Deploying Machine Learning Models in Shadow Mode

A Guide

Created: 30 March 2019
Last updated: 12 April 2020

Introduction

The strategies you adopt when deploying software have the potential to save you from expensive and insidious mistakes. This is particularly true for machine learning systems, where detecting subtle data munging, feature engineering or model bugs in production can be very challenging, particularly when the production data inputs are hard to replicate exactly. “Shadow Mode” is one such deployment strategy, and in this post I will examine this approach and its trade-offs.

Contents

Picture a data scientist called Jenny. She works at a bank. Jenny is building a new credit risk assessment model. This new model requires a dozen additional features that none of the bank's other credit risk models use, but it is decided that the improvements in performance warrant the inclusion of these extra features. Jenny is happy that her model performs well, and passes it to one of her ML engineer colleagues who proceeds to write some new code to perform the feature engineering steps in the production ML application. The features are all captured in config and all the tests pass. The model is released to customers, where it begins assessing loan applicant credit worthiness.

Somewhere between the data science department and the release, one of the features was created incorrectly. Perhaps it was a config typo, or maybe the pipeline code had an edge case, or it could have just been a simple breakdown in communication. Now for example, instead of using a key feature which distinguishes student loan payments, an older version of the feature which doesn't make the distinction is used. This makes it appear that a subset of bank customers are spending more on their lifestyle, which means the bank issues them a slightly worse credit score, and therefore a loan with a slightly higher interest rate than it usually would (the exact specifics don't matter, the point is, there is a subtle error). But there is no dramatic effect, so no alarms are sounded, and a few months go by. Parts of the loan book are sold to external parties and assurances are made to the industry regulator about the level of risk being taken.

One day, three months later, someone discovers the bug. By this time, thousands of loans have been issued to people who should have been offered a better interest rate, meaning that all the bank's future predictions of default rates, its promises to the regulatory authority, and its guarantees to companies which bought a part of the loan book, are inaccurate. The CEO of the bank spends a long time shouting at the CTO and the Chief Data Scientist, and then spends the next month apologizing to everyone. The business survives, if the regulatory authority is in a forgiving mood.

This is a story to illustrate a point. The fact that it involves a bank doesn't mean that these are risks faced only by ML practitioners in finance. Many industries using machine learning to make important predictions are at risk of similar disasters. Think of models in healthcare, agriculture, shipping and logistics, legal services, the list goes on. But this story could have had a very different ending if alternative deployment strategies had been in place, and these strategies are what we will consider in the upcoming sections.

2. What is Shadow Mode?

"Shadow Mode" or "Dark Launch" as Google calls it is a technique where production traffic and data is run through a newly deployed version of a service or machine learning model, without that service or model actually returning the response or prediction to customers/other systems. Instead, the old version of the service or model continues to serve responses or predictions, and the new version's results are merely captured and stored for analysis. There are a few ways to implement this functionality, which we will discuss in section 4.

Shadow mode should not be confused with feature flagging/toggling. Feature flags may well be used to switch shadow mode on or off, but they are a separate type of technique, and do not offer the simultaneous testing element central to shadow mode. Broadly speaking, there are two key reasons to use shadow mode:

1. Ensure that your service/model handles inputs in the way that you intended
2. Ensure that your service can handle the incoming load

The second reason is much more of a general engineering load testing purpose, which is a highly valuable and important area to check, but since we are focusing more on machine learning-specific issues, this post will focus on the first area.

