DC4400 Project Report

Extension of the BBC iPlayer off product schedules system to process change notifications

Oliver Matthew Bowker (220263618)

February 15, 2024

Contents

List of Figures

List of Tables

1 Executive Summary

2 Introduction

The BBC is now over 100 years old (BBC, 2022) and is well known for it's TV channels and radio stations that are broadcast over the airwaves (Pilnick, Baer, 1973, p.3) to peoples homes across the UK. However this old way of broadcasting, sending out airwaves on a certain frequency to an antenna, is becoming less popular in the modern age of the internet. A study done by Ofcom showed that people *'watched on average about 16% less broadcast TV between 2019 ... and 2022'*, with viewing *'decreasing by 47%'* (Ofcom, 2023, p.7) between ages 16-24. In addition another study carried out by media analyst firm Ampere found that in 2021 37% of people claimed to watch no linear TV, this increased to 45% by 2023 (Ampere Analysis, 2023).

This fall correlates with the significant rise in internet enabled TVs in the home, with statista finding that *'In 2014 just 11 percent of households in the UK owned a Smart TV, whereas, in 2023, nearly 74 percent of households reported owning a Smart TV.'* (Statista, 2023).

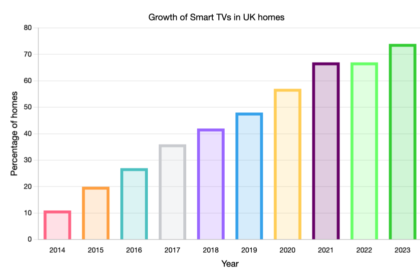

Figure 1: Bar chart showing growth of smart TVs in UK homes (Statista, 2023) Created with Line Graph Maker (Line Graph Maker, 2023)

Some of these devices still support OTA broadcasts, however devices like the Amazon Fire TV stick and Googles Chromecast, are purely internet based; However they do offer a *'guide/epg'* section with Amazon having a development guide (Amazon, 2021) on how to integrate with it. Director general of the BBC, Tim Davie, in 2022 stated:

> *'The vision is simple: from today we are going to move decisively to a digital-first BBC'* (Davie, 2022)

This statement highlights the goal to put more organisational focus on these new forms of media and internet enabled devices.

This report will discuss an upgrade carried out to the BBCs *'off-product'* schedules system, responsible for delivering up to date schedules to partners such as Freeview, Amazon and more. First I will give some background on the project, where I will discuss topics including storage Solutions and how they can work in parallel/multi-threaded systems, and strategies to protect live code systems in a CI/CD environment. I will also give some background on the starting architecture of the system and how the changes align with the BBCs and teams OKRs (Sparks, 2024).

Following that, I will discuss the work that was done. This will be broken down into 5 sections that align with our teams ways of working flow.

1. Requirements and epic creation
2. Investigation and Spike
3. Slicing and task/ticket creation
4. Development of software
5. Releasing of software

I will then talk about the outputs of the project. Theses will include burn-up charts for the projects, dashboards created, documentation of the final architecture and a description of the final product.

Finally I will discuss potential improvements for future iterations. This will range between small code changes to a complete re-architecture of the system.

3 Background

In this section I will discuss the background work and research done for this project. I will start by disusing my teams place in the organisation and our OKRs, explaining how this project helps us hit these objectives. I will then outline the current architecture and the initial design for the project. Finally I will discuss some areas of interest around project, these include cloud computing, database parallelisation strategies and CI/CD challenges.

3.1 Organisation and Team

The BBC is broken into multiple layers with different responsibilities and goals. I am in a team called *SpaceChimps* which is partner of the partnerships group, which itself is in the product group.

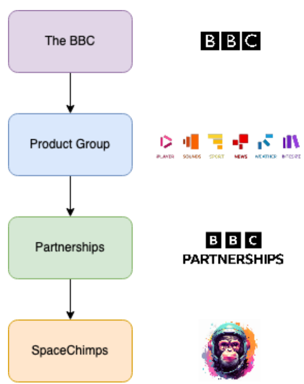

Figure 2: Image showing SpaceChimps place in the BBC (Bowker, 2023).

Our main aim as a team is to provide data to partners that they can use on their devices to promote BBC content. The project described in this report does just that by providing schedules for live content to partners. This aim fits directly into Partnerships objectives:

- It helps drive growth as we are able to get content out to more people on more devices, increasing exposure to the BBC.
- It helps us improve our partner experience by working with them on integrating the data into their feeds.
- This project reduces the total time processing data, which therefore reduces our costs and makes us more sustainable.

All objectives can can be seen in **Appendix A** (BBC Partnerships, 2023).

As well as schedules we also provide a *'catalogue'* of episodes, series and brands that are currently available on iPlayer.

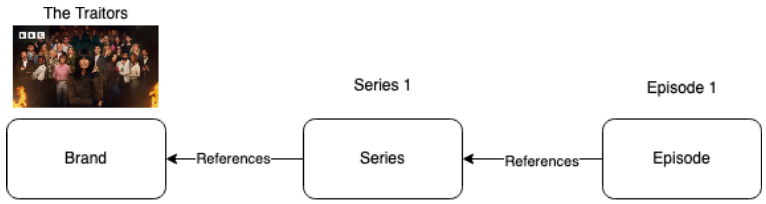

Figure 3: Image showing catalogue hierarchy and how they reference each other.

