and Monitoring.



ML Lifecycle (Image by Author)

When we are deploying the ML model, we need to consider some factors like:

1. Model size and packaging—Model size plays a massive role in how we plan to package it. Smaller models can generally be wrapped in a FastAPI server and containerised in a Docker container. Larger models however might require to be loaded during deployment—they may be pulled from a remote storage and run via model servers such as TFServing or TorchServer.

2. Model retraining and versioning— How often you retrain your model impacts your deployment strategy. Do you often need to compare your model performances? How often do you have to update your models in production? Will you be maintaining different versions of your model in production?

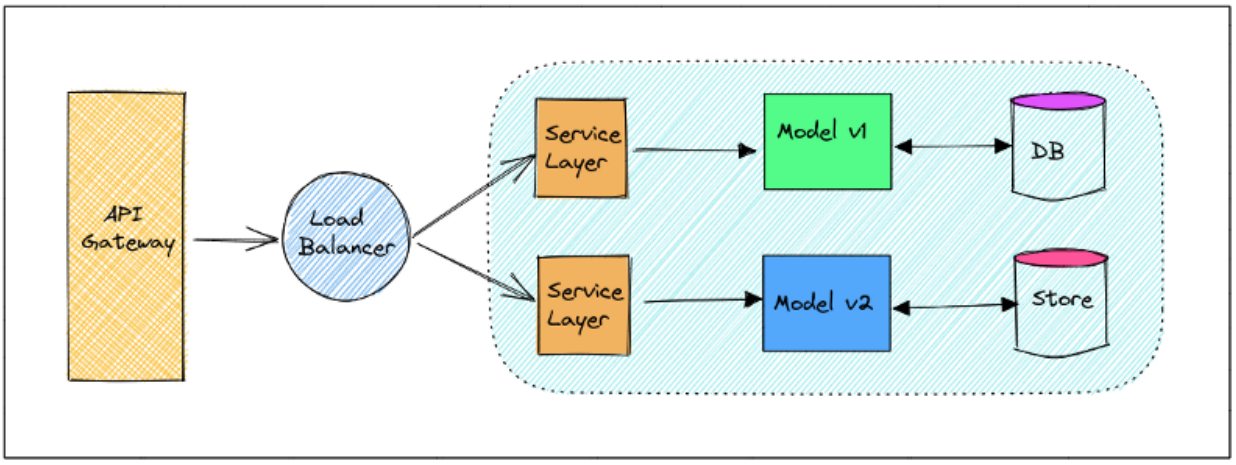
3. Traffic and requesting routing—Depending on the traffic and type of model, you would have to decide on either real-time inferencing or batch model deployment. How much traffic would you want to divert to each version of the model? How many users will have access to a specific model version?

4. Data and Concept drift—Over a span of time real-world data keeps changing, and might not be reflected in the model. Like, say, how buying power is related to salaries, which might change yearly or monthly. Or how the consumer buying pattern changed during the Covid-19 pandemic, but models mostly relied on historical data. This affects how our deployment architecture has to be designed: should we retrain and redeploy?

Should we only retrain and stage the model for now? This factor comes into play over the long-term deployment strategy of a data science team.

Keeping these factors in mind, we have about six common strategies for model deployment. These are mostly borrowed from DevOps and UX methodologies, applicable quite well in ML scenarios.

