**Table of Contents**

[Intro 2](#_Toc72700554)

[Guiding Principles 2](#_Toc72700555)

[Components 3](#_Toc72700556)

[Dashboard (Product) 3](#_Toc72700557)

[Averages 3](#_Toc72700558)

[Headers 3](#_Toc72700559)

[Colours 3](#_Toc72700560)

[Indicators of Travel 4](#_Toc72700561)

[Indicator of Confidence 4](#_Toc72700562)

[Charts 5](#_Toc72700563)

[Hover 5](#_Toc72700564)

[Performance Band Indicators 5](#_Toc72700565)

[Deployment Failures 5](#_Toc72700566)

[Time to Recover Chart 5](#_Toc72700567)

[Export 5](#_Toc72700568)

[Trendline 5](#_Toc72700569)

[Dashboard (Team) 6](#_Toc72700570)

[Averages 6](#_Toc72700571)

[Charts 6](#_Toc72700572)

[Multiple Products 6](#_Toc72700573)

[Trendline 6](#_Toc72700574)

[Deployment Info 7](#_Toc72700575)

[Headlines 7](#_Toc72700576)

[Commits 7](#_Toc72700577)

[Swagger 8](#_Toc72700578)

[Technical Design 9](#_Toc72700579)

[Agent 9](#_Toc72700580)

[Configuration 9](#_Toc72700581)

[Notable Third Parties 10](#_Toc72700582)

[Data Anonymisation 10](#_Toc72700583)

[Calculators 11](#_Toc72700584)

[Deployment Frequency 11](#_Toc72700585)

[Lead Time 11](#_Toc72700586)

[Change Fail Rate 11](#_Toc72700587)

[Time to Recover 11](#_Toc72700588)

[Averages 11](#_Toc72700589)

[Web Portal 12](#_Toc72700590)

[Data Persistence 12](#_Toc72700591)

[Flexibility 13](#_Toc72700592)

[Number of months 13](#_Toc72700593)

[Current Date 14](#_Toc72700594)

# Intro

I’ve got to a POC / MVP stage that I think conveys some ideas on how we can build upon on how we use our 4 key metrics to shape behaviours and work. Head over to <http://4km.raganmcgill.co.uk> for a sneak peek at the solution, <https://github.com/raganmcgill/Deployment-Insights> for the code and the rest of this doc for some of the concepts.

|  |
| --- |
| 😊 ALERT & DISCLAIMERS 😊  ! The code is pretty poor – cobbled together for proof of concept purposes only  ! The calculators need validating – they have been constructed by using basic assumptions and understanding – Let’s not take any decisions off the back of them  ! There is absolutely **zero security** present – everything is open (including the delete api calls).  This might seem irresponsible but worse things exist – beware :D !  (ssshh.. don’t tell anyone, but if you double click the **Admin** menu header, a super-secret *reset* option appears – which – deletes everything – HAHAHA) |

# SOME Guiding Principles

* Teams can have multiple products
* Products can be composed of multiple git repositories
* Individual deployment state is stored

# Components

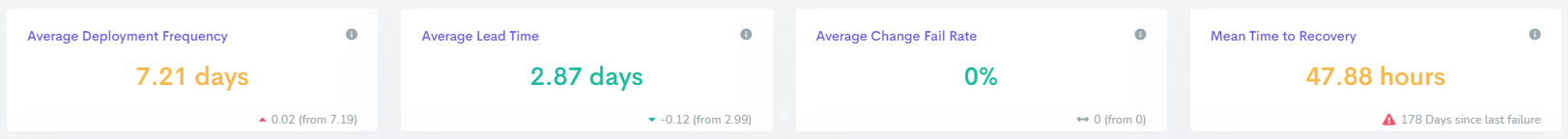
## Dashboard (Product)



The *product dashboard* is accessed via the menu by clicking on a product and is composed from a number of components

### Averages

The average section provides average for the four key metrics for a given product; average deployment frequency, average lead time, average change fail rate and mean time to recovery.



#### Headers

Clicking on the headers will take you to advice pages on how to improve – The pages exist but are not developed – demo only for concept purposes