We provide both a 0 day and 8 day catalogue, these containing programme data that are available now or up to 8 days in the future. We also have an *'unfiltered'* catalogue that is not available to partners which contains all programme data with no availability limits. This unfiltered catalogue is what is used by the schedule pipeline to get it's data about episode/series/brands within the schedule.

3.2 Original Architecture

The original solution was composed of AWS services that created a pipe and filter architecture which transformed inputs into multiple outputs (Somerville, 2016, pp.182-183) that can be used by partners. The figure below shows this architecture.

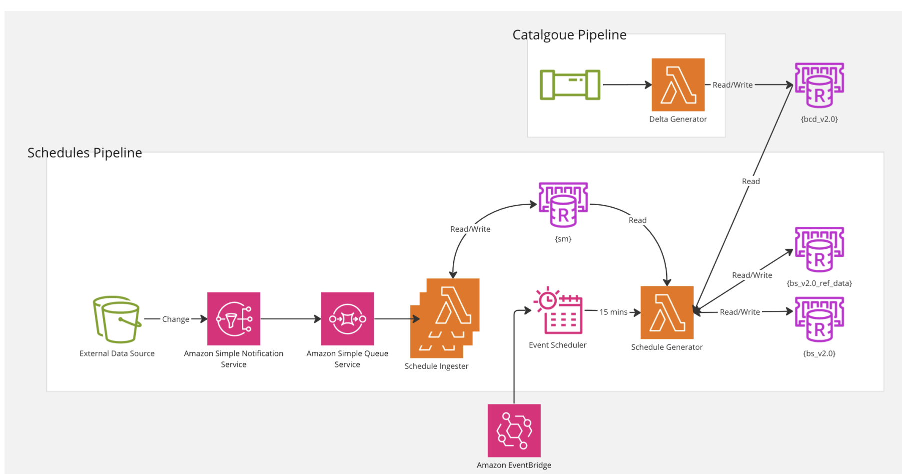

Figure 4: Image showing the initial architecture.

A pipe and filter architecture is achieved with the combination of AWS lambda (Amazon Web Services, 2024a) with updates being triggered by a publish/subscribe model (Somerville, 2021, pp.179) achieved through AWS' Simple Notification Service (SNS) (Amazon Web Services, 2024b) and Simple Queue Service (SQS) (Amazon Web Services, 2024c) with the former publishing and the latter subscribing. The lambda only runs when a message is published to the SQS, this triggers the lambda and the pipeline begins precessing the new message.

The data is updated by an external system in AWS S3 (Amazon Web Services, 2024d), this publishes a message that the data has changed, our first lambda (the ingester) receives this message and processes/stores it into a *'common'* model that's used internally only. Our internal model is then up to date, however our partner facing model, created by the schedule generator, is not. In the original system this lambda was not driven by events, but instead ran every 15 minutes and processed all the schedules in one go, whether they had updated or not.

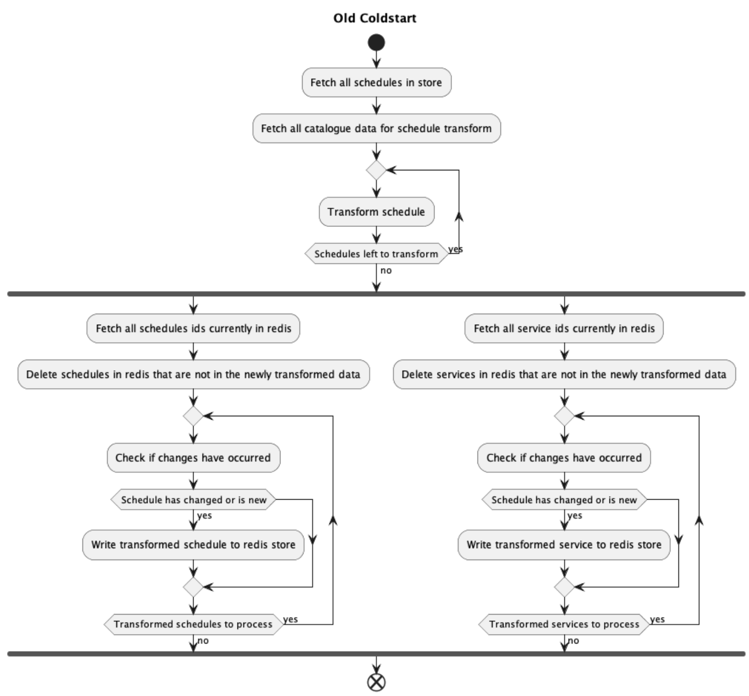

Figure 5: Activity diagram showing schedule generators logic.

This could leave partners with data up to 15 minutes out of date, dependant on when they retrieve the data, resulting in end users being shown incorrect schedules.

3.2.1 Initial design solution

The schedules pipeline also requires data from our catalogue pipeline, for titling, descriptions, viewer discretion warnings and subtitles. For this reason it needs to be alerted when catalogue data changes as well as when schedules change.

