Production Deployment in ML Deployment

About

We have previously discussed deployment in the context of ML Training as it was also relevant to the issues of testing. More often than not it is important to test in production as it is impossible to know if you have covered all your bases despite having integration, smoke, as well as acceptance testing and so forth. So here we will do a quick recap on the deployment methods.

Deployment Methods

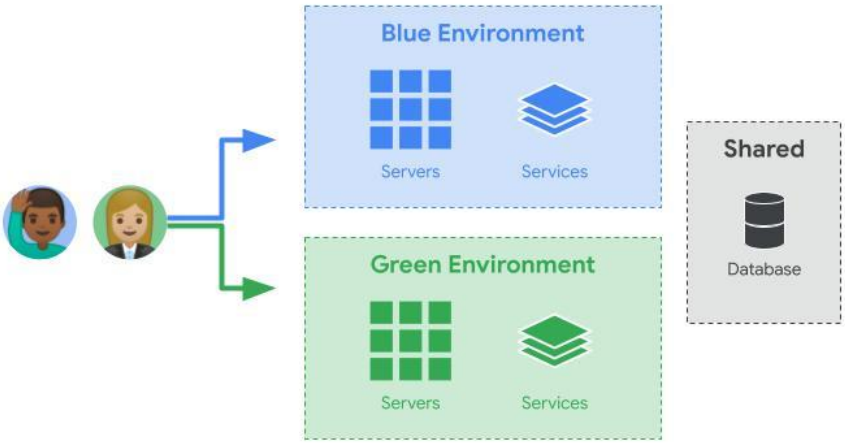
First, we have the **basic deployment**. This is the riskiest technique, and it involves simply replacing our current model (if one exists) with our new model. If everything works fine, then everything runs smoothly and there are no issues. This type of deployment is also very fast since there is little to no delay in predicting serving. However, if there are any issues, the lack of a failover mechanism can cause problems. The issues may be time-consuming to fix and even require the entire production environment to undergo downtime.

Next is the **multi-service deployment**. In this type of deployment, we do not “overwrite” a saved model or application, but merely add an updated version of it. This way, if there is an error in the production environment, we can always failover to an earlier version. This method is safer than the basic deployment but still suffers from issues in terms of maintenance since extensive testing might be necessary to make sure any infrastructure changes are compatible with all current and previous versions.

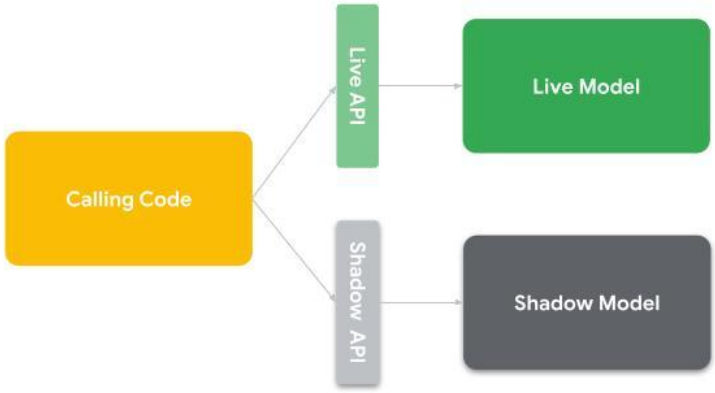
Rolling deployment offers some new advantages. With this method, all of our deployment environments will be updated with our new model or application, but this update will be done in batches. The advantage of this method is that only a few deployed devices are updated at a time making it easier to roll back a release and making it safer than a basic deployment as only a few users at a time will be affected by the update.

A similar approach is a **canary deployment**. This allows us to put our model into production, but only targets a subset of users which may be selected randomly or through some form of filter (e.g., geographically). This allows us to test our model in a true production environment and to compare the real-time performance to the existing deployment. This deployment strategy is essentially an implementation of A/B testing.

Another popular strategy is **blue-green deployment**. This is a common deployment strategy and involves two identical environments, one called the “staging” environment, and the second called the “production” environment. Within the staging environment, user testing and quality assurance can be performed before our model or application is deployed to the production environment. Once we are satisfied with our model’s performance in the staging area, the model is then deployed into the production environment. This method is more expensive than rolling or canary deployment since it requires both a production and staging environment which each may be extremely large.



The final deployment strategy we will discuss is **shadow deployment**. This involves having both our current and tentative deployments running simultaneously in the production environment. Real-time traffic is sent to the “shadow” deployment (which represents our tentative model or application), but predictions are not sent to the user. Instead, they are monitored to test for performance and stability. Once some threshold values are reached, the model can then be solely deployed in the production environment. This procedure is complex to set up compared to several of the other methods.



As you can now see, there are many ways for us to introduce a new model into our production environment. Some of these are easy to implement, but relatively risky, while others are complex but relatively safe. The most suitable deployment strategy depends strongly on the application, but also things like cost, existing infrastructure, and technical expertise available.

We just talked about production deployment from the lens of all the things that can go wrong even after we have done numerous amounts of testing. Therefore, we want to have some sort of a staged deployment to ensure that deploying our models into production won't cause things to crash. This is why we learned about various **rolling deployment** strategies such as the canary, shadow, and blue-green deployment strategies.