Monitoring and Observability in Machine Learning

Ensuring Reliable and Effective ML Models in Production

Lecturer: Vangelis Oden - Technology Lead (Kera)
Assistant: Natalija Mitic - AI/ML Engineer (Kera)

Agenda

- Monitoring ML Systems
- Logging and Alerting Systems for ML Applications
- Tools for Monitoring ML Systems
- Best Practices Monitoring
- Conclusion
- Q&A

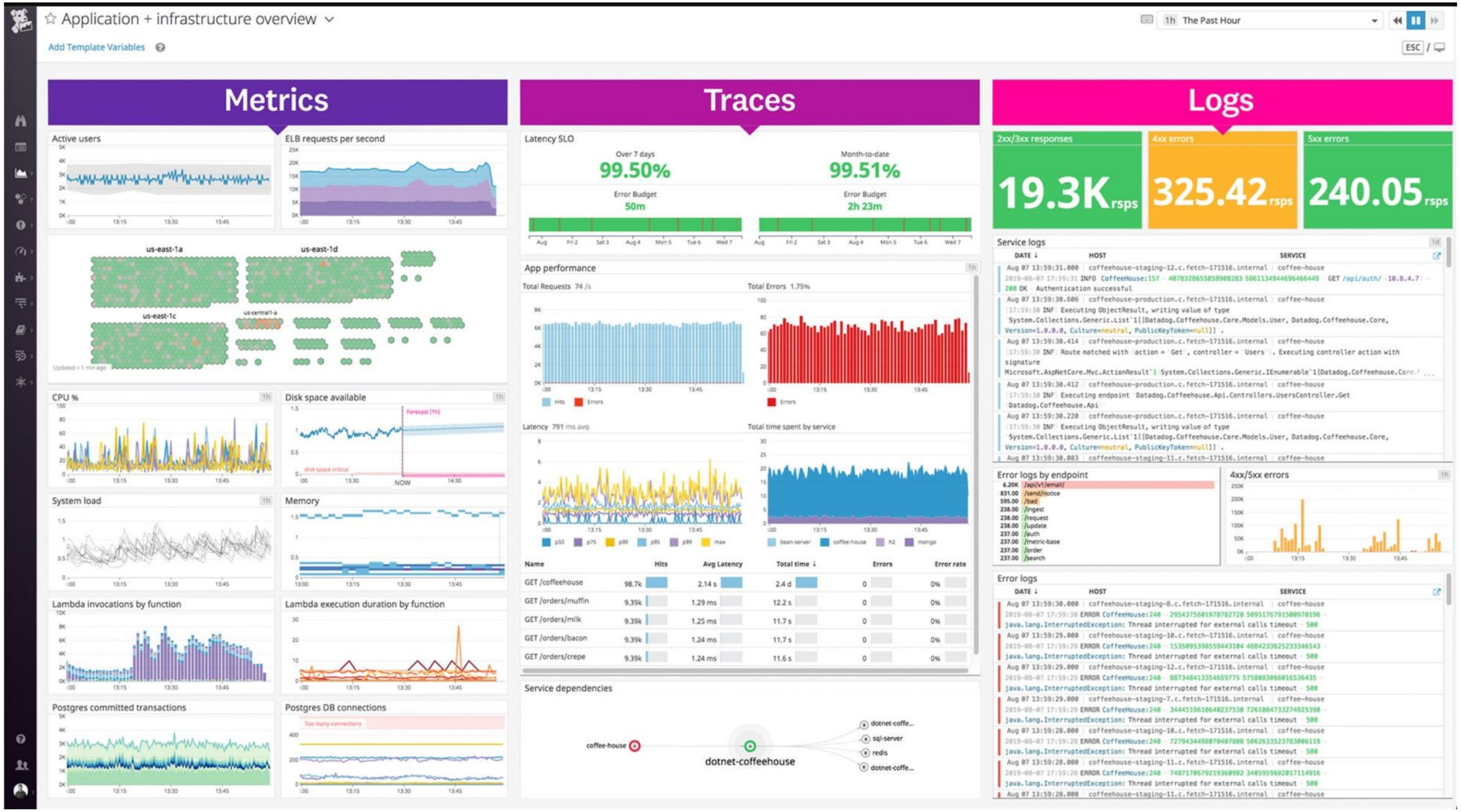
Monitoring ML Systems

Intuition:

Real work begins once our models are deployed to production. With Machine learning, we haven't explicitly how something works but used data to architect a probabilistic solution. This approach is subject to natural performance degradation over time, as well as unintended behaviour, since the data exposed to the mode will be different from what it has been trained on.

- Don't try to avoid it, rather understand and mitigate as much as possible.

