Monitoring and Testing web Applications in the Wild

Sam Clarke

https://unsplash.com/@droneviewsfr

Introduction

Something's not right, but what?

We can do better.

Introduction

The **On-board Diagnostic (ODB) II** vehicle interface.

“Gives the vehicle owner or repair technician access to the status of the various vehicle subsystems.”

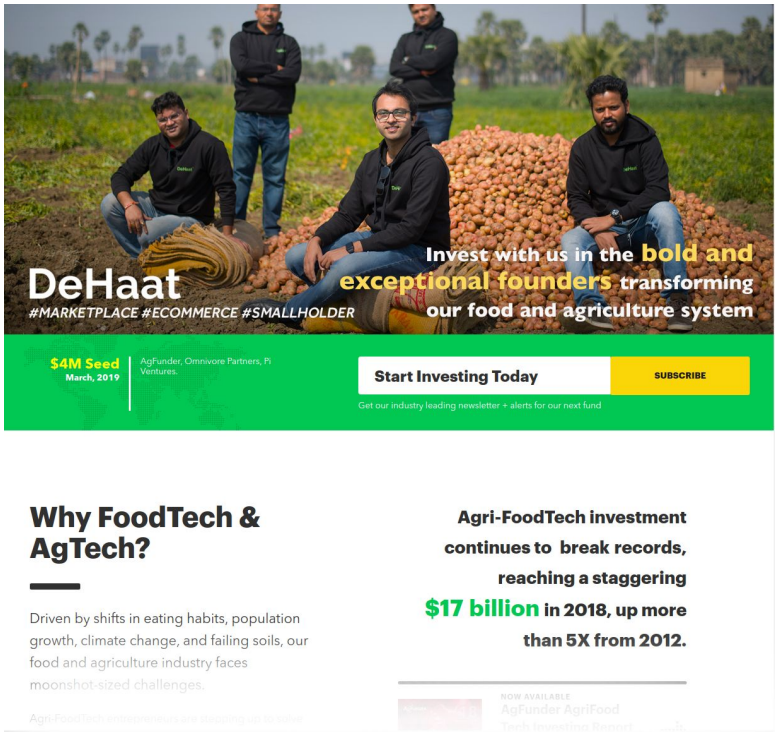
https://en.wikipedia.org/wiki/On-board_diagnostics

Who am I?

Sam Clarke

Senior Developer at

AgFUNDER



Summary

1. Why monitoring matters
2. What can be monitored
3. What happens when an issue occurs
4. Takeaways

Definitions

- **Dev** - the construction site
- **Production** - user-facing software
- **Monitoring** - continuous analysis
- **Metric** - something measurable
- **Load** - quantity of requests


Your app
will fail
https://unsplash.com/@sebastianhuxley

Why this talk?

“No battle plan survives first contact with the enemy.”

- Helmuth von Moltke the Elder, 1871

Dev != Production

“It worked in Dev.”