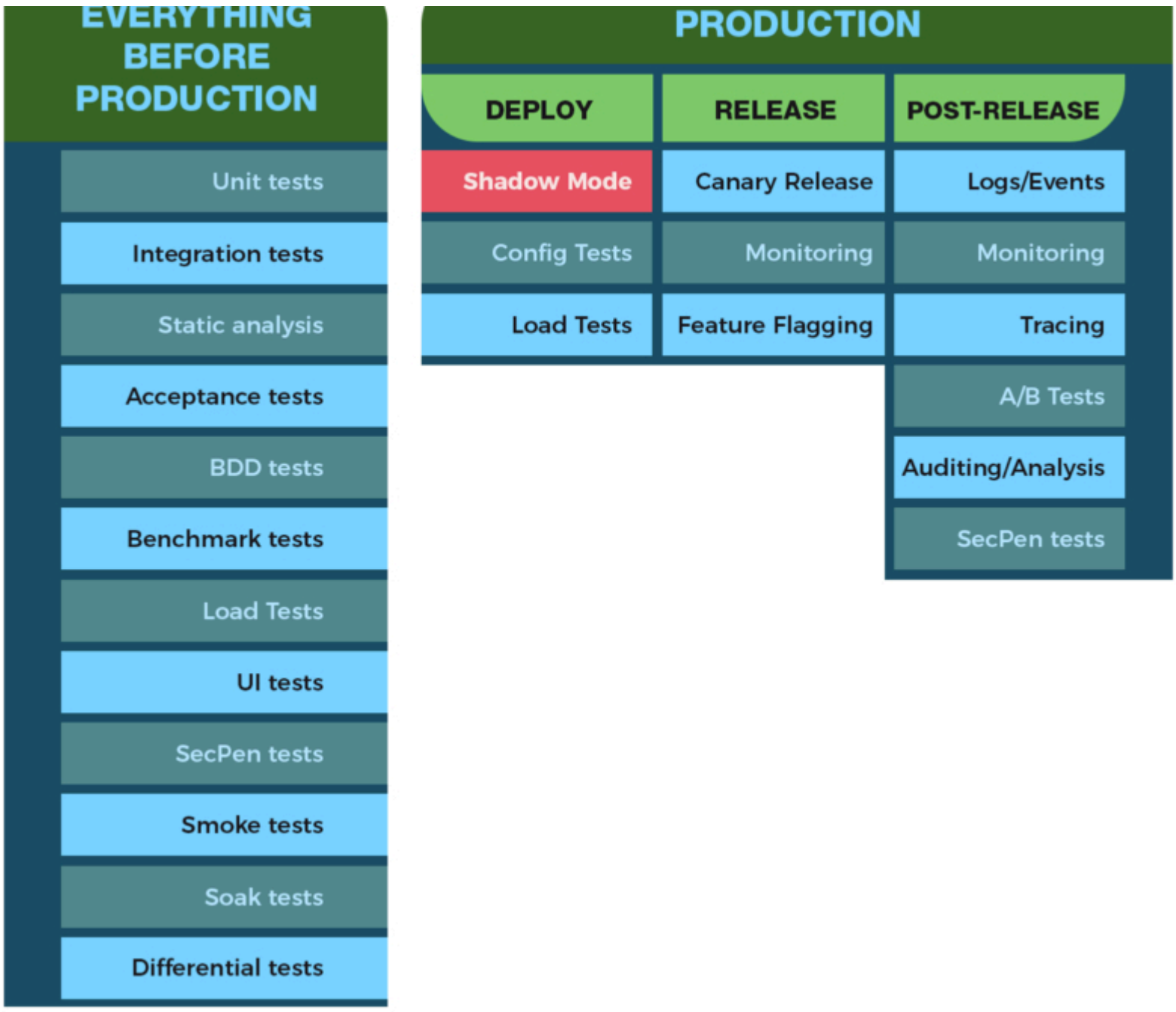
Fundamentally, "Shadow Mode" is a form of testing in production. Some people's first reaction to hearing about testing in production is "why would you do that when you have a dev/sandbox/staging environment?" This is where some of the realities of maintaining environments designed to replicate production come into play, namely:

- Creating realistic data in non-production databases (this is particularly difficult when production data is sensitive and requires GDPR-type compliance). If inbound data is complex (images, credit files, medical records) then creating test data for a non-production environment that covers the vast number of potential edge cases in the input data is a serious challenge.
- Keeping the non-production environment data up to date.
- Ensuring the infrastructure of the non-production environments matches - in practice it is rarely cost-effective to have a staging cluster with as many nodes as that of production.
- Replicating the inbound traffic realistically.
- Investing in the same quality of monitoring, metrics, tracing and analysis for non-production environments.
- In the case of real-time ML systems, the challenge is even harder, as you need to simulate frequent updates to your models, as well as evolving input data.

This is not to say that staging environments don't serve a very useful purpose - they are extremely important. It *is* to say, however, that just because your machine learning system seems to be behaving as expected in the staging environment, doesn't mean you can expect it to be fine in production.

So generally speaking, what are our potential tools for testing beyond the staging environment?

Considering options before, during and after release:



This isn't an exhaustive list, but it gives a feel for the range of tools available. Crucially, it highlights a distinction between "Deployment" and "Release". Let's consider this for a moment.