Monitoring ML Systems



Name	Hits	Avg Latency	Total time ↓	Errors	Error rate
GET /coffeehouse	98.7k	2.14 s	2.4 d	0	0%
GET /orders/muffin	9.39k	1.29 ms	12.2 s	0	0%
GET /orders/milk	9.39k	1.25 ms	11.7 s	0	0%
GET /orders/bacon	9.39k	1.24 ms	11.7 s	0	0%
GET /orders/crepe	9.39k	1.24 ms	11.6 s	0	0%

Error logs by endpoint

6.20K	/api/v1/email/
831.00	/send/notice
595.00	/bad
238.00	/ingest
238.00	/request
238.00	/update
237.00	/auth
237.00	/metric-base
237.00	/order
237.00	/search

Service logs

```
DATE ↓                HOST                                              SERVICE
Aug 07 13:59:31.000 | coffeehouse-staging-12.c.fetch-171516.internal | coffee-house
2019-08-07 17:59:31 INFO CoffeeHouse:157 - 4078328655858908283 5861134944696466449 - GET /api/auth/ (10.8.4.7) -
200 OK - Authentication successful
Aug 07 13:59:30.606 | coffeehouse-production.c.fetch-171516.internal | coffee-house
[17:59:30 INF] Executing ObjectResult, writing value of type
'System.Collections.Generic.List`1[[Datadog.Coffeehouse.Core.Models.User, Datadog.Coffeehouse.Core,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null]]'.
Aug 07 13:59:30.414 | coffeehouse-production.c.fetch-171516.internal | coffee-house
[17:59:30 INF] Route matched with {action = "Get", controller = "Users"}. Executing controller action with
signature
Microsoft.AspNetCore.Mvc.ActionResult`1[System.Collections.Generic.IEnumerable`1[Datadog.Coffeehouse.Core.! ...
Aug 07 13:59:30.412 | coffeehouse-production.c.fetch-171516.internal | coffee-house
[17:59:30 INF] Executing endpoint 'Datadog.Coffeehouse.Api.Controllers.UsersController.Get
(Datadog.Coffeehouse.Api)'
Aug 07 13:59:30.220 | coffeehouse-production.c.fetch-171516.internal | coffee-house
[17:59:30 INF] Executing ObjectResult, writing value of type
'System.Collections.Generic.List`1[[Datadog.Coffeehouse.Core.Models.User, Datadog.Coffeehouse.Core,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null]]'.
Aug 07 13:59:30.083 | coffeehouse-staging-11.c.fetch-171516.internal | coffee-house
```

Error logs

```
DATE ↓                HOST                                              SERVICE
Aug 07 13:59:30.000 | coffeehouse-staging-8.c.fetch-171516.internal | coffee-house
2019-08-07 17:59:30 ERROR CoffeeHouse:240 - 2954375601978782728 5095176791500978196 -
java.lang.InterruptedException: Thread interrupted for external calls timeout - 500
Aug 07 13:59:29.000 | coffeehouse-staging-10.c.fetch-171516.internal | coffee-house
2019-08-07 17:59:29 ERROR CoffeeHouse:240 - 1535095398559443104 4884233625233346143 -
java.lang.InterruptedException: Thread interrupted for external calls timeout - 500
Aug 07 13:59:29.000 | coffeehouse-staging-12.c.fetch-171516.internal | coffee-house
2019-08-07 17:59:29 ERROR CoffeeHouse:240 - 8873484133546S9775 5758083066016536435 -
java.lang.InterruptedException: Thread interrupted for external calls timeout - 500
Aug 07 13:59:29.000 | coffeehouse-staging-7.c.fetch-171516.internal | coffee-house
2019-08-07 17:59:29 ERROR CoffeeHouse:240 - 3444519610640237538 7261084733274925390 -
java.lang.InterruptedException: Thread interrupted for external calls timeout - 500
Aug 07 13:59:28.000 | coffeehouse-staging-10.c.fetch-171516.internal | coffee-house
2019-08-07 17:59:28 ERROR CoffeeHouse:240 - 7279434498070407808 5062633523783006119 -
java.lang.InterruptedException: Thread interrupted for external calls timeout - 500
Aug 07 13:59:28.000 | coffeehouse-staging-11.c.fetch-171516.internal | coffee-house
2019-08-07 17:59:28 ERROR CoffeeHouse:240 - 7487170679219360982 3405959682017114916 -
java.lang.InterruptedException: Thread interrupted for external calls timeout - 500
Aug 07 13:59:28.000 | coffeehouse-staging-11.c.fetch-171516.internal | coffee-house
```

Monitoring ML Systems

System Health:

First step to ensure that our model is performing well is to ensure that the actual system is up and running as it should.

- This can include metrics specific to service requests such as latency, throughput, error rates, etc. as well as infrastructure utilization such as CPU/GPU utilization, memory, etc.
- Cloud providers usually provision, but you can setup Grafana, Datadog, etc. to ingest system performance metrics from logs to create a customized dashboard and set alerts.

Monitoring ML Systems

Performance:

This could be monitoring quantitative evaluation metrics that we used during model evaluation (accuracy, precision, etc.) but also key business metrics that the model influences (ROI, click rates, etc.)

- We want to always inspect how the model performs across a period of time that's significant for our application - daily, weekly, monthly, etc.
- Using sliding metrics might be more indicative of our system’s health and allow us identify issues faster by not obsuring with historical data.

Monitoring ML Systems

Delayed Outcomes:

Ground-truth outcomes may not always be available to determine the model's performance on production inputs. Especially true if there is significant lag or annotation is required on the real-world data. We could mitigate by:

- Devise an approximate signal that can help us estimate the model's performance.
- label a small subset of our live dataset to estimate performance. This subset should try to be representative of the various distributions in the live data.
- Accept the risk of delayed outcomes.