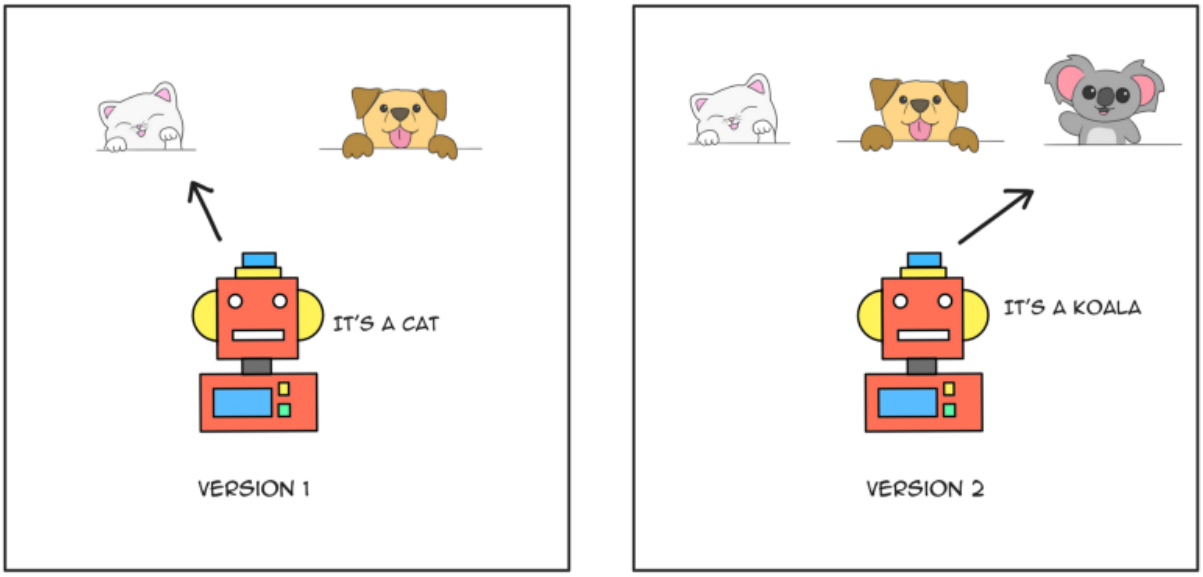
Usually, deployment of the model in production on a technical level involves an API endpoint gateway, a load balancer, a cluster of virtual machines, a service layer, some form of persistent data storage and the model itself.



Generic Model Deployment (Image by Author)

Deployment strategies are usually configured at the load balancer and service level, mostly configuring the routing and ingestion rules.

Let's take the example of an animal recognition and classification system. ***We begin with a simple Cat-Dog Classifier. This will be our version 1 of the model.*** Let's say we have trained a copy of the model to recognize Koalas as well, so ***our version 2 is a Cat-Dog-Koala Classifier***. How would we approach the deployment of the newer version of our model?



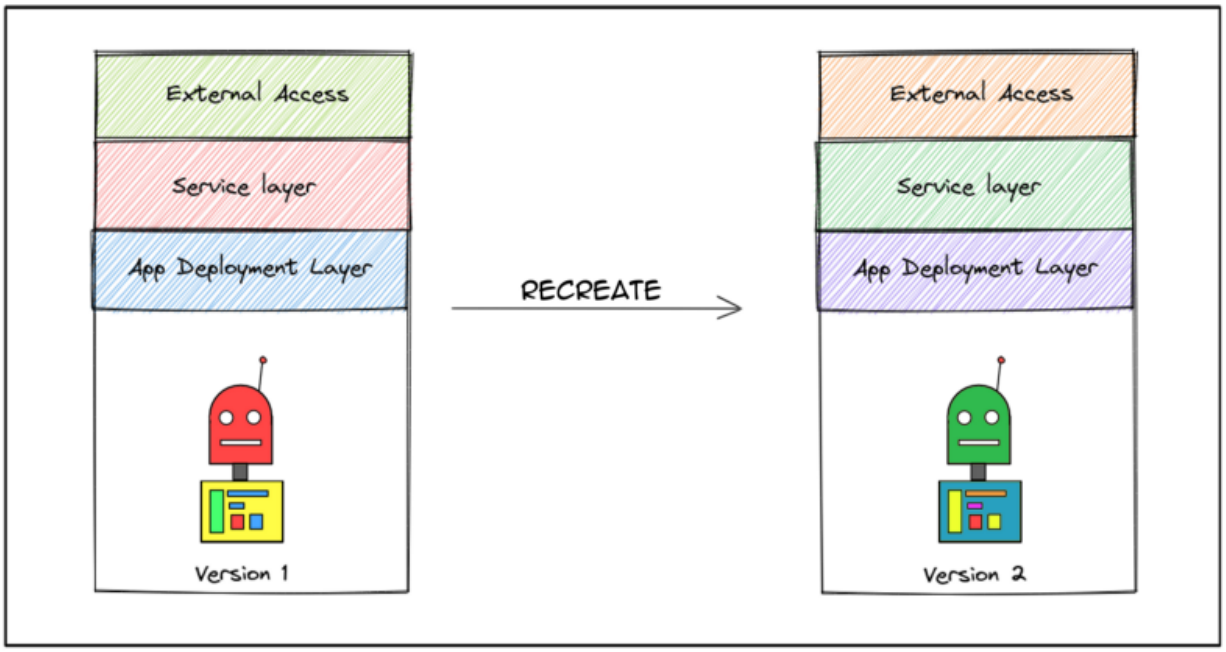
Model Version (Image By Author)

Model Deployment Strategies

Big Bang—Recreate

WHAT—This form of deployment is a “from scratch” style of deployment. You have to tear down the existing deployment for the new one to be deployed.