An initial design had already been complete before the work started by another member of the team.

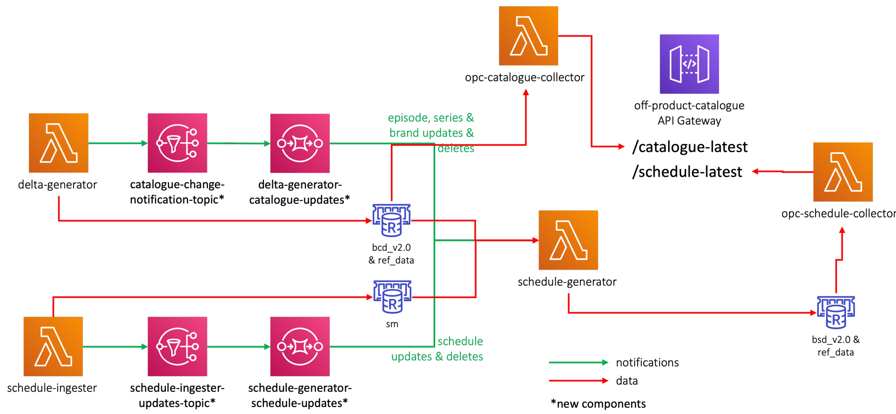

Figure 6: Image showing the initial updated schedules design (Lloyd, 2023).

The above diagram has been edited down to include only work related to the project, a full diagram including notifications being directly sent to partners can be found in **Appendix B**. This diagram also includes our endpoints and collectors that partners use to retrieve the data. This proposed system uses the pub/sub model described earlier and will subscribe to both catalogue and schedule events.

Alongside the architectural design an algorithm was proposed for these events.

- **For schedule updates** - We want to update the partner facing model of the schedule linked in the notification. This schedule has a list of broadcasts that map to episodes in the catalogue. A list should be maintained within each episode to create a link between both, allowing episode updates to trigger updates to schedules that reference them. All catalogue data referenced in a schedule should be copied over to the schedules keyspace, leaving the catalogue keyspace and items untouched.

- **For schedule deletes** - Remove schedule from store, and remove schedule reference from list contained within each episode that schedules references in its list of broadcasts.

- **For catalogue/programme updates:**

  For episodes - Update episode in store, update each schedule that is linked in episodes broadcast list.

  For series/brands - Get all episodes linked to either the series or brand, update each schedule that is linked in each episodes broadcast list.

Flow diagrams can be found in **Appendix C** (Lloyd, 2023).

4 Research

I will now explore research done relating to the project specified. This will outline some design decisions that were made and link to other sections of the report where they may be explored more. This section will explore parallelism and thread safety focusing on data stores as well as CI/CD, it's pros and cons when deploying to live systems and how we as a team worked around it.

4.1 Storage Solutions and Parallelism

Currently the schedule pipeline consists of two components, the ingester, followed by the schedule generator. The first part of this pipeline is parallelised, multiple lambdas can be ran at the same time to insert data into the redis. This is a harder task for the schedule generator to do, as it needs update a list of linked schedules to an episode in the redis store. This data can be edited through multiple streams, both the schedule catalogue pipelines, meaning the array could easily become incorrect/polluted. Sharing memory in a threaded/parallelised system is a well known challenge and you need to know when it's safe to update/edit this memories value (https://homes.cs.washington.edu/ djg/teachingMaterials/spac/sophomoricParallelismAnd
This is also known as being thread safe which can be described as *'different threads can access the same resources without exposing erroneous behavior or producing unpredictable results'* (Ugarte, 2024).

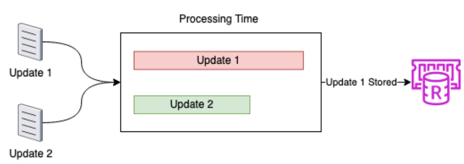

Figure 7: Diagram showing how a broadcast list of an episode can become incorrect/polluted.

4.1.1 Redis and Elasticache

We use amazons Elasticache (Amazon Web Services, 2024f) for our redis (REmote DIctionary Server) solution as it stores data in memory, which makes it really quick to retrieve stored data (IBM, 2024). This is vital for us as we store large documents that need to be retrieved and sent to partners on an API request. Redis is single-threaded but supports concurrency, *'when at least two threads are making progress'* (Oracle Corporation, 2010) which is not the same as parallelism, *'when at least two threads are executing simultaneously'* (Oracle Corporation, 2010).

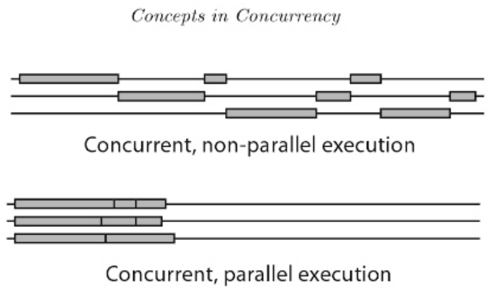

Figure 8: Difference between concurrency and parallelism.
https://books.google.ie/books/about/Introduction_to_Concurrency_in_Programmi.html?id=J5-ckoCgc3IC&redir_esc=y

This concurrency allows redis to support multiple requests at once, but it cannot do multiple operations at once; although more recent versions are allowing some safely threaded operations such as deleting records (Redis Ltd, 2024a). It also supports batch uploading and blocking commands, however these also don't help keep the data thread-safe.