#### Colours

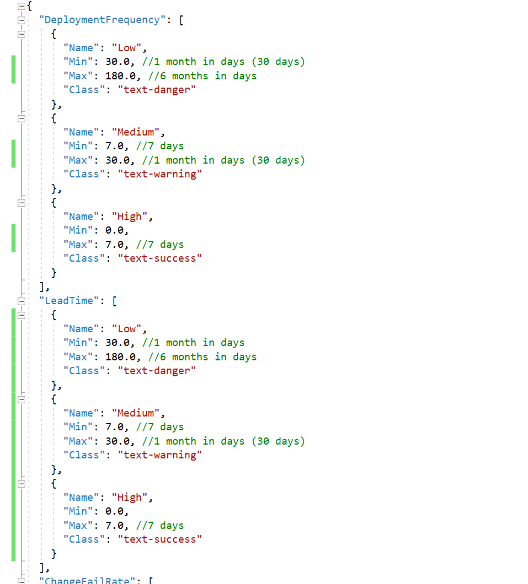
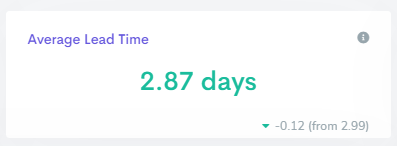
The values of the averages are visually representative of the performance bands in which they reside. In this example, both average lead time and average change fail rate are both to be considered as high and both average deployment frequency and MTTR are to be considered as medium. In the implementation, the performance bands are denoted in a *json* config file (example on the right..)

Figure 1: Example performance band config file

#### Indicators of Travel

In the bottom left of the card, there is a direction of travel indicator.



This indicator is relative to the previous deployment and will show whether the current average is lower, higher or equal to the previous values along with the change amount and previous value. Indicators of travel are applicable to deployment frequency, lead time, and change fail rate.

#### Indicator of Confidence



I’ve looked through a different lens for MTTR – as a thought experiment – as I’d like these figures to promote insights and thoughts. In this instance, it’s been 178 days since our last failure – something that can, and should, be looked upon as a success! So.. why the red triangle? If I think about why we track the time to recovery, I would suggest its strongly related to our confidence around recovery – but how confident can we be if the last time we recovered was nearly six months ago? I’d argue, not very.

I’m happy to admit that there are far better ways to represent our recovery confidence – and I’m not advocating that we *promote failure* – but acknowledging that it’s been nearly 6 months since our last failure, teams could ask themselves how confident they about recovery should a failure happen. Are there any processes in place? Are there opportunities to practice recovery - safely? Release the chaos monkey?

### Charts

The 4 charts represent the 4km data at each deployment stage. These are actual figures and not averages



#### Hover

Hovering over the deployments, highlights both the date and the metric – an example of this can be seen in the deployment frequency chart where on the *30th Mar* the deployment frequency was *7.55 days*

#### Performance Band Indicators

On each graph, there is a green horizontal line indicating the high threshold (as specified in the aforementioned config json file)

#### Deployment Failures

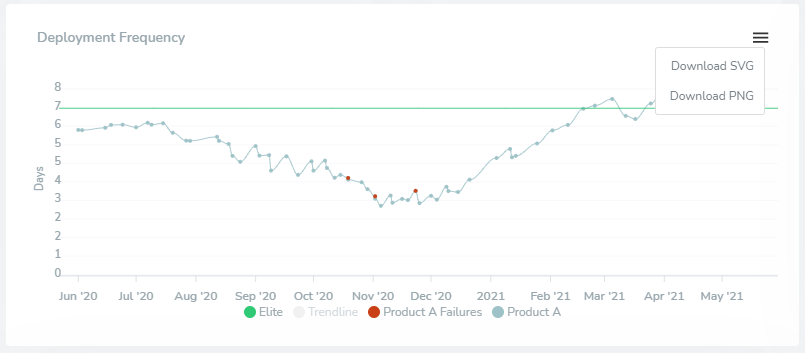
All charts display failures – represented by red data points

#### Time to Recover Chart

This chart is different to the other three and instead of a line graph, the data is represented in the scatter format. Each data point represents an individual failure and the time taken to restore

