

LESSONS FROM BUILDING A HIGH VOLUME DATA PROCESSING PIPELINE IN JAVASCRIPT

Seán Barry – Lead Software Engineer

About

Tech startup

Energy intelligence

London, Houston, Singapore



January 2021



Bloomberg

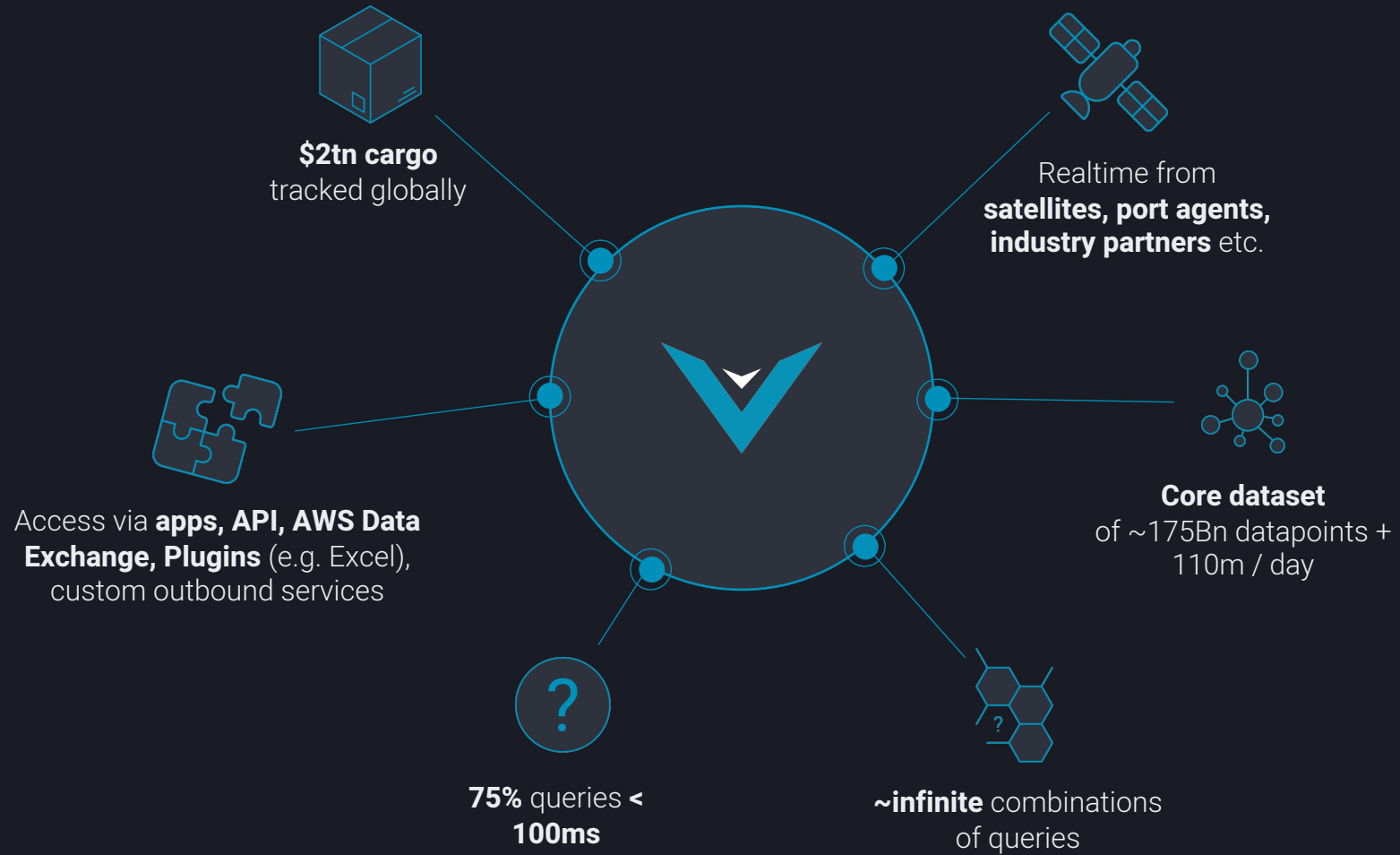


VORTEXA

vortexa.com

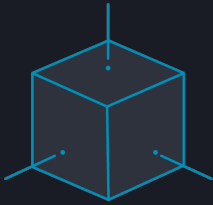
2

Platform



Engineering challenges

January 2021



Scale

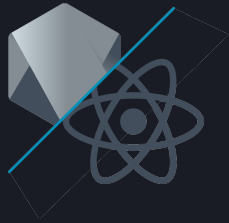


Performance

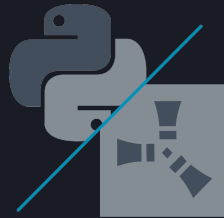


Usability

Core languages



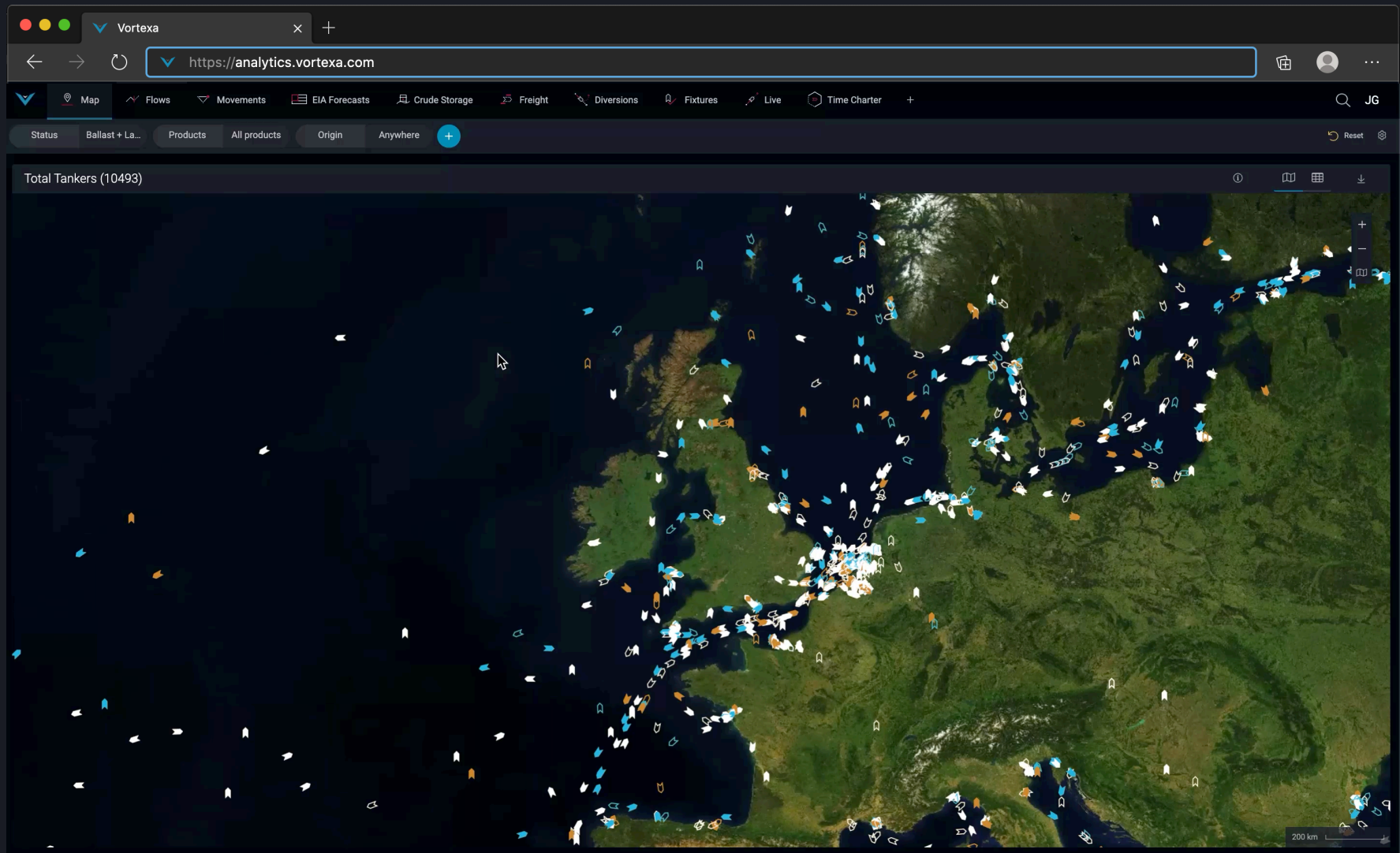
Node APIs,
React Apps



Python & Rust



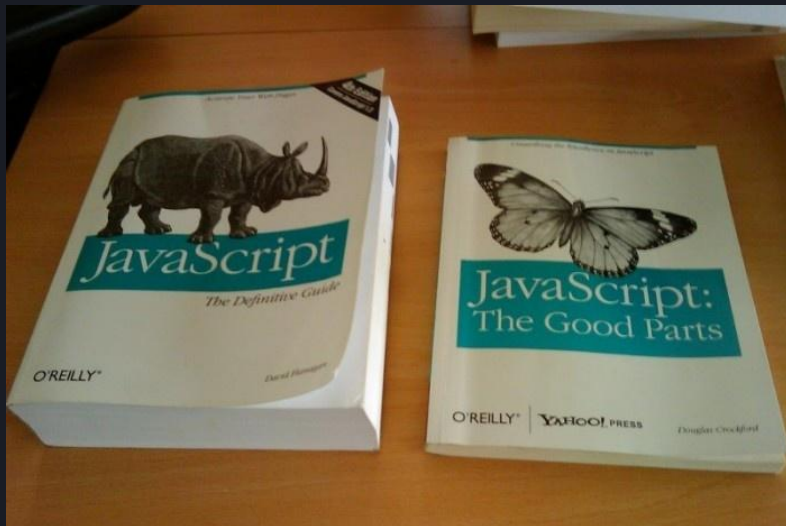
Java & Scala






January 2021

JavaScript's reputation



> typeof NaN	> true==1
< "number"	< true
> 9999999999999999	> true===1
< 10000000000000000	< false
> 0.5+0.1==0.6	> (!+[[]+[]+![]).length
< true	< 9
> 0.1+0.2==0.3	> 9+"1"
< false	< "91"
> Math.max()	> 91-"1"
< -Infinity	< 90
> Math.min()	> []==0
< Infinity	< true