- **Pipelining** - Pipelining sends a block of commands at once, however does not guarantee that commands sent are done in sequence (Redis Ltd, 2024b).

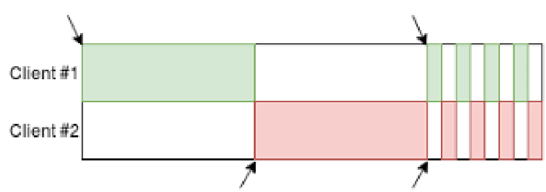

Figure 9: How redis pipelines don't guarantee sequential execution (Eyng, 2019).

- **Transactions** - Transactions are very similar to pipelines, however guarantee that the transactions commands are not interrupted by another clients requests and are therefore executed in sequence (Redis Ltd, 2024c).
- **Blocking Actions** - Blocking actions stop the current client from executing commands until the blocking action is complete. However other clients can still send requests to the server whilst this client is blocked (Redis Ltd, 2024d).

The options above still allow the previous race condition to occur, as instances of the schedule generator may vary in processing time and therefore the time of writing to the redis cannot be guaranteed to be in order of the events.

4.1.2 How thread-safety can be achieved

When researching and spiking (Visual Paradigm, 2024) the project, other technologies were found that could help offer thread safety whilst parallelising the schedule generator. These were types of store/database locking mechanisms.

- **Pessimistic Locking -** This method *'assumes that access to shared memory will be contended'* (Weston, 2011) and acquires a lock on the data to be edited. Any other client/connection attempting to edit this data must wait until this lock is released to update the data (Thornton, 2001). This can lead to issues such as deadlock which is when two clients are both awaiting on another clients lock to be released. This can end up with both clients being stuck in a endless cycle of waiting for each other (Thornton, 2001; Apache Software Foundation, 2013).

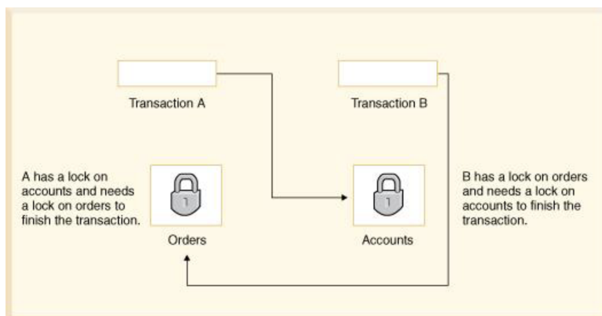

Figure 10: Example of a deadlock scenario (Apache Software Foundation, 2013).

  There is also a situation where a client locks a piece of data and for one reason or another halts, this would cause needless delay, when a timeout is specified, if a timeout is not specified then this object would be locked from changes indefinitely (Thornton, 2001).

- **Optimistic Locking -** This method *'relies on end-of-transaction validation'* (Graefe, 2016). Unlike its counterpart (pessimistic locking) it does not lock the record that is being updated. Instead before the new data is written, the original data is checked against the current data stored (Thornton, 2001). If this data doesn't match then a change has occurred during processing and the new data to be written must be re-calculated with the new changes.

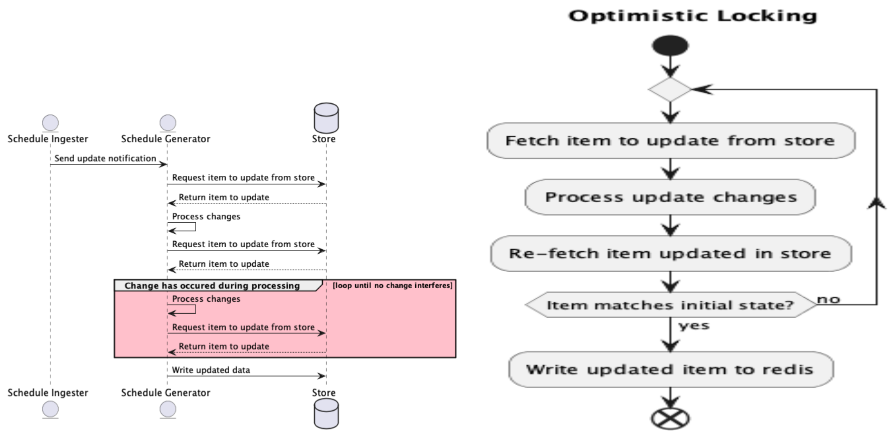

Figure 11: Sequence and activity diagrams outlining logic of optimistic locking.

> This above logic could be implemented into our redis solution. It would require either a total comparison of the object or a simple *version* field that specifies when the object has changed.

During this investigation DynamoDB (Amazon Web Services, 2024e) was highlighted as a potential option due to it supporting optimistic locking. The final decision was to not use it however and stick with the current solution as there was a want to get the project started to keep up with our roadmap and not further investigate this option. There was also more unknowns with this technology as we had never used it before. I will discuss a solution using DynamoDB in the **Future Work** section of this report.