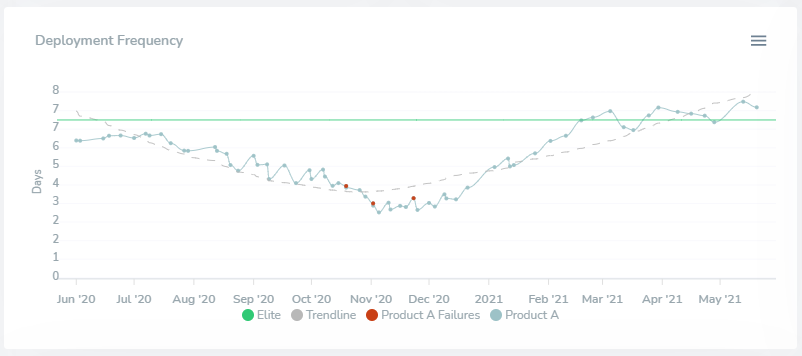
#### Export

Each chart can be exported to either an *svg* or *png* file



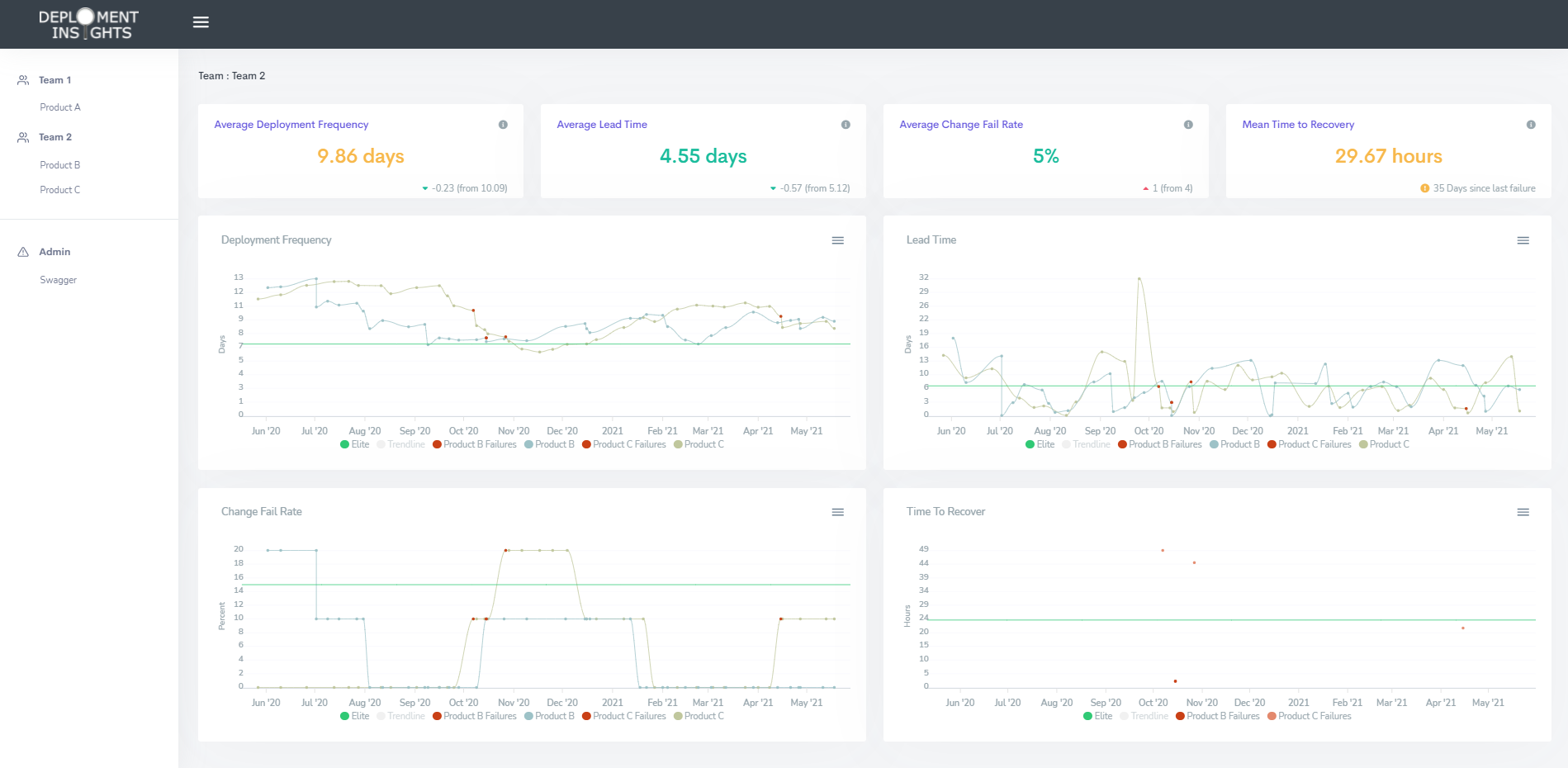
#### Trendline

Each chart has a trendline which is represented by a dashed grey line. Denoted as a separate series and turned off by default (for visual cleanliness). Can be toggled on by clicking on the series in the legend (as can any series)



## Dashboard (Team)

If team looks after a single product, then the team dashboard will look exactly like the product dashboard. This dashboard is particularly prevalent to teams that have responsibility over many products as it shows an aggregated view.