> []+[]	
< ""	
> []+{}	
< "[object Object]"	
> {}+[]	
< 0	
> true+true+true===3	
< true	
> true-true	
< 0	

JavaScript reality

- Easy syntax
- Massive ecosystem (NPM)
- Typescript - static typing
- No context switch between front/backend
- Hire from same pool of people
- JSON – first class citizen

Node & V8

- Node - asynchronous event driven architecture
- V8 – performant, progressive



Not a deep dive in to JavaScript

- Not a study of JS internals
- Show what it's capable of
- Hopefully teach
- Hopefully inspire

The challenge

- New ETL (Extract, Transform, Load) pipeline in ADT
- Comprehensive data from incredibly smart colleagues
- We decide the schema
- Optimise querying

Why build this in JavaScript

- Pool of skilled engineers
- Ecosystem of interfaces and shared utils
- Upskill engineers

Mission: Extract, transform, load

January 2021

```
const myProcess = async () => {  
  const data = await getDataFromSource();  
  
  const validatedData = validateData(data);  
  
  const transformedData = transformData(validatedData);  
  
  await syncToStorage(transformedData);  
}  
  
myProcess();
```

Mission: ~~Extract, transform, load~~ failed

- Fetching all data?
- Tracking the data we've processed
- Transform 50x faster than data fetch
- Sync 2x slower than data fetch
- Inefficient design with mixed responsibilities

```
const myProcess = async () => {  
  const data = await getDataFromSource();  
  
  const validatedData = validateData(data);  
  
  const transformedData = transformData(validatedData);  
  
  await syncToStorage(transformedData);  
}  
  
myProcess();
```

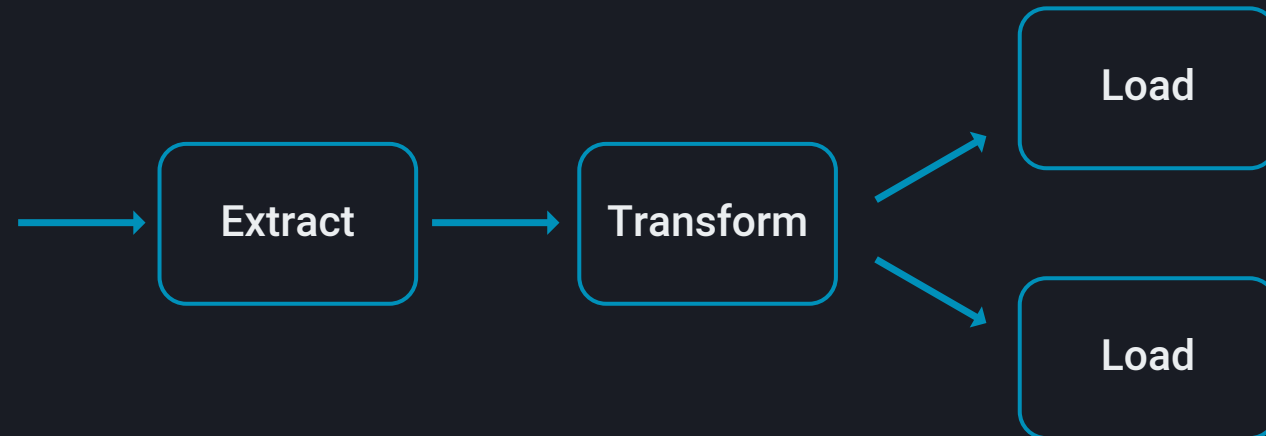
Design concerns

- Modular – delegated responsibility, other languages
- Robust – loosely coupled services
- Maintainable – independently built and deployed components
- Scalable – services independently scalable
- Auto-healing
- Trackable – telemetry to alert on issues