4.2 Agile and CI/CD

As a team we follow an agile approach to software development, more accurately we follow a kanban approach. The kanban methodology *'focuses more on monitoring and improving workflows'* (Heil, C, 2022) and doesn't use concepts such as sprints from scrum (Rehkopf, 2024). Instead software created using the kanban approach is deployed/released when it's done (Rehkopf, 2024) and uses a kanban board where tickets move across columns depicting where they are currently at in development cycle (Mauvius Group Inc, 2021).

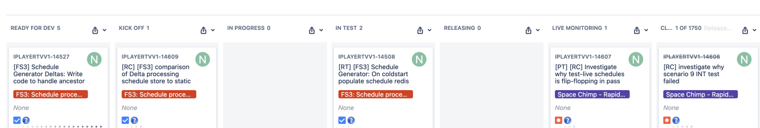

Figure 12: SpaceChimps kanban board.

This workflow fit's in well with Continuous Integrations and Continuous Deployment (CI/CD). As a team we release/build all our own code to multiple environments using Jenkins declarative pipelines (Jenkins, 2024).

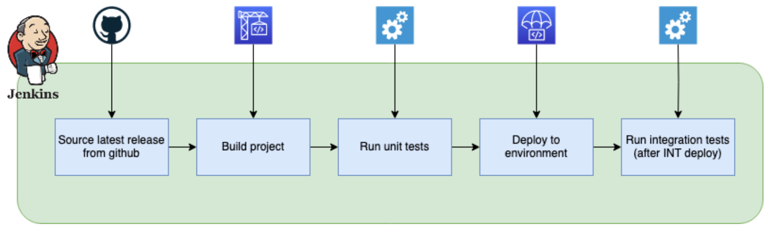

Figure 13: General pipeline flow.

Having these well defined deployment pipelines makes deploying new code easy and helps manage deployments across multiple environments as configuration can be built in to the pipeline (Rodriguez, et al, 2016). This approach allows bugs to be caught faster than when using a *'big-bang'* approach, as the bug is likely to be with a single new change (Department of Defence, 2021). The additional testing also gives more confidence in the code deployed and rollbacks are made extremely simple due to version control management.

There are of course some downsides to CI/CD, whenever an external platform, Jenkins in our case, is being used this opens up a new attack vector (NSA, 2023). OWASP keeps a list of some of these threats (OWASP Foundation, 2023), however they mostly comprise of poor authentication to CI/CD systems and attacks by people who already have access to source code and the CI/CD platform itself (employees).

Other issues include maintenance of the pipelines code/infrastructure and potential complexity (Wikström, 2019), this can be especially true when pipelines call other processes.

Code quality can become an issue, with technical debt increasing due to the encouraged continuous deployment of new software (Rodriguez, et al, 2016) and more bugs also appear to occur. One study showed that the number of bugs actually increased when using CI/CD (Fairbanks, Tharigonda, Eisty, 2023).

	CI/CD	No CI/CD
Github average issues	135.38	57.33
Gitlab average issues	52.04	17.68

Table 1: Table from study showing difference in issues found between approaches (Fairbanks, Tharigonda, Eisty, 2023).

However this same study also showed a commit velocity increase of 141%, so it could be argued that these bugs are quickly remedied due to this quicker release time. In addition to this as a team we do pull requests, another developer checks new code before release, which should also mitigate some of the outlined issues above.

4.2.1 Software Changeover Strategies

The system that is being upgraded is in LIVE use by partners, but the work itself will take time to implement. We don't want to make changes to how the LIVE system works for partners but we do want to test that the new software works on the LIVE environment without a *big-bang* deployment.

There are three main types of changeover strategy, direct (big-bang), parallel running or a phased strategy and it's multiple variants (Banerjee, 2017).

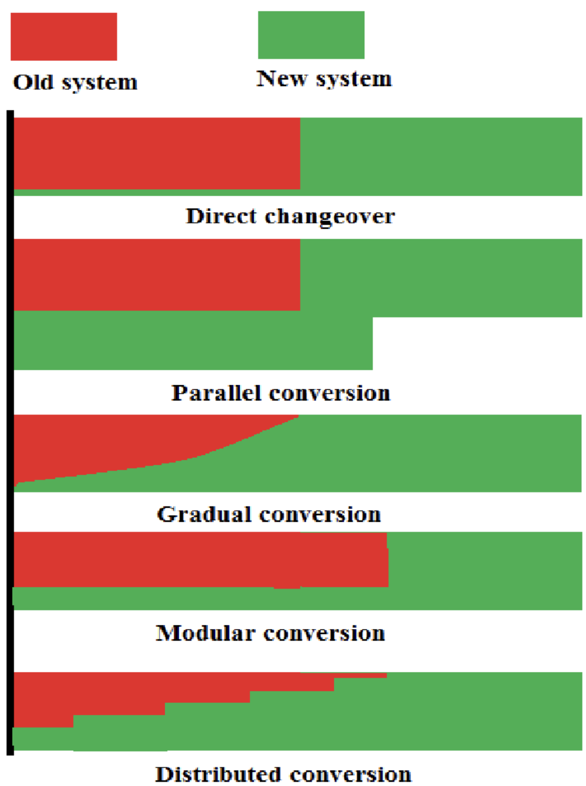

Figure 14: Timeline of different changeover strategies (Banerjee, 2017).