The *team dashboard* is accessed via the menu by clicking on a team and is similar to the product dashboard but with a few differences

### Averages

The averages section on this dashboard is calculated by using the combined deployments of all products – representing the four key metrics for the team as opposed to a single product

### Charts

#### Multiple Products

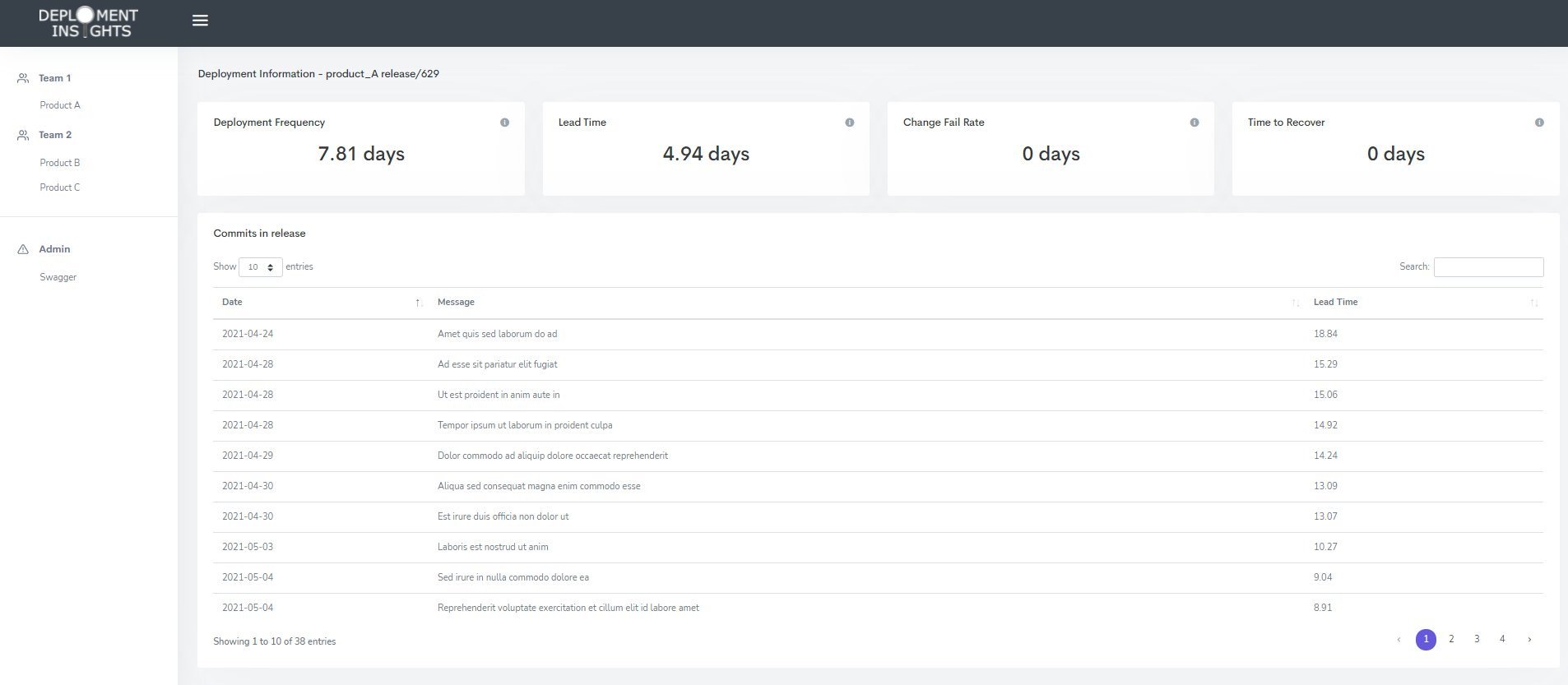
The charts display all products (along with their failures) for the team

#### Trendline

The trendline is operationally similar to the products trendline but, similarly to the average, is representative of the combined deployments for all products

## Deployment Info

Some information on the deployment can be seen by clicking on any of the data points on the graph. This page is valuable to help teams understand both how the four key metrics are calculated and the commits that help shape these figures.