WHERE—Generally acceptable in development environments—you can recreate the deployments as many times as you wish with different configurations. Usually, a deployment pipeline tears down the existing resources and creates new updates in its place.



Recreate Deployment (Image by Author)

There is a certain amount of downtime involved with this kind of deployment. Not acceptable in our current pace of ML development in organizations.

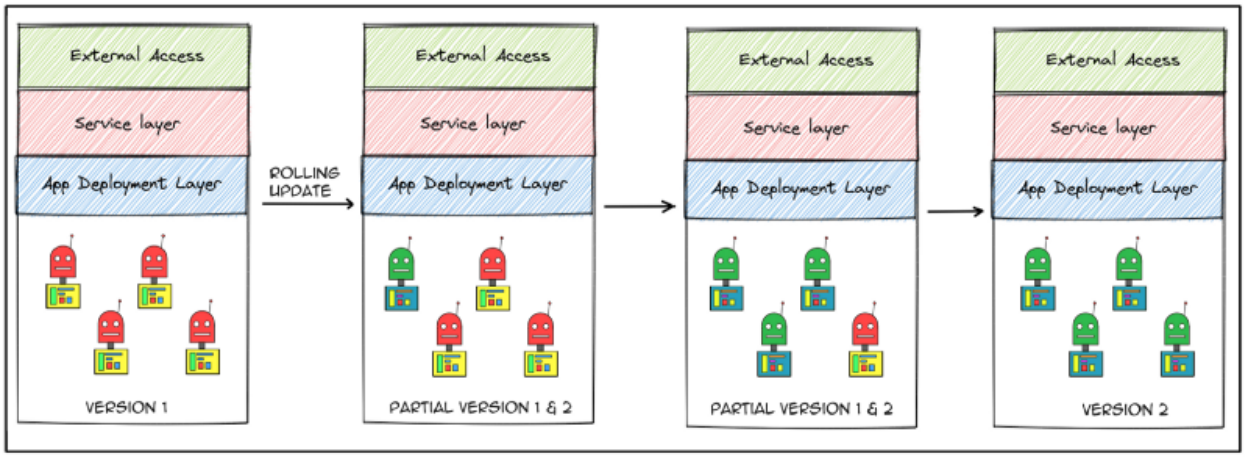
In our example, we would replace version 1 with version 2; this includes replacing all related infrastructure and library setups.

Rolling Updates

> They see me rolling…They hating…🎵

WHAT—Rolling updates involve updating all instances of your model/application one by one. Say if you have 4 pods of the application running currently and you activate a rolling update with a new version of your model. One by one the older pods are replaced with a newer ones. There is zero downtime in this approach.

WHERE—Useful when you want to make a quick update of your entire model line with a new version. Also allows you to roll back to an older version when needed. Mostly used in testing or staging environments where teams need to test out the new versions of the models.



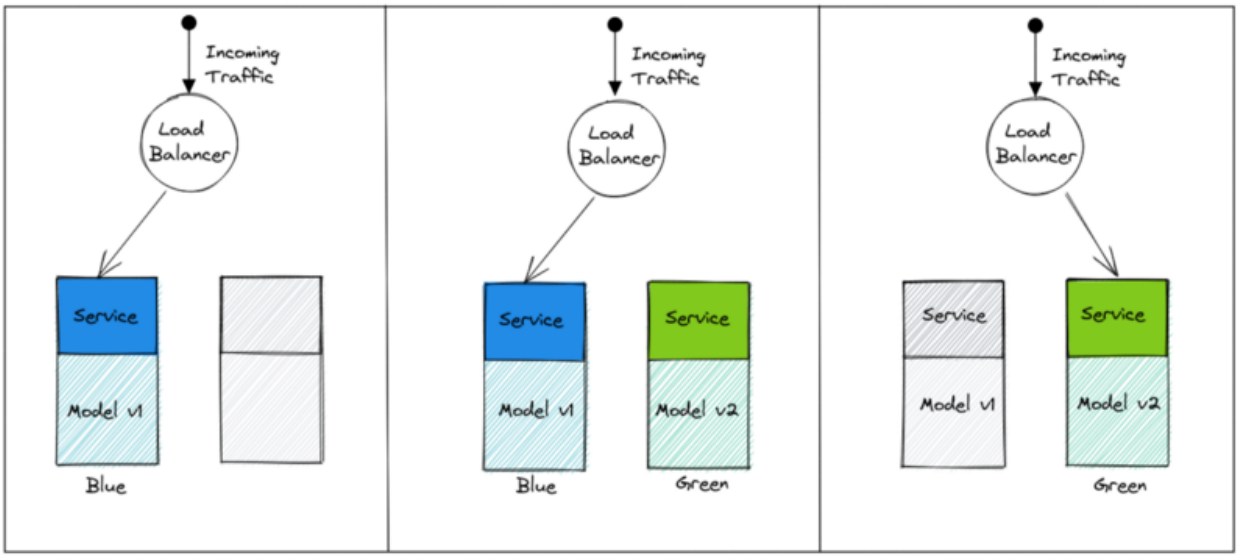
Rolling Update (Image by Author)