Mission: Extract, transform, load – v2

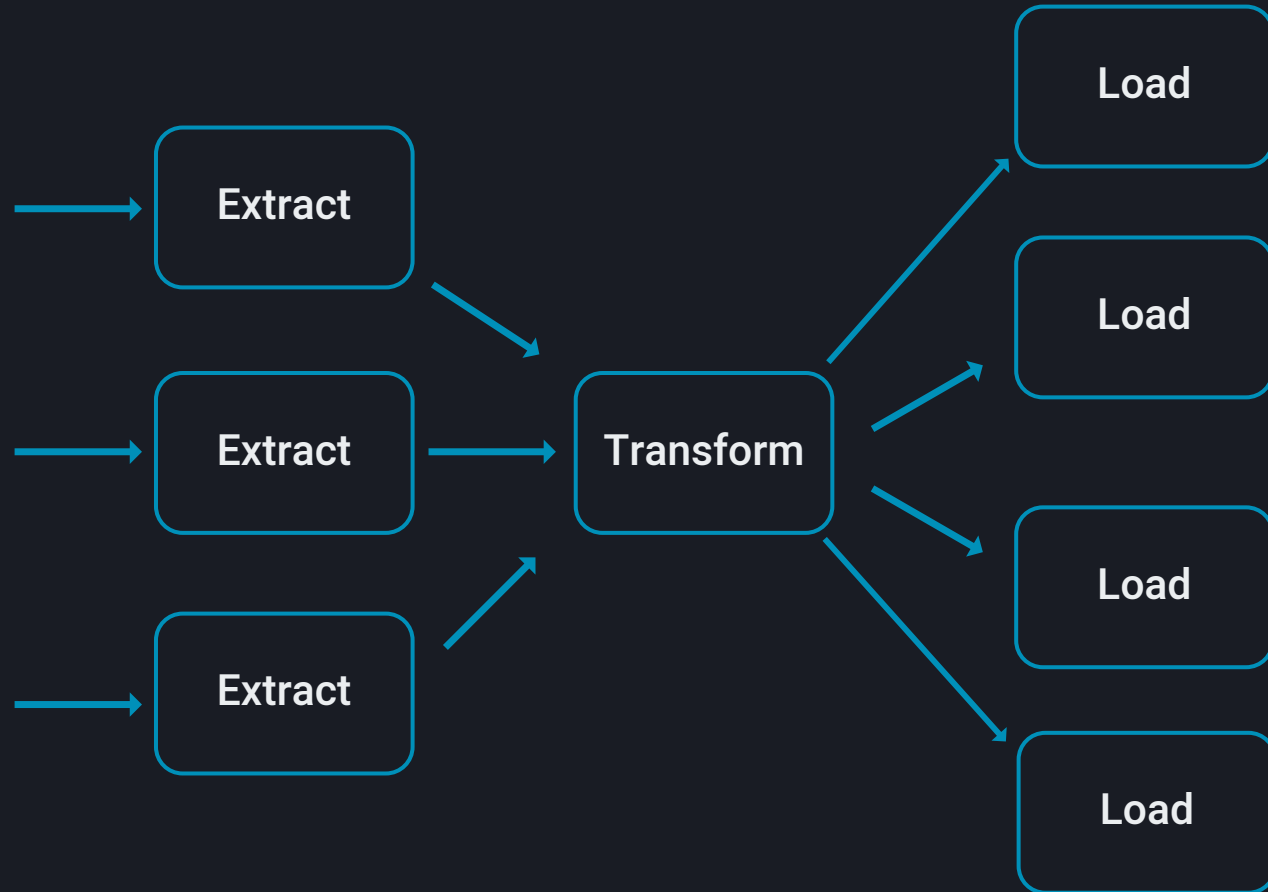
January 2021

- Split each task in to separate process
- Connect with messaging bus
- Scale each process as necessary