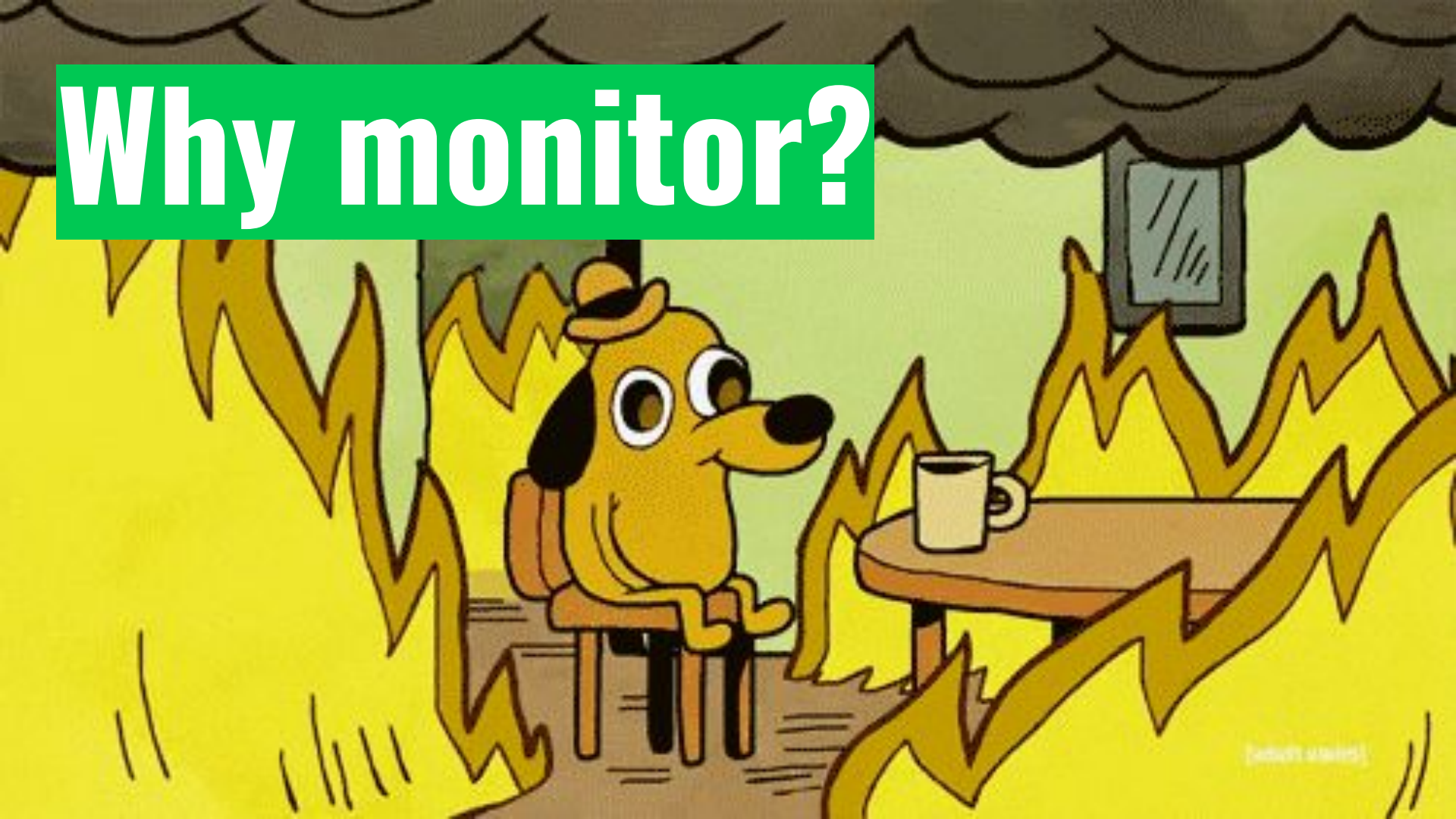
- Most developers at some point.

Dev != Production

- **Debug** mode is off (it is off right?)
- Multiple **users**, concurrent **requests**, varied **clients**.
- **Network latency** (clients, dbs, APIs)
- Production architecture may be significantly more **complex** (containers, clusters, permissions etc.)

Dev != Production

Staging environments reduce the gap between **Dev** and **Production**.


Why monitor?

Why monitor?

- Catch problems before they **escalate**
- Improve **response** times
- Create an historical **record** of performance
- **Observability** and **Insight** via **feedback**

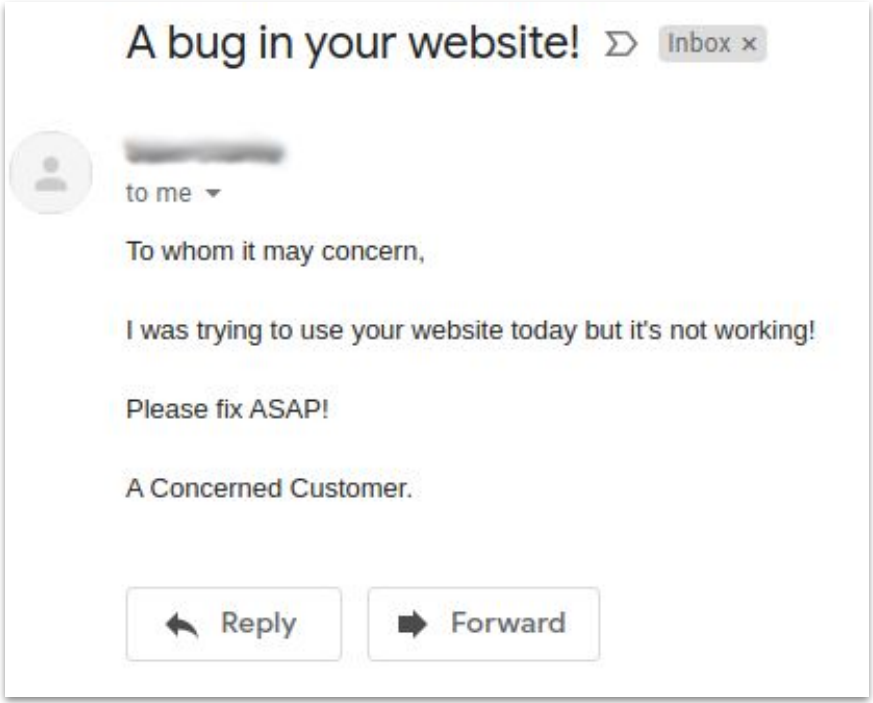
Insight requires feedback

https://unsplash.com/@hayden_dunsel

Why monitor?