Monitoring ML Systems

Warning:

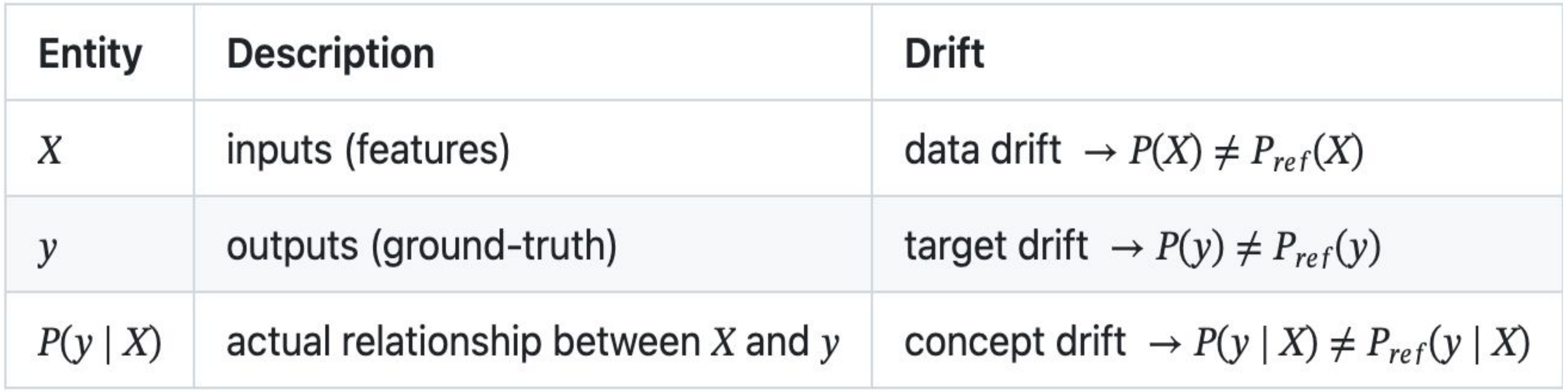
If we wait to catch the model decay based on the performance, it may have already caused significant damage to downstream business pipelines that are dependent on it. We need to employ more fine-grained monitoring to identify the sources of model drift prior to actual performance degradation.

Monitoring - Drift

Intuition:

Understanding the different types of issues that can cause our model's performance to decay is key.
Best way is to look at all the moving pieces of what we're trying to model and how each once can experience drift.

Monitoring - Drift



Entity	Description	Drift
X	inputs (features)	data drift $\rightarrow P(X) \neq P_{ref}(X)$
y	outputs (ground-truth)	target drift $\rightarrow P(y) \neq P_{ref}(y)$
$P(y \mid X)$	actual relationship between X and y	concept drift $\rightarrow P(y \mid X) \neq P_{ref}(y \mid X)$

