# Dynamic Performance Framework

Dean Gaffney

November 30, 2017

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Purpose of Iris

The aim of this project is the design a full implementation of a system for application performance monitoring. The proposed system, has the working title, Iris.

On completion, Iris will provide users with a dynamic performance framework which will allow them to fully customise and centralise their application performance monitoring. This will be achieved through a web interface where a user can specify a schema for a specific application they wish to monitor. Once a schema has been set up, a REST endpoint will be generated for the application. This endpoint will allow a user to send their monitoring data from their desired application to the framework in the form of JSON (matching the specified schema). Iris will also contain features which allow a user to monitor and analyse incoming data, using an intelligent, fully customisable graph and dashboard builder. Iris will then visualise any received data in real time using to the appropriate dashboards using websockets. Iris will come with some out of the box scripts/applications that users can use to monitor typical tasks such as JVM (Java Virtual Machine) performance, Linux OS System Performance.

## 1.2 Motivation for Iris

Onaware are IAM (Identity and Access Management) specialists.

Onaware is an international company that deal with IAM and have offices of 20 staff in Waterford.

More information on Onaware can be found here https://onaware.com/

The motivation for this project comes from database and system performance issues that Onaware has experienced in recent projects. It is often the case that they must deal with large amounts of identity data being aggregated into a third party system called IIQ.

Onaware has faced major issues with aggregating data in the past, in some cases it was taking up to five days, and sometimes they would fail halfway through meaning aggregations would have to be restarted, due to the amount of software involved it is hard to pinpoint what software is causing the issue.

In one such instance of aggregating data issues several attempts were made to rectify the performance issue such as optimising sql queries, increasing ram, multi threading tasks and increasing disk space, none of which worked. Due to the performance issue the IIQ instance became unusable so debugging the issue was not possible from inside the application and log files became so big that text editors would crash when trying to open them. In this case the issue turned out to be a customer putting size constraints on the database storing the aggregated data. While monitoring would not prevent such a mistake it would have reduced the time needed to locate the issue.

In response to difficulties in identifying performance issues Onaware have tried to monitor specific application elements. The aim at the time was to try and combine SQL, JVM and Operating System scripts to track the performance of the tools, however this approach is not very scalable and it would need to be reconfigured for future projects.

Iris is attempting to solve this problem. Iris will allow a user create a new application monitor with little effort using a web interface, give the user a REST endpoint specific to the application for their scripts to target their data, and allow a user to monitor the data in real time using graphs and dashboards. The aim is to make the framework as flexible as possible and not specific to the issue Onaware faced, meaning a user can monitor any data

they want from any application they want all they must do is send their data to a REST endpoint.

Users of Iris will consist of Onaware developers who will be monitoring IAM project data and generic tools which may be released to clients at a later time.

# Chapter 2

# Specification

## 2.1 Description

Iris will act as a web interface for a user to create an application monitor and allow the user to query and create personalised dashboards of their data through the use of Elasticsearch. A user may setup an application schema definition within Iris that matches the data they wish to monitor, a schema will consist of field names and corresponding data types specific to the application. Using the schema Iris will know what data to expect from the user. Once a schema is in place, Iris will generate a unique endpoint associated with the schema, this unique endpoint will be given to the user as a means of sending data to Iris. Data sent to the schema endpoint will be in JSON format and will conform to the schema definition created by the user in Iris.

A user creates an application monitor for an SQL database, they may create a schema like the following:
Schema Name: “SQL Monitor”
Schema Fields:
-field name: “writeSpeed”, fieldType: “double”
-field name: “tableName”, fieldType: “String”
Iris will then expect a json object to come back in the form:

```
{
        "writeSpeed": 3000,
        "tableName":  "students"
}
```

Figure 2.1: Example Schema Created for Iris

Iris will take the users data and create data mappings (Elastic.co, Mapping) inside Elasticsearch, as well as insert any incoming data into the correct Elasticsearch index (Elastic.co, Basic Concepts). With a schema in place a user can route their data through Iris; turning Iris into a centralised area for monitoring application performance data. With Iris being the centralised location to route and view your data a user can write a data transformation script for incoming data. The advantage of this is that it can help reduce the need for applications being redeployed to view new data or to transform data.