It generally won't be the sole implementation in production systems, unless you need to deploy only a specific version of the model across the board. In our example we would replace only the model application pod, keeping the rest of the infrastructure as it is.

🔵Blue/Green🟢

WHAT—This form of deployment is essentially a server swap. There are two identical systems available. The user requests are routed to one of the systems, and the newer changes and updates are done on the other system. Once the updates are tested and verified, the user requests are routed to the newer system, essentially swapping out the old model for a new one.

WHERE—Mostly employed in application or web app scenarios. Can be applied for model deployment as well, in both batch and real-time inferencing deployments. There is essentially zero downtime as we are just pointing the load balancer to a different set of machines.



Blue-Green Deployment (Image by Author)

As you can see here, we create a new identical system with an updated version of the model and then just switch the traffic to the new system.

However, we will have to factor in the cost of maintaining two identical infrastructure systems. Choose this method depending on the scale and affordability of the infrastructure.

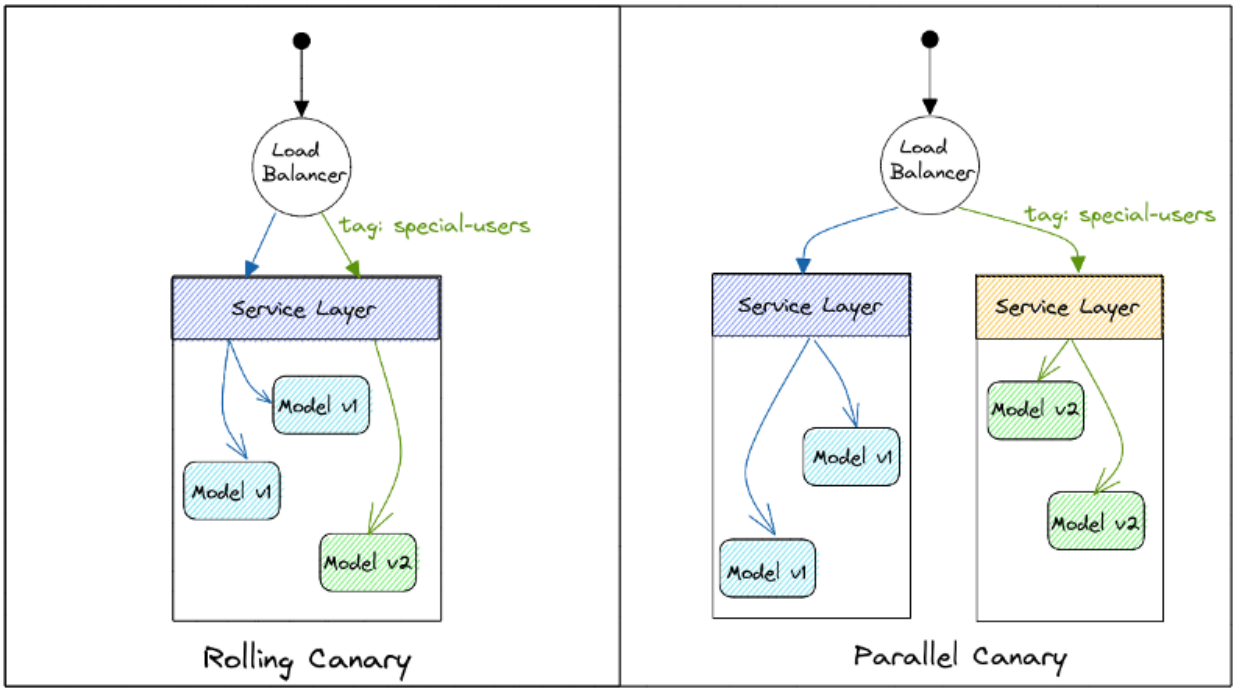
Canary🐦

WHAT—In Canary Deployment, we deploy the update to our existing system and expose the users partially to the new version. This means a small percentage of our users will be able to access the updated model and the rest will still use the old version.

This style of deployment is used to mainly test how well the new version is working or if it is working at all. Usually, a small percentage of users (around 5%–30%) will be exposed to the update, as this helps the ML engineers and developers to understand which features might be needed to roll out and which ones to be refactored.