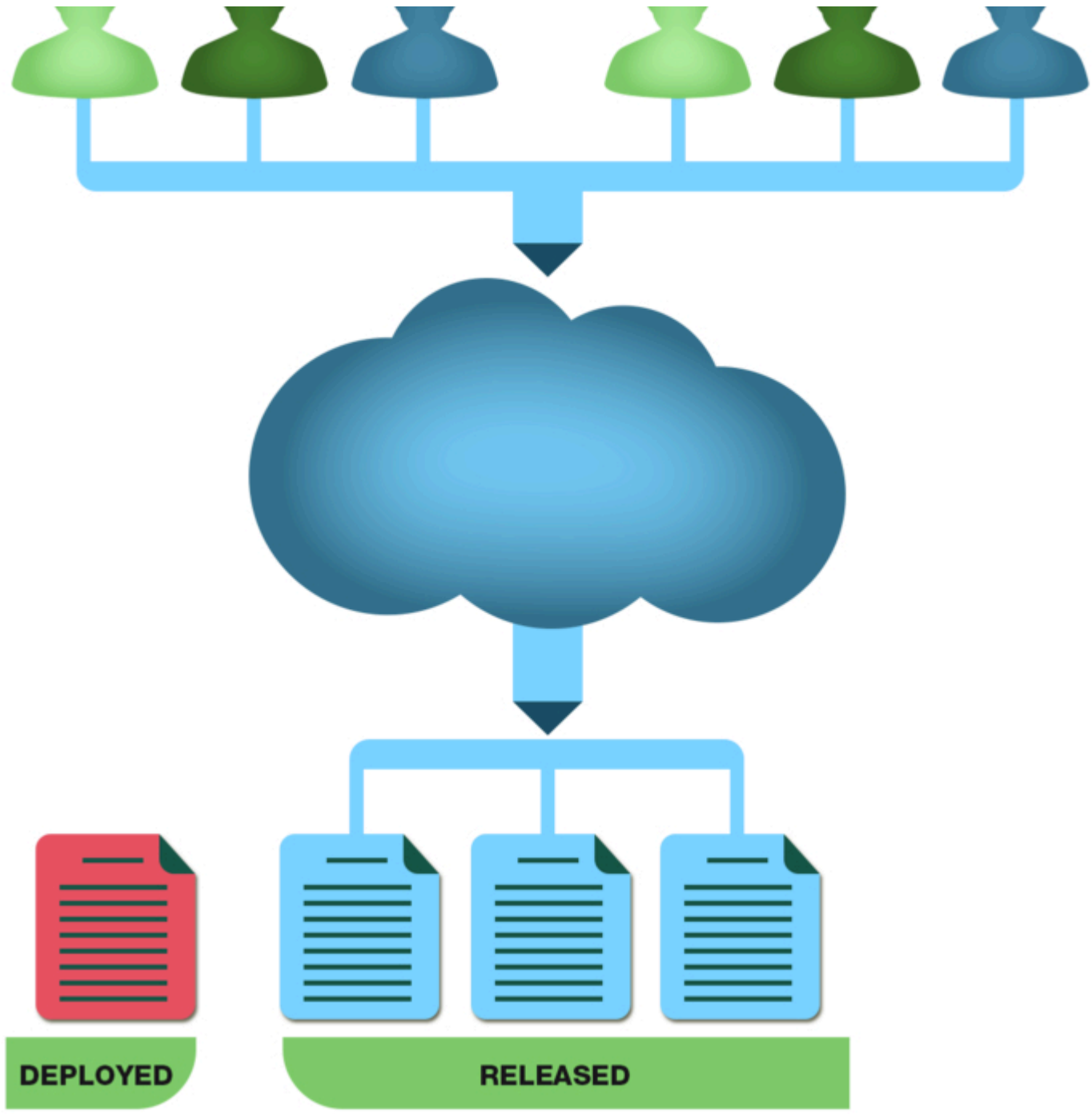
Deploying vs. Releasing

In most companies where I have worked, the terms "deploy" and "release" have been used more or less interchangeably. Whilst at first glance the need for a distinction between the two terms might seem like laughable pedantry, there is great value to be gained when designing our ML systems and deployments if we *do* consider the difference. This post from Turbine Labs does a great job summarizing the issue, with a key quote being:

> *Deployment need not expose customers to a new version of your service.*

Which is to say, you can make the distinction between having a new version of your system and all its dependencies running on production infrastructure, and switching customer-facing systems to using that new version. For an industry so obsessed with naming things, there isn't really an agreed terminology for this. Things can be more confusing because many companies talk about "deploying a release", see this post from Octopus Deploy, and because other companies use alternative terminology such as "rollout" or "ship". The main thing is to have an understanding of the nuance, and for the rest of this post I will use the terms like so: Deployment ("in production but not affecting customers/users") and release ("in production and affecting customers/users").