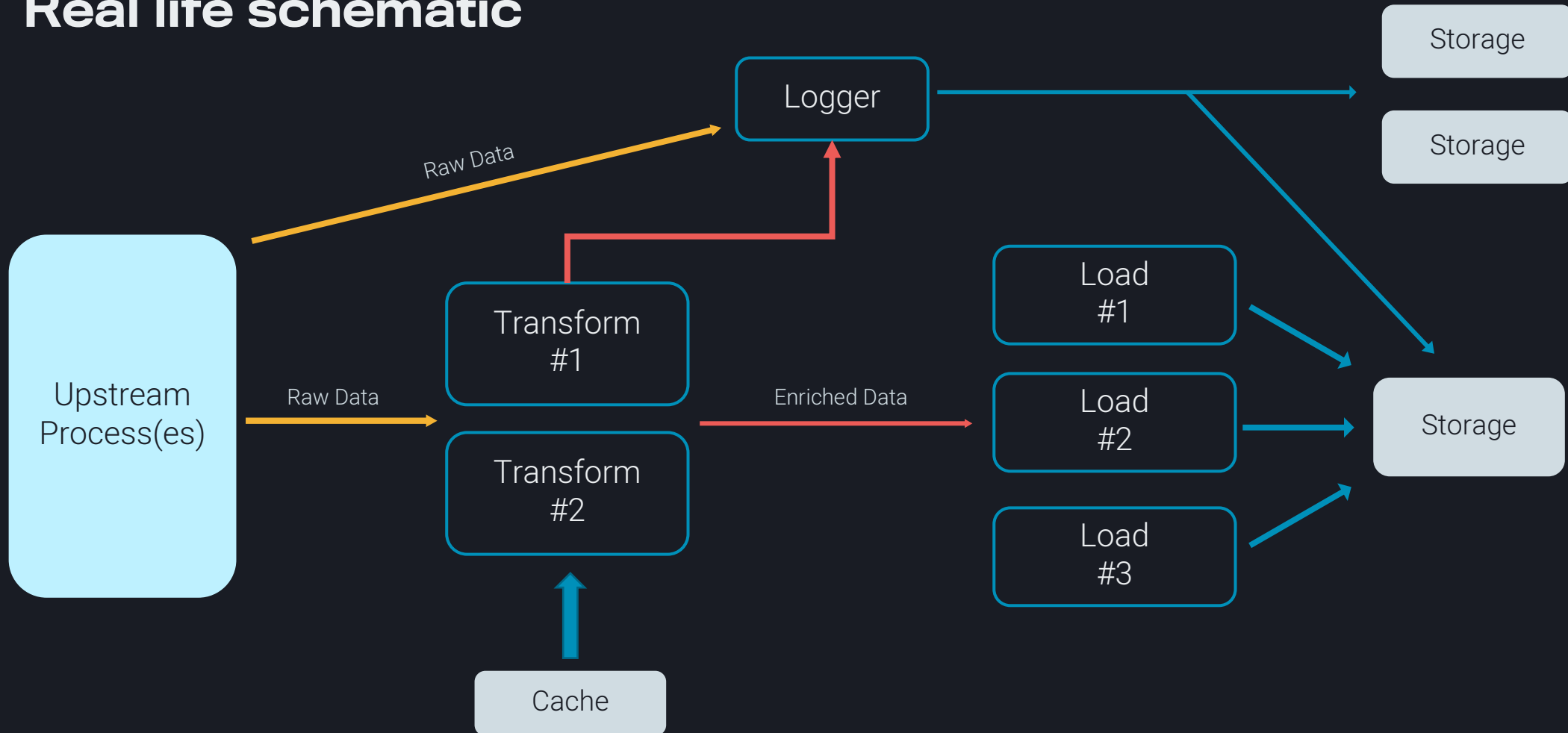


Stateless services

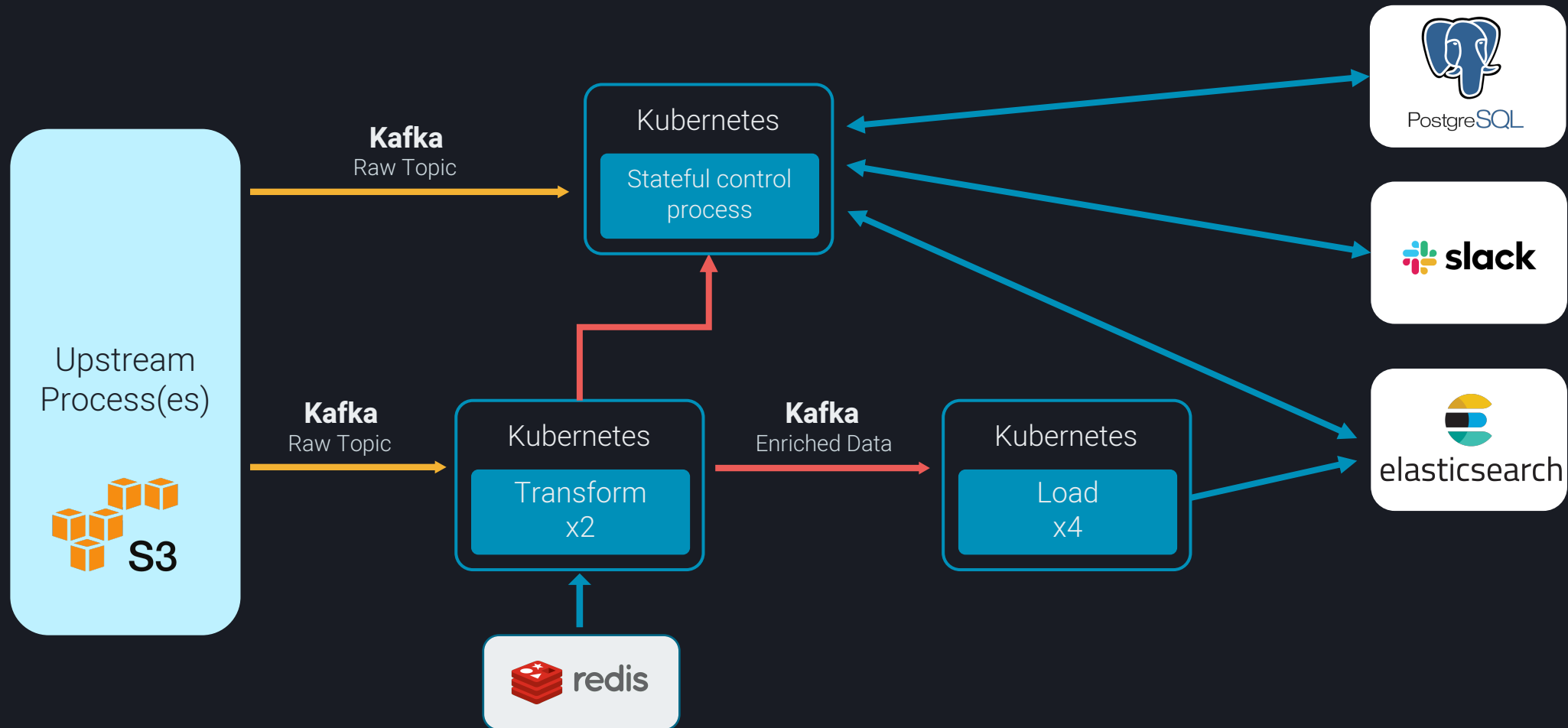
- No reliance on external processes
- Remember: infinite stream of messages
- Each process does a job and moves on
- One process dying has no effect
- Maximise throughput