WHERE—Canary Deployment is usually performed in staging and production when the teams need to understand the performance of the new updated models. This can be done in two ways:

1. Canary Rolling Deployment
2. Canary Parallel Deployment



Canary Deployment (Image by Author)

Rolling Deployment patches update to a small number of instances within the same cluster and a set of user requests are sent to these pods.

Parallel Deployment creates a smaller set of new instances alongside the existing setup and sends a percentage of user requests to these pods.

Depending on the traffic load and infra availability, you can choose which Canary implementation you wish to set up.

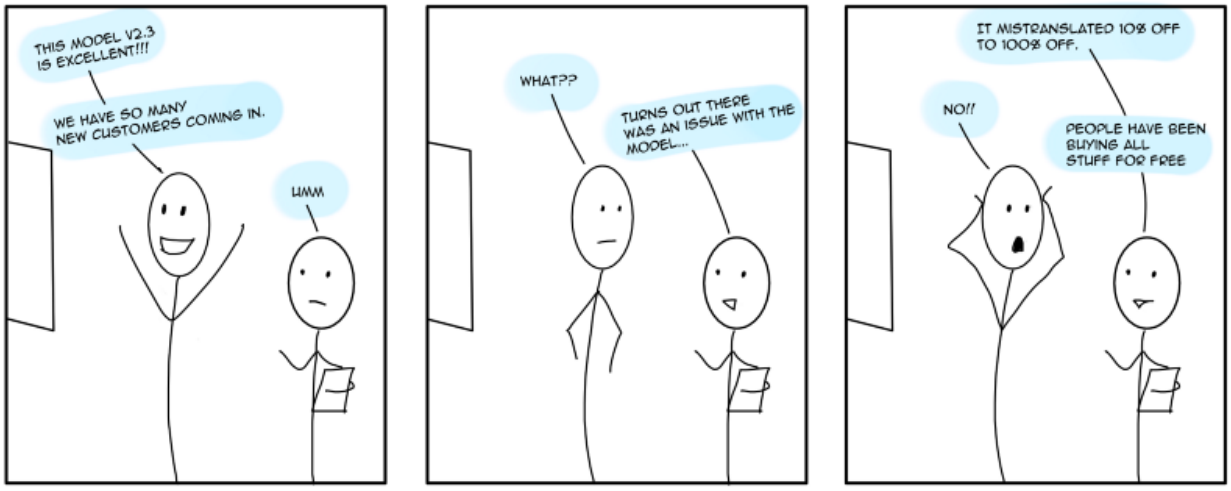
The user requests are usually tagged via headers and then the load balancer settings are configured to send them to their appropriate destination. Meaning ***a set of users are chosen*** to view the updates, and ***the same set of users will see the updates every time.*** User requests aren't randomly sent to the new pods. Canary Deployments have session affinity.

In our example, let's say 10% of select users can submit their image to the model and it would classify them with the Koala option, the rest of the users can only use the binary classifier.

A/B Testing

WHAT—This methodology is most used in User Experience research, to evaluate what the users prefer. In the ML scenario, we can use this style of deployment to understand what users prefer and which model might work better for them.

WHERE—Mostly employed in the deployment of recommendation systems worldwide. Depending on the demographics, say an online market site employs two different types of recommendation engines, one might serve a general set of users and one serves a specific geographic locality—with more native language support. The engineers can determine after a length of time, which engine gives a smoother experience for the users.