### Headlines

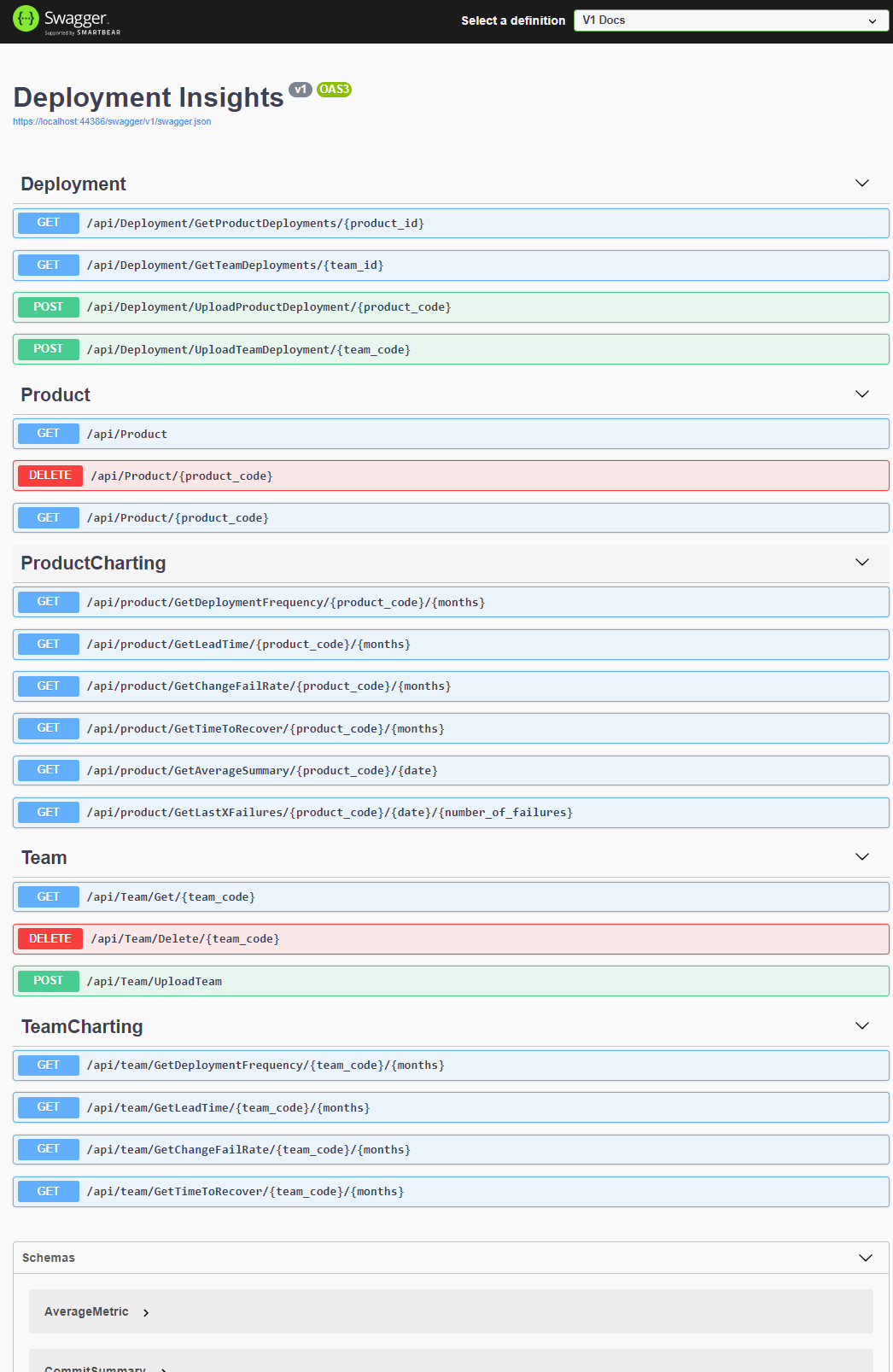
This view will display the four key metrics for the deployment at that point in time. These are the individual figures and not averages.  
There are tonnes of possibilities here if found valuable – Things that could be included are the relative average figures at that time, colour coding (as on the dashboards), direction of travel, etc

### Commits

All of the commits that are pertinent to the deployment are presented in a data grid for the use to page through and inspect.