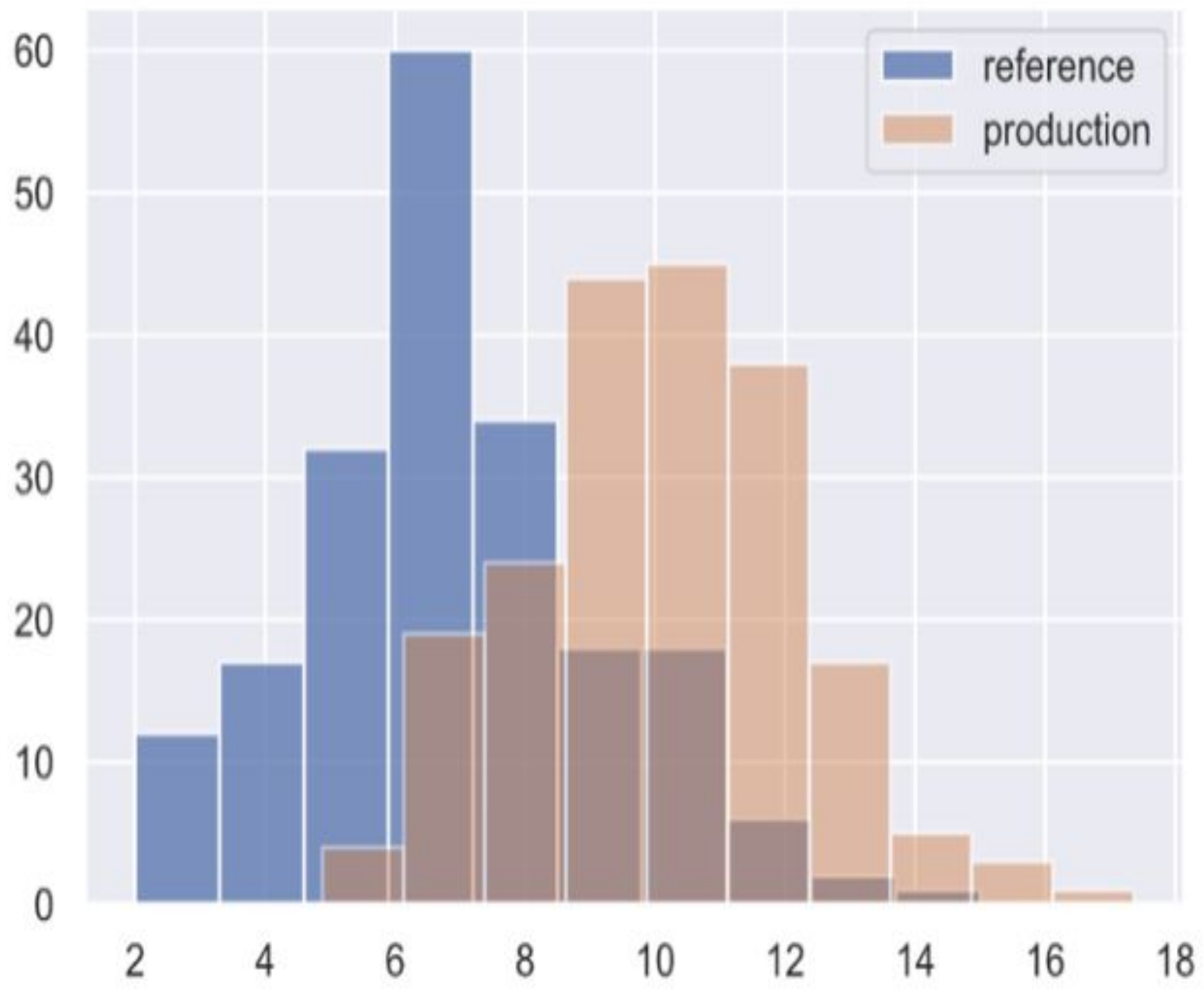
Monitoring - Drift

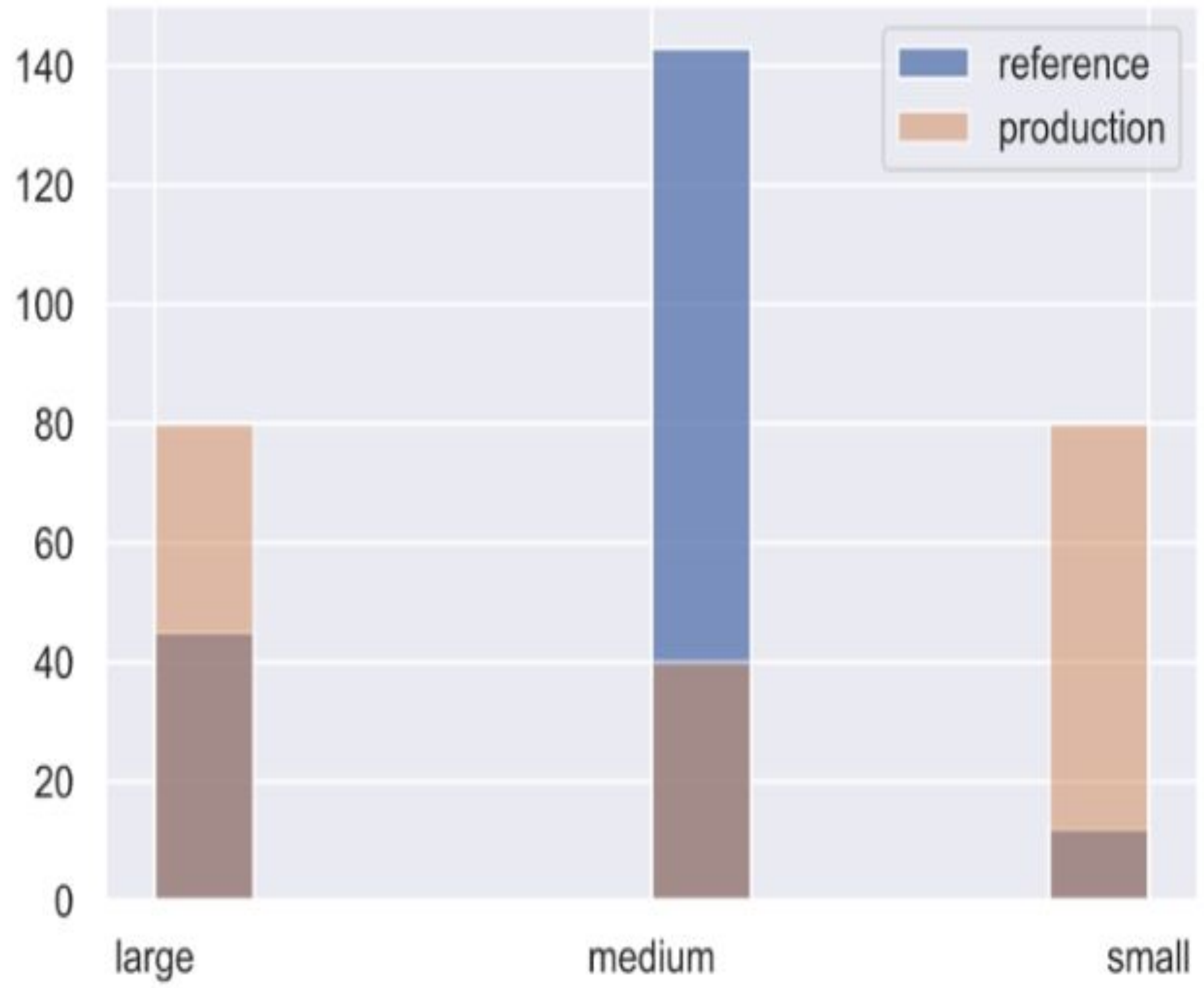
Data Drift:

Also known as feature drift or covariate shift occurs when the distribution of the production data is different from the training data.
Actual cause of drift can be attributed to natural changes in the real-world but also to systemic issues such as missing data, pipeline errors, schema changes, etc.