The user releases their application, and it is downloaded by 500 people. This data is now being sent from 500 instances of this application. To make any change to this data the developer must add in their desired field and and redeploy the app, those 500 users would then need to download an update for the application in order for it to take effect. The original JSON object passing through Iris looks like the following:

```
{
        "firstName": "Dean",
        "lastName":  "Gaffney"
}
```

In this example lets say the developer prefers to have the data mapped to a field called fullName which is all lowercase, the developer wants to avoid having to redeploy the app for one single field, instead the developer goes to Iris and applies a script to the schema. By running the script on incoming data the developer has made their desired change in a central location with no redeploys. The data may look like this after the developer has applied the script to the data:

```
{
        "firstName": "Dean",
        "lastName":  "Gaffney",
            "fullName":  "Dean Gaffney"
}
```

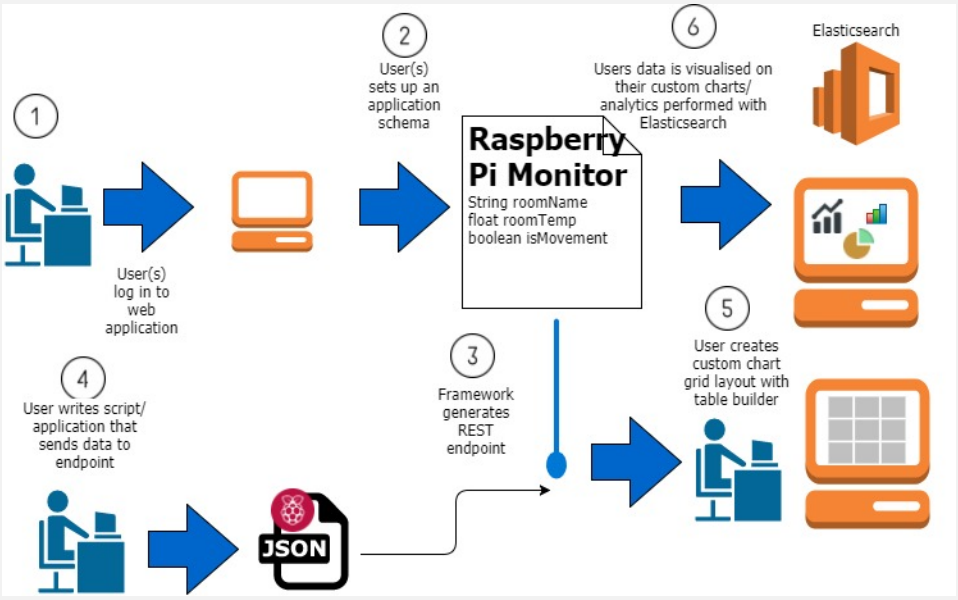
Figure 2.2: Iris Transforming Data

To aid performance monitoring, Iris will allow users to create personalised dashboards where they can create charts from their data and place them in the dashboard. This allows each user to have their own set of visualised data relative to them. To help a user see their data and charts rapidly an Elasticsearch aggregation (Elastic.co, Aggregations, 2017) playground will be put into place in order to allow a user create charts and get results back immediately, this will help a user to plan their dashboard charts before setting them up. The playground will also allow a user to chain several aggregations together which will allow them to create complex queries without having to have any prior knowledge of how Elasticsearch works.

Iris will also come with some common scripts that will be downloadable for

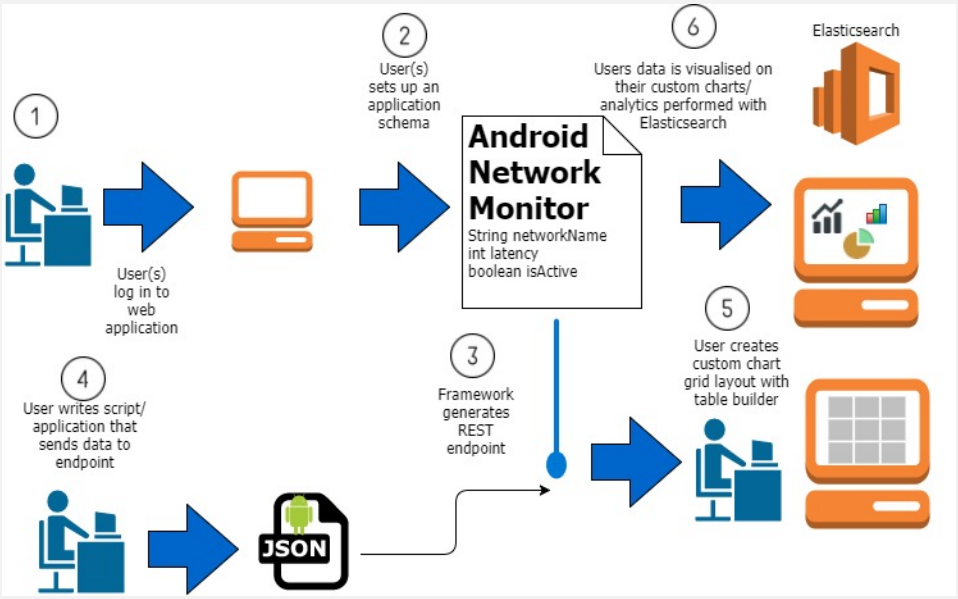
users to use straight away, some of these scripts will include JVM monitoring which can be placed into a java application, web page statistics which are retrieved from using Selenium web driver. Not all of the pre packaged monitoring scripts have been decided at this time, as the web application must be in place first.