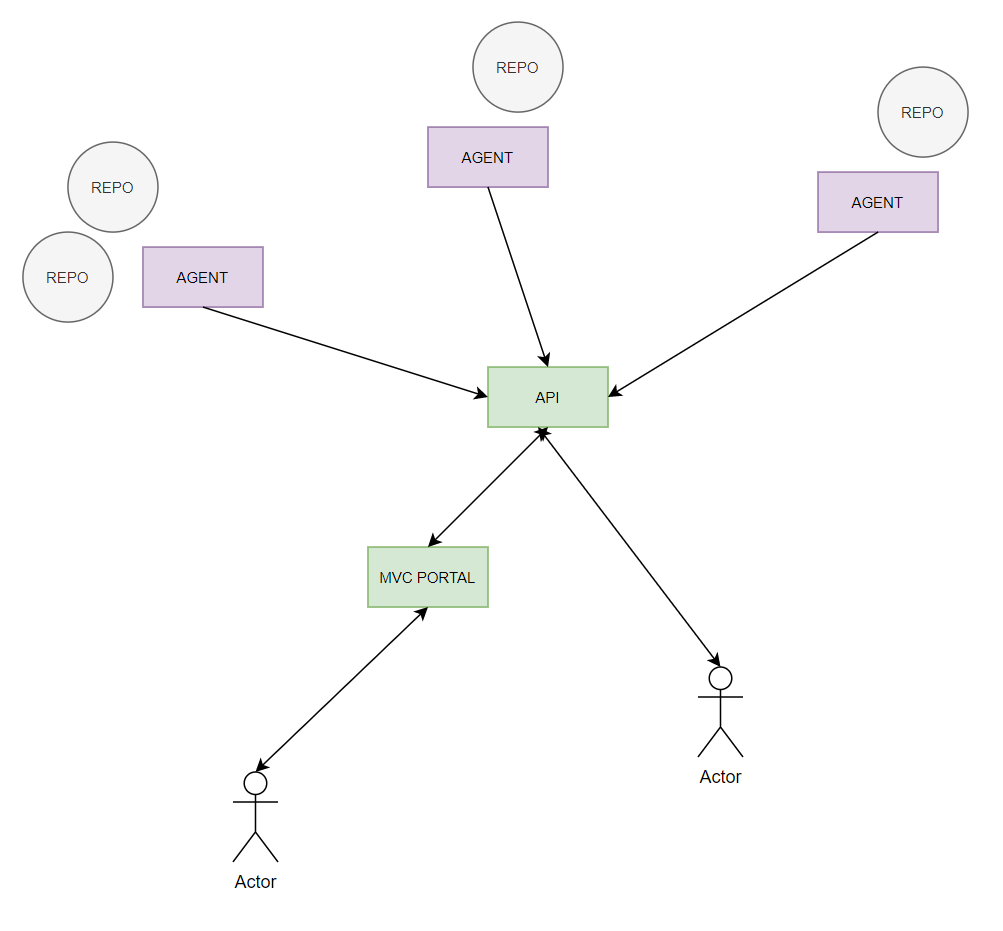
## Swagger

The site has a number of http interfaces that the frontend uses which can be found and explored in the swagger documentation (ish) by either clicking on the swagger menu item or navigating to/swagger



## Technical Design

Very rudimentary design in keeping with a POC or MVP largely comprising of 2 parts; a *frontend* web portal and a remote agent.

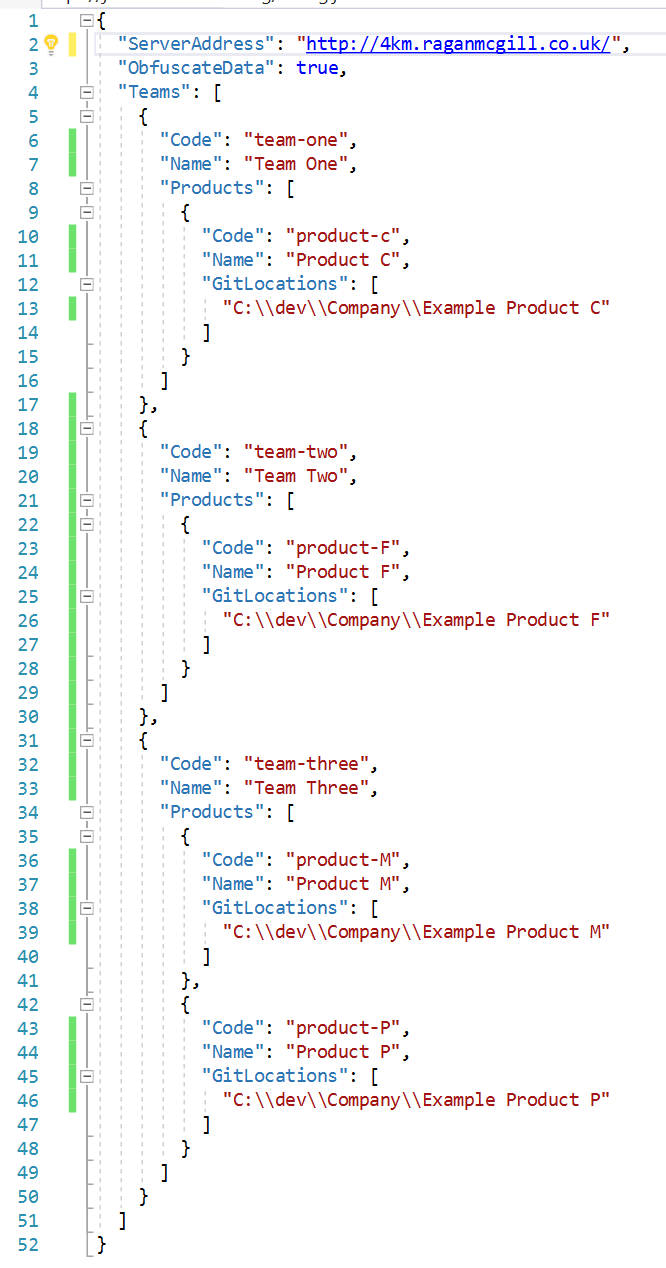


### Agent

The agent is a simple .net core *etl* console app and whose primary function is to traverse a git repository, extract deployment information, transform the data into a construct that’s useful and finally to load it to the portal via rest endpoints.