- inspect and trace data drifts along its pipeline to identify when and where the drift was introduced.

Monitoring - Drift





Data drift in continuous and categorical features

Monitoring - Drift

Target Drift:

We can also experience drift in the outcomes of the model.

- This can be a shift in the distributions but also the removal or addition of new classes with categorical tasks.
- Retraining can mitigate mitigate the performance decay caused by target drift, it can often be avoided with proper inter-pipeline communication about new classes, schema changes, etc.

Monitoring - Drift

Concept Drift:

This basically looks at the actual relationship between the inputs and outputs and how they drift over time.
Concept drift renders our models ineffective because the patterns it learned to map between the original inputs and outputs are no longer relevant.
Concept drift can sometimes occur in various patterns individually or simultaneously:

- gradually over a period of time.
- abruptly as a result of external events.
- periodically as a result of recurring events.

Monitoring - Locating Drift

How to locate and how often to measure drift?
We need to consider the following constraints:

- reference window : the set of points to compare production data distributions with to identify drift.
- test window : the set of points to compare with the reference window to determine if drift has occurred.

Typically the reference window is a fixed, recent subset of the training data while the test window slides over time.

Monitoring - Measuring Drift

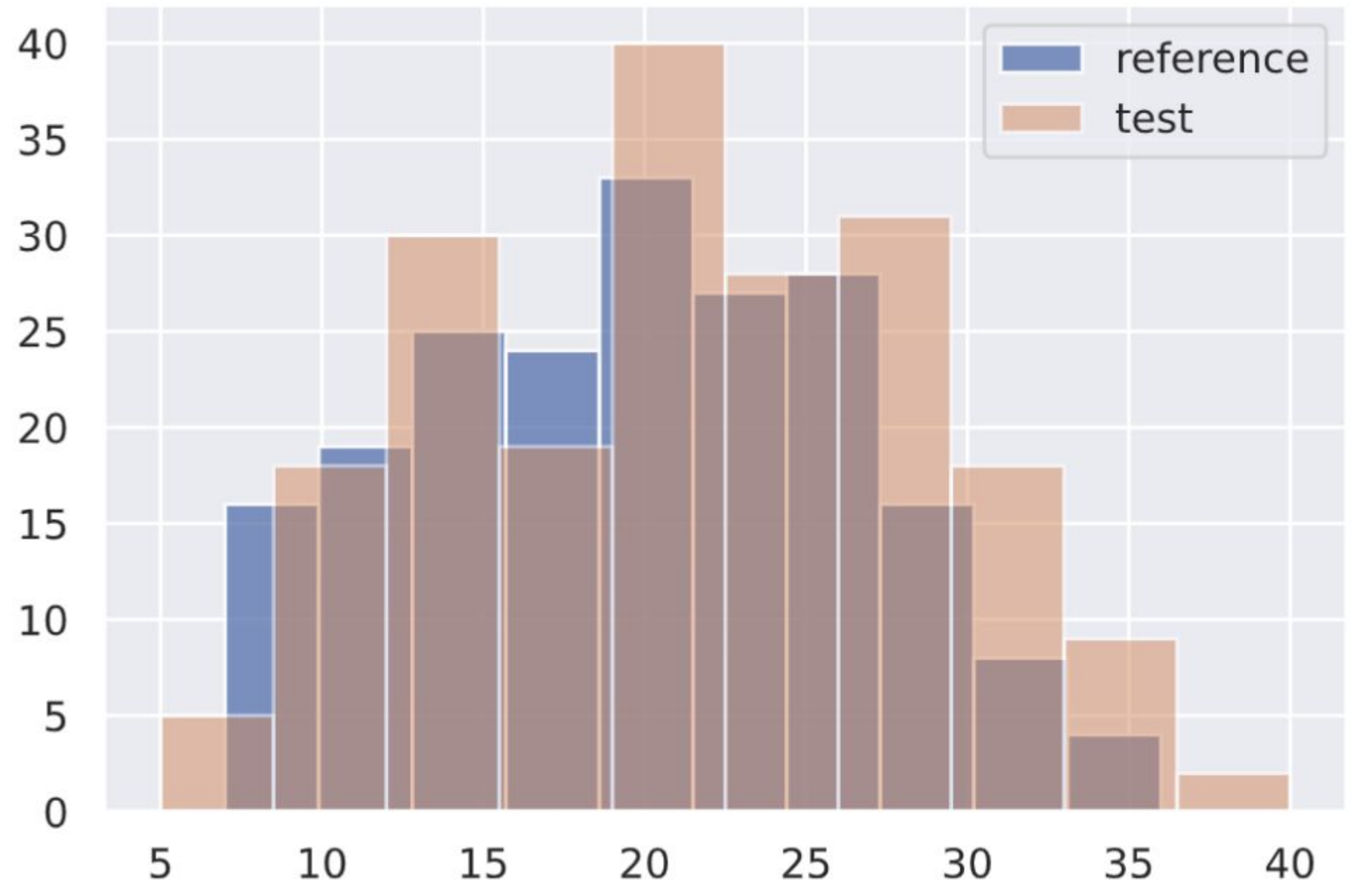
Univariate: KS Test:

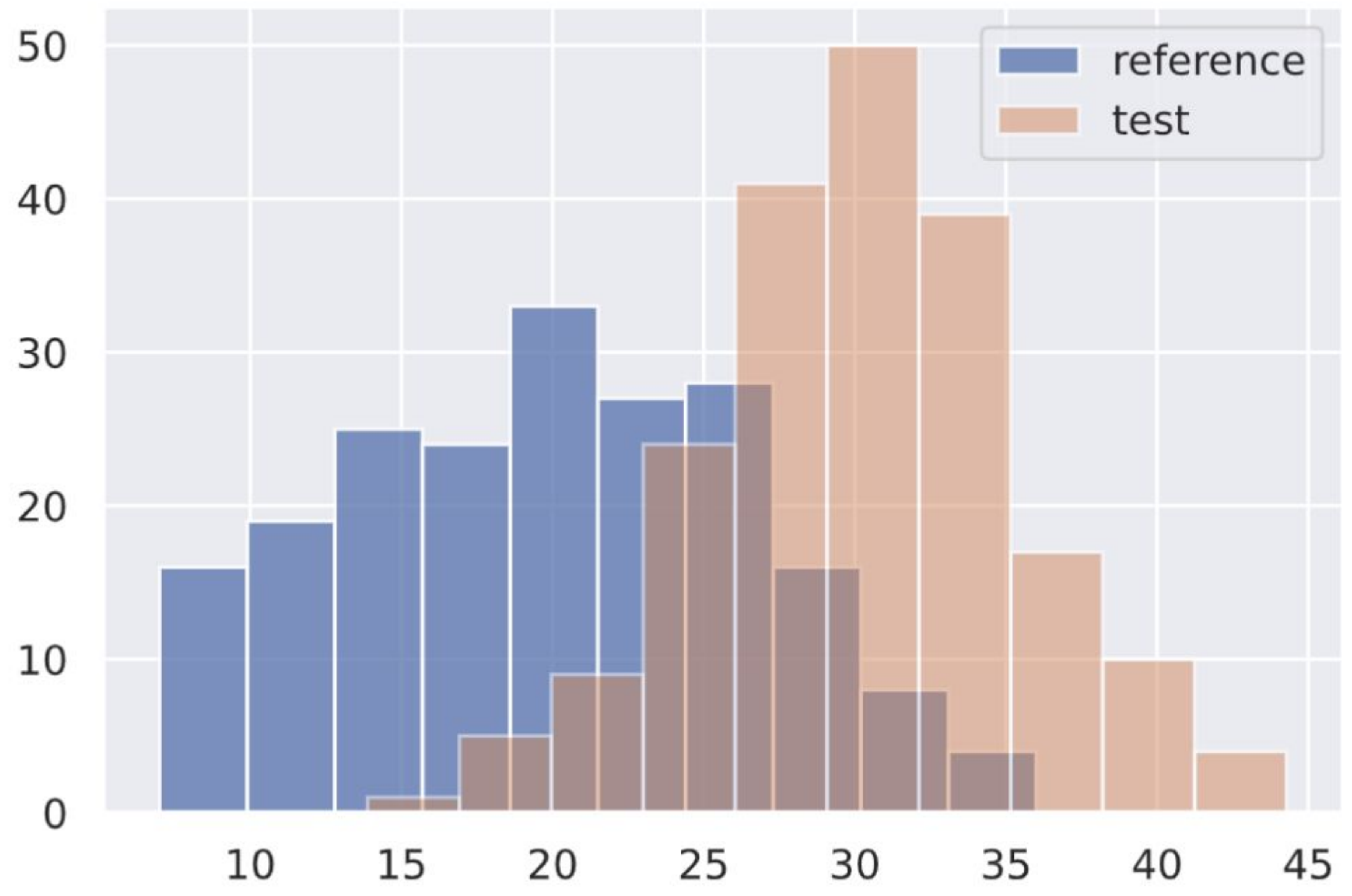
Case of 1D-univariate feature we want to monitor, we can use the popular Kolmorogov-Smirnov (KS) test.

The KS test determines the maximum distance between the cumulative density functions of two distributions.

metric : The lower the p-value, the higher the confidence that the distributions are different.

Monitoring - Measuring Drift





distance: 0.09,
p_val: 0.3927307

distance: 0.63,
p_val: 6.7101775e-35

Monitoring - Measuring Drift

Univariate: Pearson's Chi-squared:

We can apply the Pearson's chi-squared test to determine if a frequency of events in production is consistent with a reference distribution.

The Pearson's chi-squared test evaluates how likely it is that an observed difference between two sets of data arose by chance.

Monitoring - Measuring Drift

Multivariate::

While we can use univariate techniques for individual features within a dataset, we can also use some less-widely used techniques in analysing multivariate data.