A Machine Learning Example



4. Shadow Mode Implementation Approaches

There are two fundamental approaches to Shadow Mode:

1. Application level implementations
2. Infrastructure level implementations

Application Level

At the application level, shadow mode can be as simple as a code change passing inputs to the current and the new versions of a ML model, saving the outputs from both, but only returning back the outputs for the current version. In scenarios where performance is a concern (systems that give real-time predictions, or that have algorithms which are time-intensive), then best practice is to pass the inputs and record the outputs on the new model asynchronously (perhaps using threads or by passing the information to a distributed task queue). More advanced systems might pass the inputs to a separate Kafka topic for the new model.

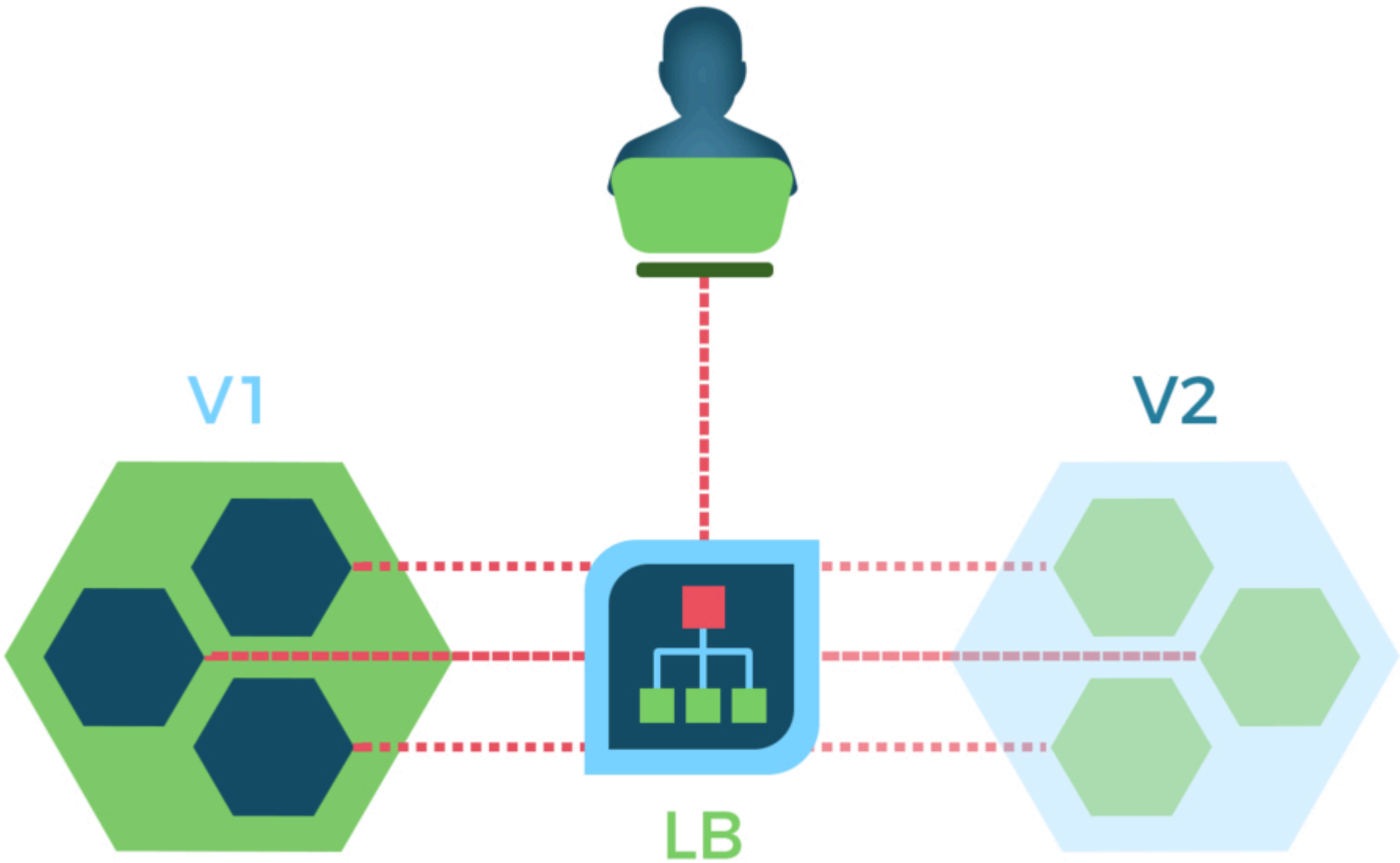
You should already be recording all inputs and outputs to your model, either in logs or a database, for reproducibility. Shadow mode introduces the need to be able to distinguish between predictions from the current (customer-facing) model and the model in shadow mode. You should design your logging and/or database schema accordingly, such that this distinction can be made, for example by including a column in a database for recording the model version.

It is also possible to split the traffic on the client side, calling separate API endpoints from the browser and/or mobile client.