This is **not** considered high quality feedback.

We want to be aware of and fix an issue **before** it gets to this stage.



What can go wrong?

- **Web server** - timeouts, file handling
- **Application** - bugs and stuff
- **Database** - data corruption, hardware
- **Host** - overloading, crashing, network
- **Caching***

What to Monitor?

- **Uptime** - availability

What to Monitor?

- **Uptime** - availability
- **Load** - host performance at scale

What to Monitor?

- **Uptime** - availability
- **Load** - host performance at scale
- **Logging** - application code

What to Monitor?

- **Uptime** - availability
- **Load** - host performance at scale
- **Logging** - application code
- **Metrics** - system resources, response times

What to Monitor?

- **Uptime** - availability
- **Load** - host performance at scale
- **Logging** - application code
- **Metrics** - system resources, response times
- **Performance** - all of the above

The Best Tool is ...

Metrics

- Types include counter, gauge histogram.
- Often **time-series** data
- Suited to triggering **alerts**

Blackbox vs Whitebox metrics

- **Blackbox** - external observations (resources, uptime, speed)

- **Whitebox** - internal observations (logging, databases, traces)

Logging

https://unsplash.com/@chanphoto

Logging Considerations

- **What** should you log?
- **Format** (plaintext, structured, binaries)?
- **Where** to store logs?
- **Who** will read these?

Effective Logging

- **Understand** your app thoroughly
- Become familiar with **Log levels**:

  DEBUG, INFO, WARNING, ERROR, CRITICAL
- Ensure your log statements have **context**

Python Logging Tips

- Use **__name__** as the logger name

```
logger.getLogger(__name__)
```

- Dump **tracebacks** in log message

```
logger.error('S3 API call failed', exc_info=True)
```

Logging and Metrics

- **Structured** logs can deliver more context than a single metric
- Turning a log entry into a metric can trivial
- Metrics generally have a **constant overhead**

Getting notified when things break

https://unsplash.com/@chuttersnap

Alerting - Policies

- **Who** should be alerted?

Alerting - Policies

- **Who** should be alerted?
- **How** are you alerted?

Alerting - Policies

- **Who** should be alerted?
- **How** are you alerted?
- Is there a **chain of escalation**?

Alerting - Policies

- **Who** should be alerted?
- **How** are you alerted?
- Is there a **chain of escalation**?
- How is **urgency** determined?

Alerting - Policies

- **Who** should be alerted?
- **How** are you alerted?
- Is there a **chain of escalation**?
- How is **urgency** determined?
- Is it clear what **action** should be taken?

Alerting - Fixing

- Is there sufficient **context** to debug the problem?
- Avoid **desensitizing** your team by over-alerting (alerting fatigue).
- What happens **post-fix**?