- Use dimensionality reduction on the source dataset.
- Apply dimensional transform on the target dataset.
- Carry out two-sample test(s)
- Combine test statistic and shift detection

Paper: Failing loudly: An empirical study of methods for detecting dataset shift.

Logging and Alerting Systems for ML Applications

Logging and alerting are essential for tracking model inputs, understanding errors, and taking corrective actions. These systems help create transparency and provide insight into model behavior during production.

- logging : captures detailed information about model inputs, outputs and intermediate steps.
- alerting : sets up automatic alerts when performance deviates from expected behavior or when there is a system downtime.

Alerting

Once we've measured and identified significant drift, we need to devise a workflow to notify stakeholders of the issues.

Be aware of false positives and setup appropriate and robust alerting thresholds and constraints based on what is important to your specific application.

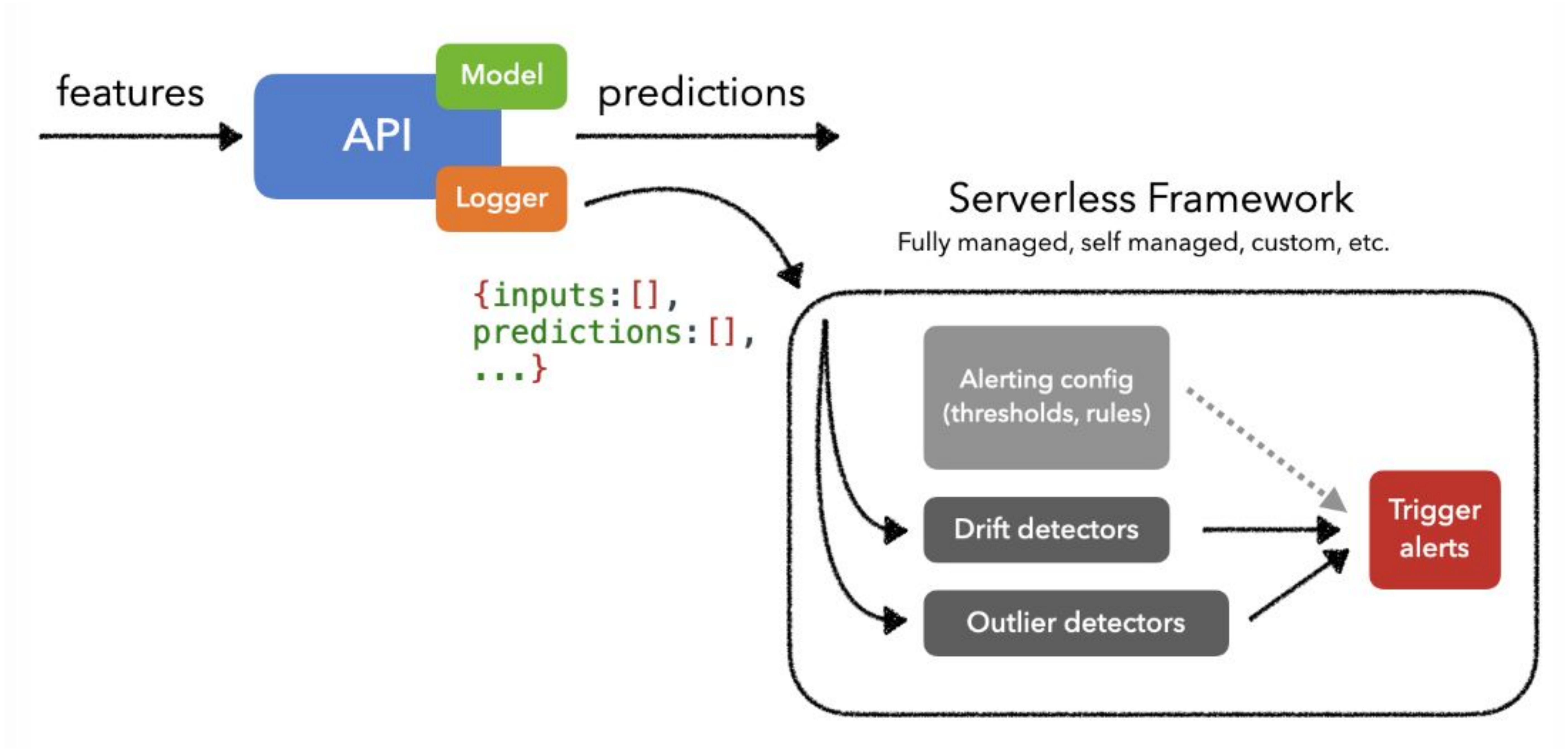
- fixed values for situations we're concrete aware of expected upper and lower bounds, i.e. missing value > 20%.
- forecasted thresholds dependent on previous inputs, time, etc.
- appropriate p-values for different drift detectors

Alerting - Production

Detecting and acting on drift can involve compute intensive operations, we need to set them up in ways that will either execute workloads on top of our event data streams (i.e. Kafka), or a batch execution process - depending on your use case.