#### Configuration

Configuration is pretty noddy and is done in the config.json file using a simple json object that represents the structure of teams, products and git repositories. (hopefully self-explanatory)



Things to note in this config file are:

* Teams can have many products
* Products can have many reposititories

#### Notable Third Parties

There have been some notable third party libraries that have made this journey really really easy! To mention some:

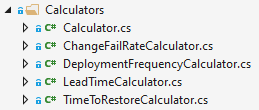
* **LibGit2Sharp** is used to inspect and extract the various git repositories
* **Newtonsoft.Json** is obviously a must 😊
* **RestSharp** is used for the loading to the portal through api requests, etc

#### Data Anonymisation

Naturally, this ETL tool traverses live repositories but for demonstration purpose data anonymising can be turned on via the aforementioned config file (ObfuscateData) – handy for workshops, demos, etc. This function allows the extraction and transformation of the data but prior to the loading of such, it will anonymise the teams, the products and the commits.

#### Calculators

The tool has representations of each 4km calculator that it applies to both products and teams



##### Deployment Frequency

On every deployment, the agent will calculate the deployment frequency (at the point in time) by:

* Looking over the last *x* deployments
* Calculating the timespan between the date & time of the deployment in question and the date & time of the deployment *x* before
* Using this timespan, the frequency is calculated by dividing the duration by the number of deployments (*x)*

The number of deployments to use in this calculation is configurable – I’ve opted for 10 in my current implementation

##### Lead Time

Lead time is determined by the duration of time between the initial commit and the deployment date – in lieu of not having a firmer, consistent identifier such as jira workflows.

On every deployment, the agent will calculate the average lead time by:

* Retrieving all commits that are associated
* Calculating the lead time of each commit by calculating the time between the commit date and the deployment date
* Order the commits by their individual lead times
* Taking the median

By taking the median, outliers are ignored

##### Change Fail Rate

Failed releases are identified by determining if the subsequent release is a *fix* releases – as determined by the annotation

On every deployment, the agent will calculate the change fail rate by:

* Looking over the last *x* deployments
* Calculating the percentage of failures to successful within the last *x* deployments
* Using this timespan, the frequency is calculated by dividing the duration by the number of deployments (*x)*

The number of deployments to use in this calculation is configurable – I’ve opted for 10 in my current implementation

##### Time to Recover

On every deployment, the agent will calculate the time to recover by:

* Determining if this deployment was classified as a failure – done so by peeking ahead and identifying whether the next deployment was marked as a *fix*
* If it was a failed deployment, the agent will calculate the time taken between this deployment and the next; done so in hours

##### Averages

On all deployments and for each of the 4 key metrics the averages are calculated by:

* Looking over the last *x* deployments
* Summing the *relevant metric* across these *x* deployments
* Dividing the *summed amount* by *x*

The number to determine the number to examine as an average is configurable – I’ve opted for 10 in my current implementation

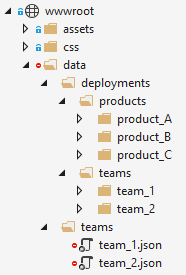
### Web Portal

A simple ASP.NET MVC Core web app exposing a interactive frontend and a variety of APIs.

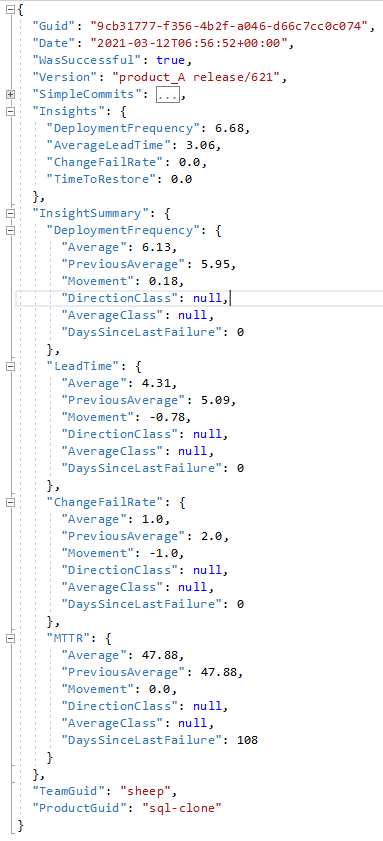
#### Data Persistence

Putting a DB on this app would not be hard in any way, shape or form. But to keep things simple, I’ve simply used serialisation and deserialization of files – as it’s good enough for now – certainly good enough to convey concepts and ideas 😊

The data is persisted to folders and files within the *data* folder under wwwroot and as mentioned are simply serialised versions of the programming objects.