Traffic and Load Testing

https://unsplash.com/photos/Xbh_OGLRfUM

Traffic and Load Testing

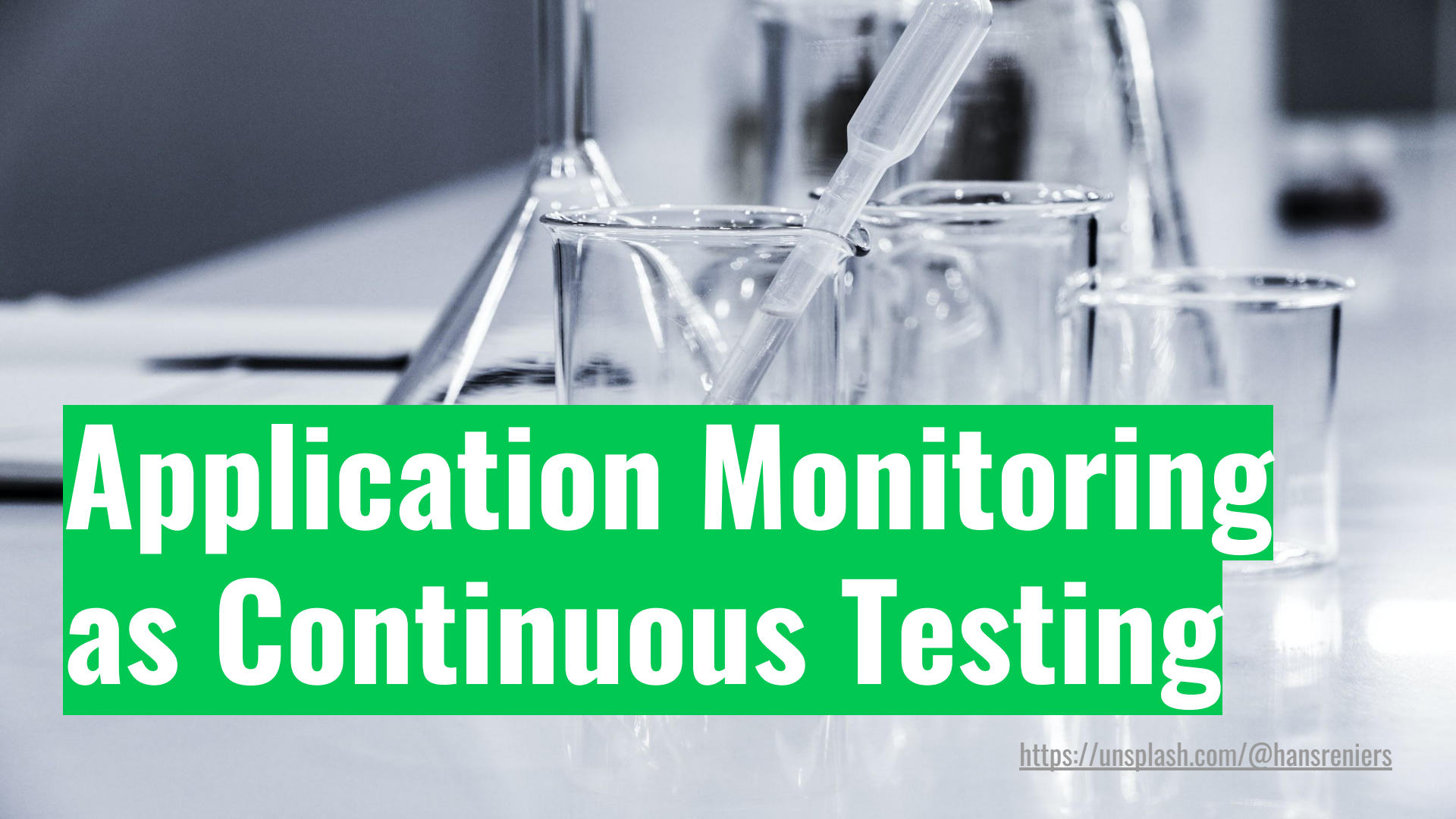
- **Traffic Monitoring** can aide historical debugging.
- **Load Testing** can help identify bottlenecks and determine baseline host capacity.
- Are you load testing the most **resource-intensive** APIs?
- What **metrics** are being gathered during testing?

But will it scale?

https://unsplash.com/@zmachacek

Microservices and Tracing

- Challenging to capture full picture in distributed systems using **logs**.
- Microservices are independent and tech stack can vary wildly.
- **Distributed Tracing** allows a single request to be followed through the system.



Application Monitoring as Continuous Testing

https://unsplash.com/@hansreniers

Takeaways

1. Have a clear idea of **what** you want to monitor.

Takeaways

1. Have a clear idea of **what** you want to monitor.
2. Quality **logs** provide context.

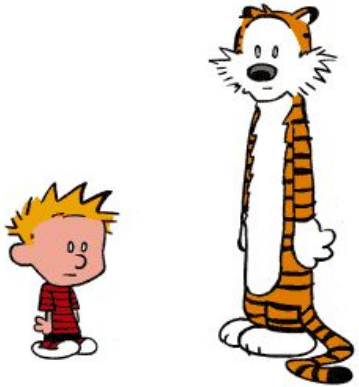
Takeaways

1. Have a clear idea of **what** you want to monitor.
2. Quality **logs** provide context.
3. Consider your response **policies**.

Takeaways

1. Have a clear idea of **what** you want to monitor.
2. Quality **logs** provide context.
3. Consider your response **policies**.
4. **Third party** monitoring tools can save time and increase insight.

We can do better.



@samclarkeg