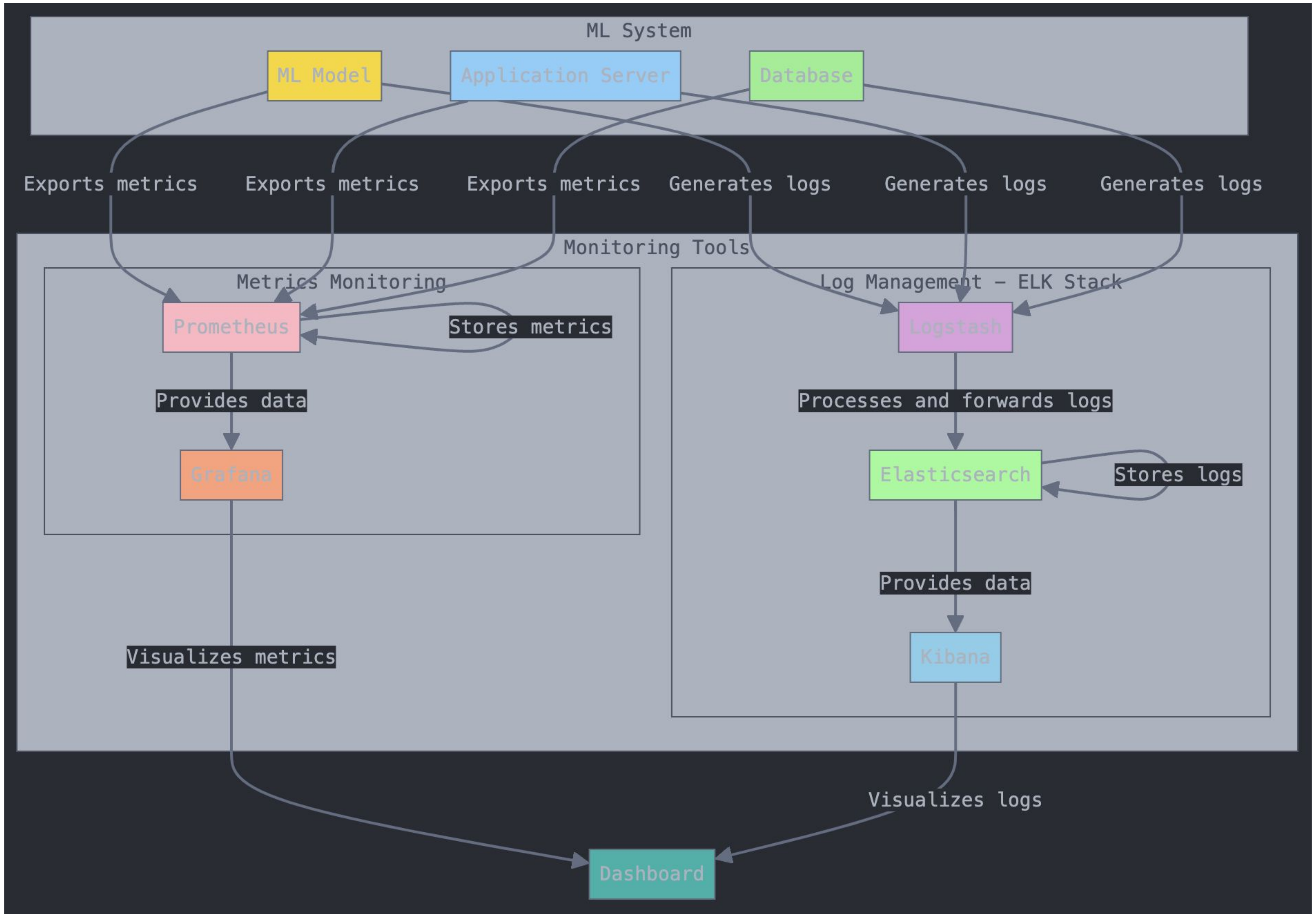
- This will allow you to segregate the resources for monitoring from your actual ML application and scale them as needed.

Alerting - Production



```
{inputs:[],
predictions:[],
...}
```

Tools for Monitoring ML Systems



Tools for Monitoring ML Systems

Popular Tools:

- **Prometheus** : A time-series database that collects and stores metrics from applications, including models.
- **Grafana** : A visualization tool that integrates with Prometheus to create real-time dashboards for monitoring model performance.
- **ELK Stack (Elasticsearch, Logstash, Kibana)** : A comprehensive logging and analytics tool to capture, process, and visualize log data in real-time.

Best Practices for Monitoring in ML

- **set baselines :** define and establish performance baselines for your models so deviations can be detected early
- **monitor data quality** : ensure that the input data maintains the same characteristics as the training data
- **track drift & decay** : implement drift detection tools to monitor changes in data distribution and model performance over time
- **automated alerts** : set up thresholds for critical metrics and ensure that alerts are triggered when these thresholds are breached
- **continuous feedback loop** : regularly retrain models using new data when drift or decay is detected.
- **use observability tools** : Prometheus, Grafana, ELK stack, etc.

Conclusion

- Monitoring and observability in ML are essential to ensure consistent model performance in production.
- Tools and techniques like drift detection, logging, alerting, and visualisation tools are integral to maintaining model reliability.
- It's not enough to be able to measure drift but also to be able to act on it.
- Adopting best practices in monitoring ensures that models remain robust and efficient, providing long-term value to the business.

Q&A

Thank You!