As discussed a direct approach is risky and could easily run into errors, a phased approach in this situation is also not valuable as the we want to remain consistent in the data we provide to all our partners, so a parallel system is our best option. Temporarily the old and new system will run side-by-side, this will allow us to write comparison tests between the old output and the new (Smyth, 2020).

In addition to having both systems the data on redis also needs to be separate as they may differ during development. This will be achieved by using different redis keyspace prefixes (IoRedis, 2024) to keep the data separate and allow a test to check both for differences (Rustagi, 2023). The partners would continue to get data from the old keyspace, when we are happy with our comparison tests the partners can be swapped over to the new keyspace and the old data can be deleted.

5 Work Done

In this next section I will discuss the work that was carried on the project. In our team we have *'ways of working'* flowchart that helps guides us in how the project should be done throughout the team (GoRetro, 2023). Our ways of working is broken down into 5 sections, all of which will be discussed. For a full diagram of our workflow see **Appendix D**.

5.1 Requirements and epic creation

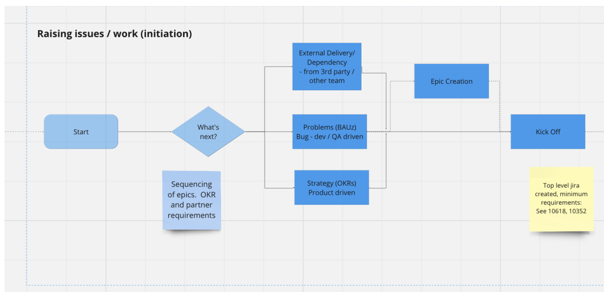

Figure 15: Initiation stage of for ways of working.

5.2 Investigation and spike

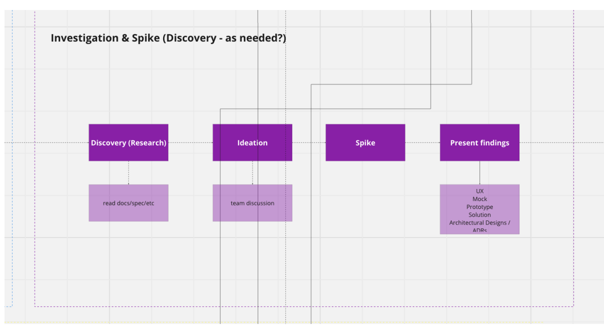

Figure 16: Investigation stage of for ways of working.

5.3 Slicing and kick-off

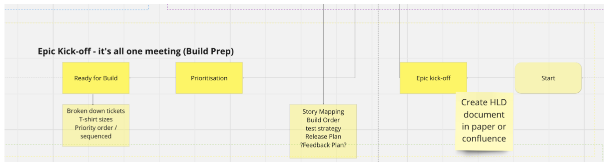

Figure 17: Kick-off stage of for ways of working.

5.4 Build software

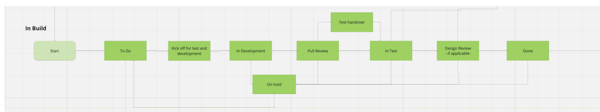

Figure 18: Build stage of for ways of working.

5.5 Release

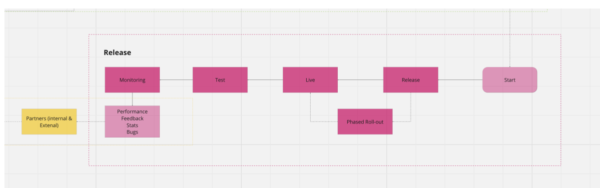

Figure 19: Release stage of for ways of working.

6 Outputs

7 Future Work

Extensible record (Dynamo DB) - https://courses.cs.duke.edu/fall16/compsci516/Lectures/Lecture-18-NoSQL-ColStore.pdf

8 Conclusion

9 References

BBC. (2022) *Timeline - History of the BBC* [online]. Available at https://www.bbc.co.uk/historyofthebbc/bbc-100/timeline/ (accessed on 18th January 2024).

Pilnick, C. Baer, S, W. (1973) *Cable television: A guide to the technology* [online]. Available at https://www.rand.org/content/dam/rand/pubs/reports/2006/R1141.pdf (accessed on 18th January 2024).

Ofcom. (2023) *Media Nations report 2023* [online]. Available at https://www.ofcom.org.uk/__data/assets/pdf_file/0029/265376/media-nations-report-2023.pdf (accessed on 18th January 2024).

Ampere Analysis. (2023) *Almost half of Internet users say they have switched off broadcast TV* [online]. Available at https://www.ampereanalysis.com/press/release/dl/almost-half-of-internet-users-say-they-have-switched-off-broadcast-tv (accessed on 18th January 2024).