Why we need A/B Testing (Image by Author)

In our example, say we deploy the Cat-Dog classifier globally, but we deploy versions 1 and 2 in Australia—Pacific Islands region, where user requests are randomly sent to version 2. Imagine retraining version 2 to recognize the more local breeds of animals and deploying it, which version do you think the people in Australia will prefer?

```
NOTE:
You might wonder what's thedifference between Canary and A/B Testing. The main differences are:
```

```
- Canary is session affinity-based, mostly the same set of users will see the updated model whereas, in A/B Testing, the users are randomly sent to different versions.
```

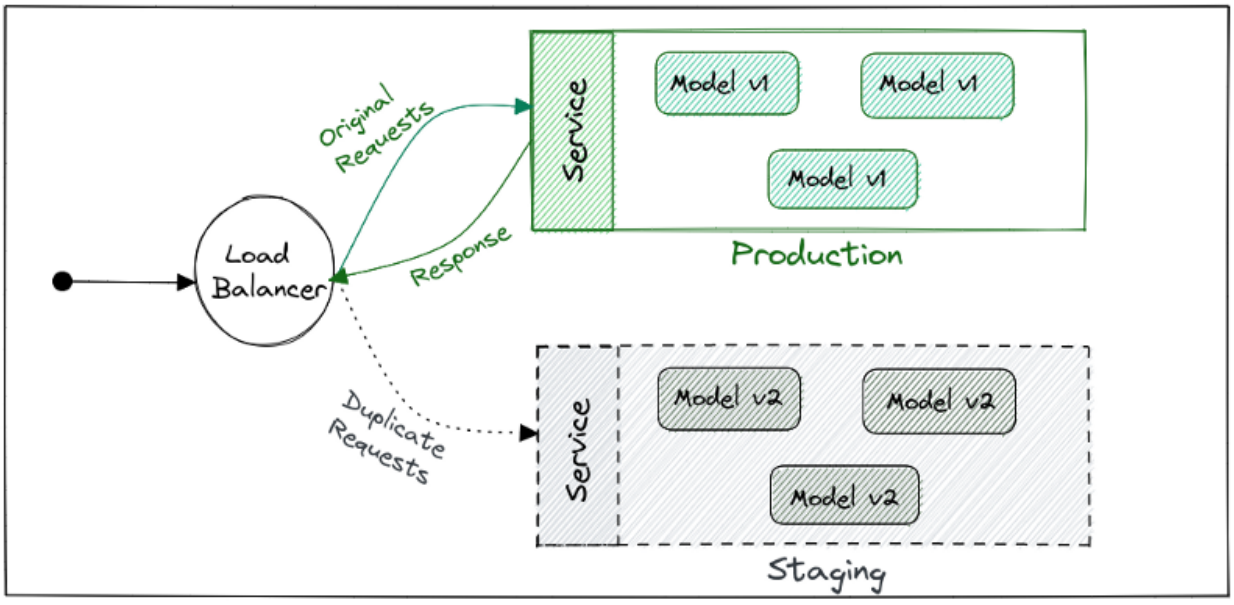
```
- Canary is specifically to test if the app or model is working as expected, A/B is more t
o understanding user experience.
```

```
- The Canary user ratio never goes beyond 50, a very small percentage of the user request
 (less than 35% ideally) is sent to the newer testing version.
```

Shadow👻

WHAT—Shadow Deployment is used in production systems to test a newer version of models with production data. A copy of the user request is made and sent to your updated model, but the existing system gives the response.

WHERE—Say you have a production system with high traffic, and to verify how your updated model might handle production data and traffic load, we deploy the updated model in a shadow mode. Every time a request is sent to the model, a copy is sent to the updated version. The response is only sent back by the existing model, not the updated one.



Shadow Deployment (Image by Author)

This type of deployment is used to understand the production load, traffic, and model performance. Primarily used in high-volume prediction services. The architectural and

design complexity increases with this kind of deployment. We have to include service meshes, request routing and complex database designs based on the use case.

In our example, we might deploy version 2 as shadow deployment to understand how version 2 might handle production load, also understand from where are we getting more koala classification requests 😉 or any specific kind of request pattern for the model.

Now we have a good basic understanding of how the models might be deployed in various use-cases. Depending on the requirements, data science teams might use a combination of these approaches. To each their own.

That’s all for now. Take care. Bye! 😇