## 2.2 Use Cases




In this example the user creates a schema in Iris for a raspberry pi. The user has a raspberry pi set up with a sensor to act as a home alarm. The user has written a script on the pi to send JSON data to the schema endpoint in Iris containing room name, room temperature and if there is any movement in the room. Data is being sent every five seconds to Iris. The user then logs into Iris and creates a dashboard for visualising the raspberry pi data.

Figure 2.3: Raspberry Pi Monitor for Iris




In this example the user creates a schema in Iris for an android application. The android application is sending network data to the Iris generated endpoint, the user sends the network name, latency and if the network is currently active in the form of a JSON object, every five seconds. The user then logs into Iris and build a dashboard to visualise the android data coming into Iris.

Figure 2.4: Android Network Monitor for Iris

# Chapter 3

# Relevant Technologies

In this chapter the technologies that will be used and the rationale for their selection, in the development of the Iris system are outlined. One section in particular Generating Dynamic Elasticsearch Queries is long but is important because the choice of technology is not obvious and the advantages/disadvantages of the alternatives is of particular importance to Onaware.

## 3.1 Grails

Iris will be built using the MVC web framework Grails. Grails is suited to for Iris due its rapid scaffolding capabilities, meaning a web application with CRUD (Create, Read, Update, Delete) actions can be implemented extremely fast. Due to the scale of the project a framework which will provide a fast POC (Proof of Concept) is needed, due to Grails command line interface a web application skeleton can be auto generated in a couple of commands.

Also a significant factor in selecting Grails is that Onaware have several of their applications written using Grails and Iris will have a new lifecycle after this project cycle in which Onaware developers will be extending upon Iris.

## 3.2 Groovy

Groovy is used by default within the Grails framework, this is a major factor as to why the technology is being used in Iris. However due to Grails being built on the JVM pure Java code could be used instead of Groovy, the major advantages of using Groovy over Java is its expressive nature and syntactic sugar, which allow you to write less code and do more. Groovy integrates seamlessly with any Java library, which is a major factor for using Groovy in Grails. Due to the popularity of Java, if Grails doesnt have a native plugin for some needed functionality, a Java library more than likely exists to provide the required functionality, this library can then be used with Groovys expressive syntax.

```
//dynamic typing, will be cast as a string
def name = 'Dean'

//built in regex support
def matches = (name ==~ /\w+/)

def message = (matches) ? "Matches" :
                "No match found in string $name"

println message
```

Output: Matches

Figure 3.1: Example of Groovy Syntax

## 3.3 Spock

Spock (http://spockframework.org/spock/docs/1.1/getting_started.html) is the default testing framework for Groovy and due to Grails using Groovy as its primary language. All unit and integration tests are written using the Spock framework. An advantage of the spock framework is that assertions are written in a behavior driven manner, which makes tests clear and concise unlike other JVM testing frameworks such as JUnit which relies on simple assertions.

```
void "test delete Schema"(){
        setup:
        setupData()

        when: "I delete a Schema"
        schema.delete(flush: true)

        then: "It is deleted from the database"
        assert Schema.count() == 0
}
```

Figure 3.2: Example of Spock Test in Iris

Peter Niederwieser has a concise introduction tutorial on Spock at the following link http://spockframework.org/spock/docs/1.1/introduction.html

## 3.4 Elasticsearch

Elasticsearch is a RESTful based search engine for indexing documents of data. Elasticsearch is suited for Iris as it accepts structured data in the form of index mappings, as well as unstructured data.

```
{
   "mappings":{
      "tweet":{
         "properties":{
            "message":{
               "type":"text"
            }
         }
      }
   }
}
```

Figure 3.3: Example of Spock Test in Iris