Statista. (2023) *Household penetration of smart TV sets in the United Kingdom (UK) from 2014 to 2023* [online]. Available at https://www.statista.com/statistics/654074/smart-tv-penetration-in-uk-households/ (accessed on 18th January 2024).

Line Graph Maker. (2023) *Bar Graph Maker* [online]. Available at https://linegraphmaker.co/bar-graph (accessed on 19th January 2024).

Amazon. (2021) *Linear TV Integration Guide Overview* [online]. Available at https://developer.amazon.com/docs/fire-tv/linear-tv-integration-guide-overview.html (accessed on 18th January 2024).

Davie, T. (2022) *A digital-first BBC* [online]. Available at https://www.bbc.co.uk/mediacentre/speeches/2022/digital-first-bbc-director-general-tim-davie (accessed on 18th January 2024).

Sparks, R. (2024) *OKRs: the ultimate guide to objectives and key results* [online]. Available at https://www.atlassian.com/agile/agile-at-scale/okr (accessed on 18th January 2024).

Bowker O. (2023) *Analysis of BBC process and proposal for a new API key management system.* Assignment for DC4000, MSc Digital and Technology Solutions Specialist, Aston University, Unpublished.

BBC Partnerships. (2023) *Partnerships - one year on.* BBC. Unpublished.

Somerville, I. (2016) *Software engineering.* Tenth edition, Global edition. London: Pearson Education.

Somerville, I. (2021) *Engineering Software Products: An Introduction to Modern Software Engineering.* Global edition. London: Pearson Education.

Amazon Web Services. (2024a) *AWS Lambda* [online]. Available at https://aws.amazon.com/lambda/ (accessed on 2nd February 2024).

Amazon Web Services. (2024b) *Amazon Simple Notification Service* [online]. Available at https://aws.amazon.com/sns/ (accessed on 2nd February 2024).

Amazon Web Services. (2024c) *Amazon Simple Queue Service* [online]. Available at https://aws.amazon.com/sqs/ (accessed on 2nd February 2024).

Amazon Web Services. (2024d) *Amazon S3* [online]. Available at https://aws.amazon.com/s3/ (accessed on 2nd February 2024).

Amazon Web Services. (2024e) *Amazon DynamoDB* [online]. Available at https://aws.amazon.com/dynamodb/ (accessed on 2nd February 2024).

Amazon Web Services. (2024f) *Amazon ElastiCache* [online]. Available at https://aws.amazon.com/elasticache/ (accessed on 9th February 2024).

IBM. (2024) *What is Redis?* [online]. Available at https://www.ibm.com/topics/redis (accessed on 9th February 2024).

Redis Ltd. (2024a) *Redis FAQ* [online]. Available at https://redis.io/docs/get-started/faq/ (accessed on 9th February 2024).

Redis Ltd. (2024b) *Redis pipelining* [online]. Available at https://redis.io/docs/manual/pipelining/ (accessed on 9th February 2024).

Redis Ltd. (2024c) *Transactions* [online]. Available at https://redis.io/docs/interact/transactions/ (accessed on 9th February 2024).

Redis Ltd. (2024d) *Redis modules and blocking commands* [online]. Available at https://redis.io/docs/reference/modules/modules-blocking-ops/ (accessed on 9th February 2024).

Oracle Corporation. (2010) *Defining Multithreading Terms* [online]. Available at https://docs.oracle.com/cd/E19455-01/806-5257/6je9h032b/index.html (accessed on 9th February 2024).

Eyng, R. (2019) *Redis: Pipelining, Transactions and Lua Scripts* [online]. Available at https://rafaeleyng.github.io/redis-pipelining-transactions-and-lua-scripts (accessed on 9th February 2024).

Visual Paradigm. (2024) *What is Spike in Scrum?* [online]. Available at https://www.visual-paradigm.com/scrum/what-is-scrum-spike/ (accessed on 9th February 2024).

Ugarte, A. (2024) *What Is Thread-Safety and How to Achieve It?* [online]. Available at https://www.baeldung.com/java-thread-safety (accessed on 9th February 2024).

Graefe, G. (2016) *Revisiting optimistic and pessimistic concurrency control* [online]. Available at https://www.labs.hpe.com/techreports/2016/HPE-2016-47.pdf (accessed on 9th February 2024).

Thornton, G. (2001) *Optimistic Locking with Concurrency in Oracle* [online].

Available at https://www.orafaq.com/papers/locking.pdf (accessed on 9th February 2024).

Weston, T. (2011) *Optimistic and Pessimistic Concurrency Control with Shared Memory Models* [online]. Available at https://baddotrobot.com/resources/concurrency-control-1.0-draft.pdf (accessed on 9th February 2024).

Apache Software Corporation. (2013) *Deadlocks* [online]. Available at https://db.apache.org/derby/docs/10.0/manuals/develop/develop75.html (accessed on 9th February 2024).

Heil, C. (2022) *Is Kanban the better Agile Approach in a Highly-Regulated Environment?* [online]. Available at https://www2.deloitte.com/content/dam/Deloitte/de/Documents/risk/Deloitte_Agile_Approach_in_Highly_Regulated_Environments.pdf (accessed on 12th February 2024).