Real life schematic



Simplified implementation



Collecting telemetry and alerting on failures

- Failure is inevitable
- Outages will happen
- Telemetry shouldn't be an afterthought



Writing robust code

- Operational vs Programmer errors
- Expect operational errors (don't trust anything)
- Handle operational errors appropriately
- Mitigate programmer errors (TS, tooling)
- Know your language!



Async pitfalls

- Avoid callback hell
- Async is syntactic sugar for promises
- Makes asynchronous code look synchronous
- Can be misleading/confusing
- Async pitfalls

```
const syncToDatabase = async () => {  
  /**  
   * Trying to do some syncing,  
   * but the DB has gone down...  
   */  
  throw Error("Oh dear!");  
}  
  
const processor = () => {  
  try {  
    syncToDatabase();  
  } catch (err) {  
    console.log(`Error: ${err.message}`);  
  }  
}  
  
processor();
```

```
► Uncaught (in promise) Error: Oh dear!  
  at syncToDatabase (<anonymous>:6:9)  
  at processor (<anonymous>:11:5)  
  at <anonymous>:17:1
```

Async pitfalls

- Avoid callback hell
- Async is syntactic sugar for promises
- Makes asynchronous code look synchronous
- Can be misleading/confusing
- Async pitfalls

```
const syncToDatabase = async () => {  
  /**  
   * Trying to do some syncing,  
   * but the DB has gone down...  
   */  
  throw Error("Oh dear!");  
}  
  
const processor = async () => {  
  try {  
    await syncToDatabase();  
  } catch (err) {  
    console.log(`Error: ${err.message}`);  
  }  
}  
  
processor();
```

Error: Oh dear!

• ► Promise {<fulfilled>: undefined}

• |

Event codes

- Enum/Dictionary of codes
- Not necessarily error codes
- Description of cause/location of error/event
- Easier to debug logs
- Used to create alerts from logs

```
export enum TransformerLogCodes {  
  /**  
   * tsfr-00 - UNCAUGHT PROMISE REJECTION  
   * this was caught at the top level of the process.  
   */  
  TSFR00 = 'tsfr-00',  
  /**  
   * tsfr-01 - Reading from redis has failed  
   */  
  TSFR01 = 'tsfr-01',  
  /**  
   * tsfr-02 - Raw batch received from raw topic  
   */  
  TSFR02 = 'tsfr-02',  
  /**  
   * tsfr-03 - Enriched batch produced to enriched topic  
   */  
  TSFR03 = 'tsfr-03',  
}
```

▶	2021-01-26T13:23:38.085+00:00	{
▶	2021-01-26T13:23:40.067+00:00	{
▶	2021-01-26T14:31:57.436+00:00	{
▶	2021-01-26T16:07:02.440+00:00	{
▼	2021-01-26T16:54:44.990+00:00	{
	<pre> { "v.meta.key": "Jan 26, 2021, 4:42:17 PM", "v.meta.offset": 21940344, "v.meta.message": "Raw batch received from raw topic", "v.code": "tsfr-02", "v.code.severity": "info" } </pre>	
▶	2021-01-26T16:54:44.990+00:00	{
▶	2021-01-26T17:08:12.418+00:00	{
▶	2021-01-26T18:00:58.627+00:00	{
▶	2021-01-26T19:51:50.647+00:00	{
▶	2021-01-26T19:51:50.647+00:00	{
▶	2021-01-26T20:48:13.200+00:00	{

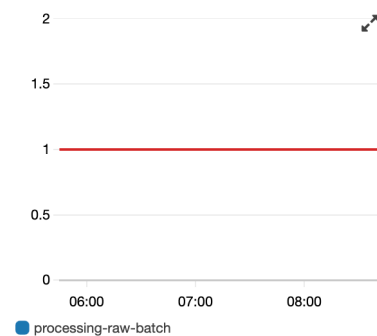
Specify metric and conditions

Metric

[Edit](#)

Graph

This alarm will trigger when the blue line goes below the red line for 1 datapoints within 1 minute.



Namespace

adt-publish-

Metric name

processing-raw-batch

Statistic

Average

Period

1 minute

Conditions

Threshold type

☒ Static

Use a value as a threshold

☐ Anomaly detection

Use a band as a threshold

Whenever processing-raw-batch is...