The folder itself is not tracked by git and although having the data in this format is poor on performance and resources, the hit isn’t too noticeable. Separate deployment files are stored for both the product and teams as the meta data and averages vary between these lenses. Deployments are stored as separate files and present the insights at that moment in time



*Team* data, again, is a simple representation of the data that is stored in the configuration file of the agent.

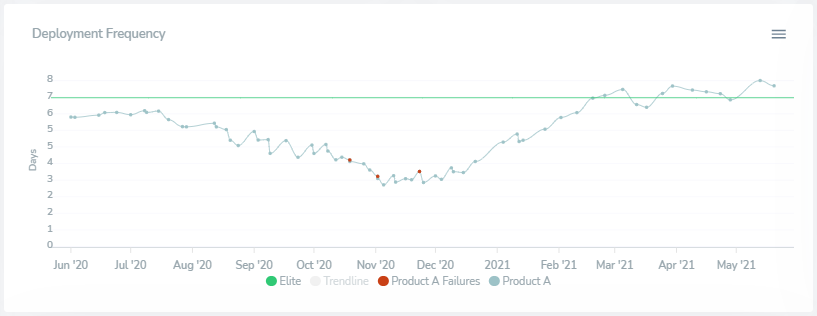
#### Flexibility

There are some data interrogation parameters that could be used but aren’t built into the GUI yet. For example:

##### Number of months

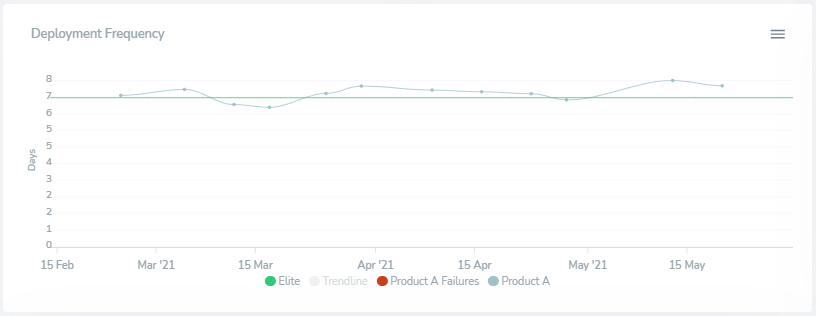
By default, the number of months to be displayed on the graph is 12 and can be seen in the querystring





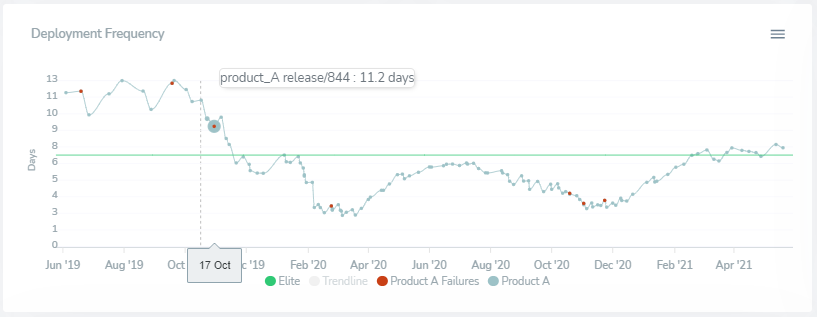
But querystrings can be manipulated 😊. You could, for example, only want to see the last 3 months…





… or, say the last 2 years…



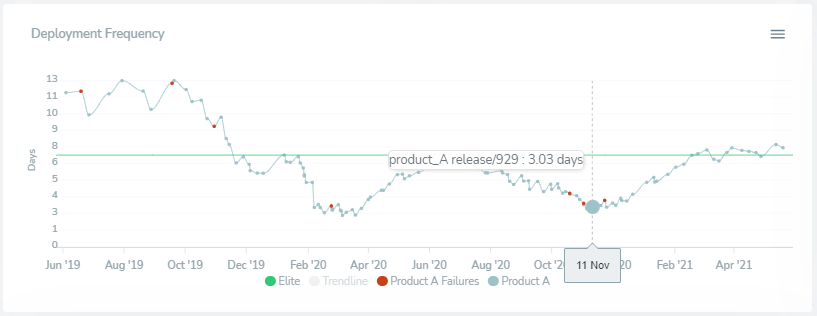


##### Current Date

Although not as obvious as the *number\_of\_months* the same principle can be applied for the reporting date. The app takes a default date of *today* but this date can equally set using the query string.

For example, taking a *recent* view using 21st May 2021 we can see this:





But if we wanted to snapshot it and explore in the ‘dip’ we could take ourselves back to 11th November 2020