Shadow deployments at the infrastructure level have some complex elements. At the most basic level, if you have a load balancer responsible for directing traffic to your applications, this can mean configuring this load balancer to fork incoming traffic to /v1 and /v2 endpoints of a prediction API. However, this needs to be done in a way that doesn't have unintended side effects. For example, if any services make external API calls (perhaps to fetch data required to generate a feature), then you need to ensure these are not duplicated to avoid slowing down the system and/or doubling costs for paid APIs. Similarly, the setup needs to ensure that operations that should only happen once, such as customer creation, payment collection, email sending etc. are not triggered on both sides of the fork. It may be necessary to mock responses to avoid triggering errors, and to inject headers to allow a distinction to be made between a shadow request and a live request.

More advanced infrastructure level implementations of shadow mode might rely on deployment tools, such as Istio (which offers a host of extra functionality on top of Kubernetes). Istio offers shadow mode deployments out of the box - in Istio terminology this is known as "mirroring":

> *Mirroring sends a copy of live traffic to a mirrored service. The mirrored traffic happens out of band of the critical request path for the primary service. In this task, you will first force all traffic to v1 of a test service. Then, you will apply a rule to mirror a portion of traffic to v2.*



Once your new model is deployed in shadow mode, it's time to reap the benefits. In addition to the standard service-level monitoring you should be conducting at all times (HTTP response codes, latency, memory usage etc.), you are now able to compare model inputs and outputs. This comparison will be with both the research environment, and also over time to make sure inputs and outputs do not suddenly change (perhaps due to a change in an external data source).

Key things to analyse include:

- The raw data entering the pipeline - are there errors, missing data, unexpected database queries.
- The features being generated as inputs to the model - are there irregularities (apply standard statistical techniques here).
- The predictions being generated by the model - do these predictions and their inputs match the expected predictions and results from the research environment? If there is a divergence, is this expected?

The time you wait before conducting this analysis depends on the business requirements and the amount of traffic coming in. On high traffic sites, a few hours of data may be sufficient to allow you to conduct your analysis. For scenarios where traffic is lower, or precision is of paramount importance, it may be worth waiting for months of data to build up. This can also be a consideration for models highly affected by timing, where a fair comparison may require data to be collected from both the weekend and a weekday, night and day, or spring and summer.

6. When to Use Shadow Mode and Trade-Offs

Shadow mode deployments don't come for free. Time will have to be spent adjusting the system to accommodate the approach and conducting the model output analyses.

Regardless of your implementation approach, you will place extra strain on your system. If you go down the infrastructure route, this can result in shadowing the entire production traffic stream against a deployed service. At peak traffic times, you might end up requiring 2X the capacity to perform such testing, which has cost implications certainly, and may be beyond your system's capabilities.

If you choose the application implementation, then additional asynchronous queues and plumbing may be required to capture the data without interfering with your system response time. Feature flags may be appropriate to make sure you can quickly switch a shadow deployment off.

Conducting the analysis itself may require additional system changes, especially if you wish to process and check data streams in real-time. In scenarios where an ML system is conducting multiple model deployments a day, manual batch testing may be unrealistic - instead a service may be required to check that predictions fall within expected bounds, which can be configured depending on the expectations from the research environment. Many large companies create entire "diffing" services and/or frameworks, although this strays more into the realm of tap comparisons.

Think carefully about where you implement the shadow mode "switch". If you are implementing it in a client (e.g. a mobile app), then if any unexpected issues occur you may have to create a new mobile app release to remove shadow API calls, which can take an excruciating amount of time. Having said that, if your backend API contains too much logic, it may be too labor intensive to change that area of the code.

No matter the approach, you will need to improve the sophistication of the entire system's logging, monitoring and alerting to make sure that errors in the shadow deployment do not cause the usual production alerts and pagers to go off - and do this very carefully so that you do not accidentally shut off genuine production alerts! Your data collection techniques also need to evolve so that you can easily distinguish between shadow and non-shadow model inputs and outputs.

From Shadow to Live

investors or regulators would have been affected. Sure, Jenny would have had to spend more time tracking down the missing feature, but that is a very different level of damage and cost. In summary, when done properly, shadow mode deployments can bring some real peace of mind to machine learning model deployments.

Best of the Blog

The Ultimate AI Engineering Tutorial Series

The Ultimate FastAPI Tutorial Series

How to Deploy Machine Learning Models

Monitoring Machine Learning Models in Production

Deploying Machine Learning Models in Shadow Mode

Why Use Python Tox and Tutorial

Category

Machine Learning 5

Tags

Python 26 Machine Learning 6 Monitoring 3