Define the alarm condition.

☐ Greater

> threshold

☐ Greater/Equal

>= threshold

☐ Lower/Equal

<= threshold

☒ Lower

< threshold

than...

Define the threshold value.

1

Must be a number

Things I wish I'd known before building this

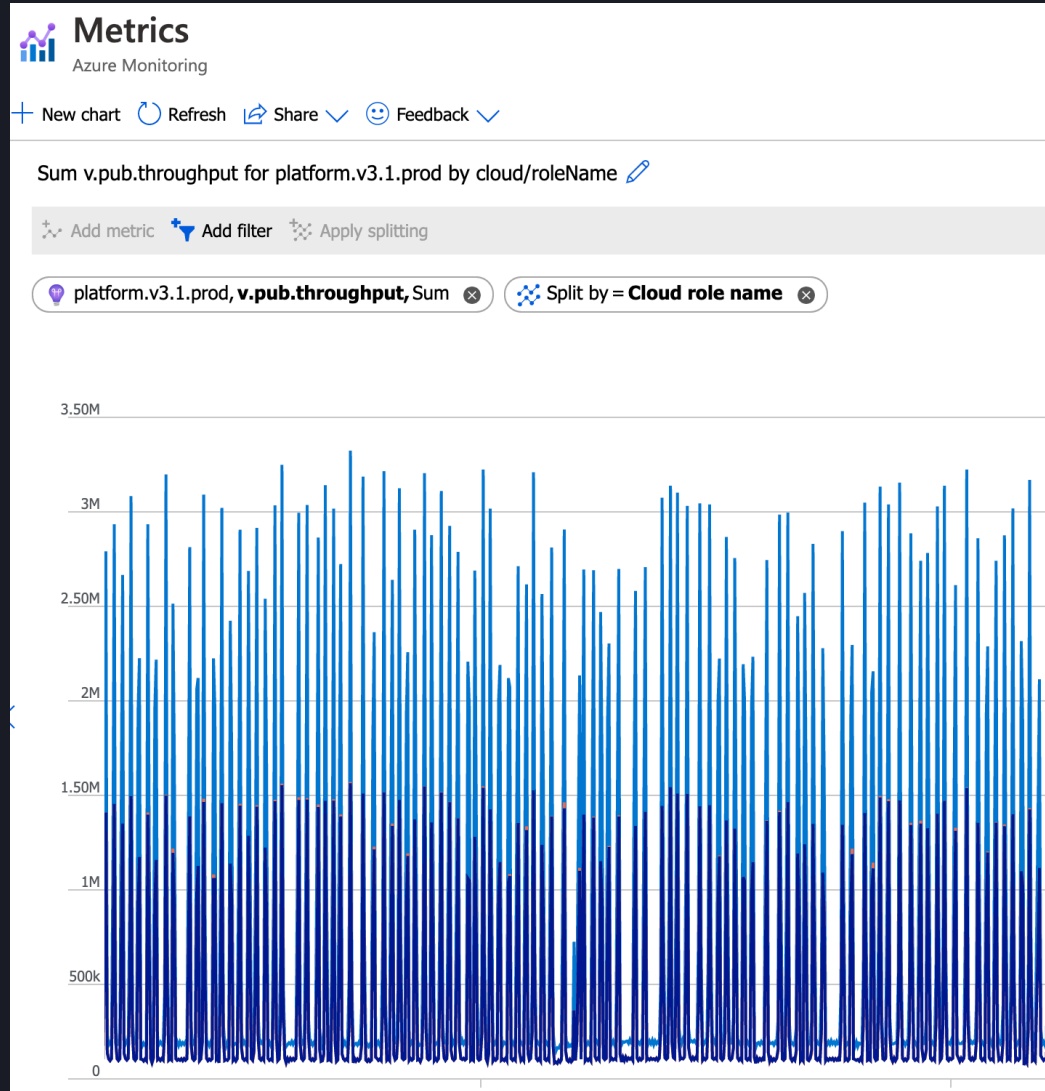
- Define schemas between services ASAP
- Gain familiarity with tools (Msg bus, DBs, Caches)
- Mock infrastructure locally
- Don't develop against prod data
- Mock all data and control volume
- Include bad data in mocks for testing
- Multiple devs working simultaneously (shared infra)
- Test unexpected events
- Configure dev/prod deployment in advance

Outcome

- Full control over schemas in storage
- Optimised queries – some an order of magnitude!
- Telemetry so comprehensive we detect upstream issues
- New datasets bring incredible possibilities
- Transform/Enrich = more comprehensive data for clients
- Opportunity for Vortexa engs to upskill
- Cool data engineering project to talk about at meetups



January 2021

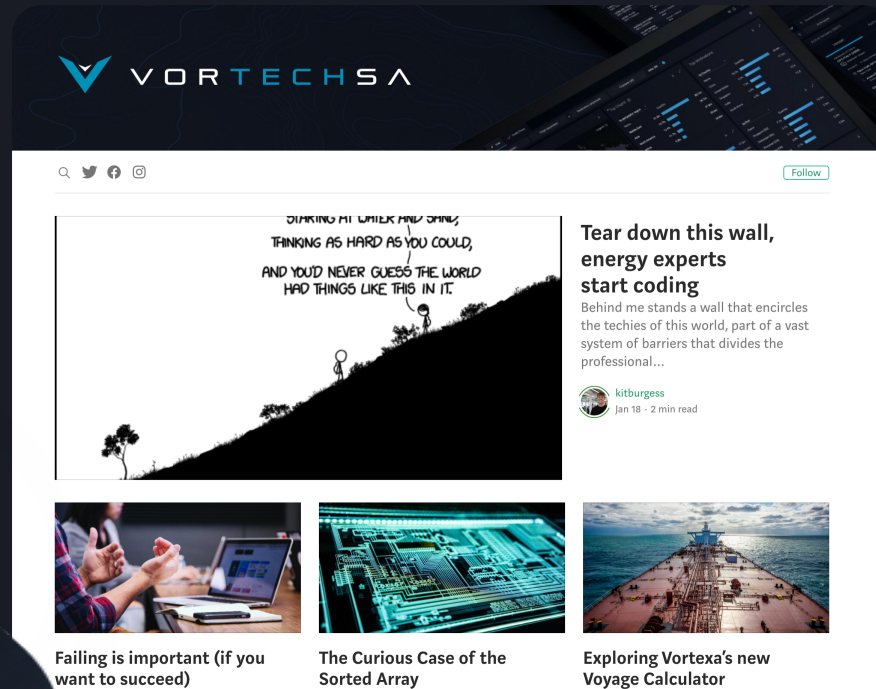


Summary

Typical engineering problems



Tech Blog

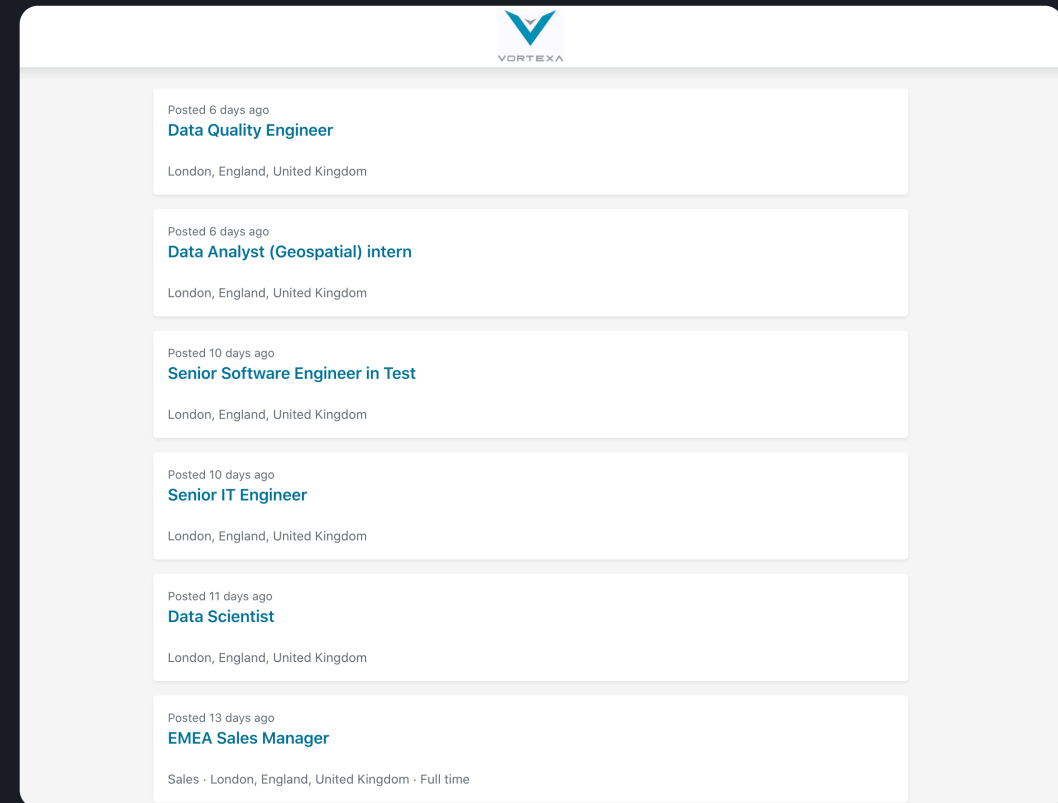


www.medium.com/vorTECHsa

Careers at Vortexa

- 1x Senior / Lead Full Stack Engineer
- 2x Junior / Mid Full Stack Engineer
- 1x Senior Engineer in Automation and Test
- We work in these areas:
- "Applications" - "APIs" - "Storage systems" - "Data engineering" - "Security and authentication"
- We will upskill you in all areas regardless of your current experience.

<https://www.vortexa.com/careers>



Thank you



VORTEXA

www.vortexa.com