Rehkopf, M. (2024) *Kanban vs. scrum: which agile are you?* [online]. Available at https://www.atlassian.com/agile/kanban/kanban-vs-scrum (accessed on 12th February 2024).

Mauvius Group Inc. (2021) *The official guide to the kanban method* [online]. Available at https://resources.kanban.university/wp-content/uploads/2021/06/The-Official-Kanban-Guide_A4.pdf (accessed on 12th February 2024).

Jenkins. (2024) *Pipeline Syntax* [online]. Available at https://www.jenkins.io/doc/book/pipeline/syntax/ (accessed on 12th February 2024).

Rodriguez, P. Haghighatkhah, A. Lwakatare, L, E. Teppola, S. Suomalainen, T. Eskeli, J. Karvonen, T. Kuvaja, P. Verner, J, M. Oivo, M. (2016) *Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study* [online]. Available at http://jultika.oulu.fi/files/nbnfi-fe201902185288.pdf (accessed on 14th February 2024).

Department of Defence. (2021) *DoD Enterprise DevSecOps Fundamentals* [online]. Available at https://dodcio.defense.gov/Portals/0/Documents/Library/DoDEnterpriseDevSecOpsFundamentals.pdf (accessed on 14th February 2024).

NSA. (2023) *Defending Continuous Integration/Continuous Delivery (CI/CD) Environments* [online]. Available at https://media.defense.gov/2023/Jun/28/2003249466/-1/-1/0/CSI_DEFENDING_CI_CD_ENVIRONMENTS.PDF (accessed on 14th February 2024).

OWASP Foundation. (2023) *OWASP Top 10 CI/CD Security Risks* [online]. Available at https://owasp.org/www-project-top-10-ci-cd-security-risks/ (accessed on 14th February 2024).

Wikström, A. (2019) *Benefits and challenges of Continuous Integration and Delivery - A Case Study* [online]. Available at https://core.ac.uk/download/pdf/226768285.pdf (accessed on 14th February 2024).

Fairbanks, J. Tharigonda, A. Eisty, N, U. (2023) *Analyzing the Effects of CI/CD on Open Source Repositories in GitHub and GitLab* [online]. Available at https://arxiv.org/pdf/2303.16393.pdf (accessed on 14th February 2024).

GoRetro. (2023) *Ways of Working* [online]. Available at https://www.goretro.ai/glossary/ways-of-working (accessed on 14th February 2024).

Banerjee, A. (2017) *System Changeover Policy and Precautions* [online]. Available at https://ijoaemorg.files.wordpress.com/2017/11/02090048_r.pdf (accessed on 15th February 2024).

Smyth, D. (2020) *Changeover Techniques* [online]. Available at https://smallbusiness.chron.com/changeover-techniques-34890.html (accessed on 15th February 2024).

IoRedis. (2024) *README - ioredis* [online]. Available at https://ioredis.readthedocs.io/en/stable/README/#transparent-key-prefixing (accessed on 15th February 2024).

Rustagi, V. (2023) *Strategies To Run Old & new Systems Simultaneously Using The Same Database* [online]. Available at https://metaorangedigital.com/blog/strategies-to-run-old-new-systems-simultaneously-using-the-same-database/ (accessed on 15th February 2024).

Lloyd, K. (2023) *iPlayer Schedules - BBC Partner Schedule Generator - Delta Update Technical Design*. BBC. Unpublished.

10 Appendix

10.1 Appendix A

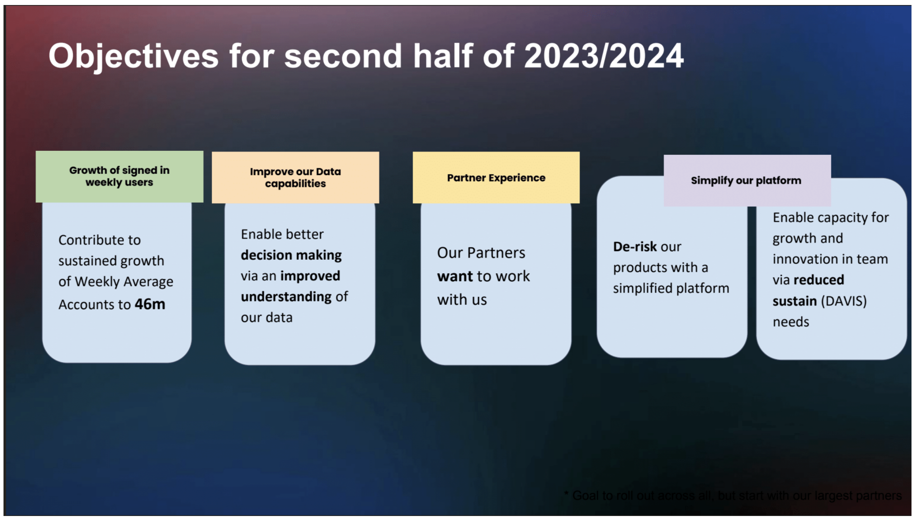

Figure 20: Image taken from a presentation given at a partnerships context setting event (BBC Partnerships, 2023).

10.2 Appendix B

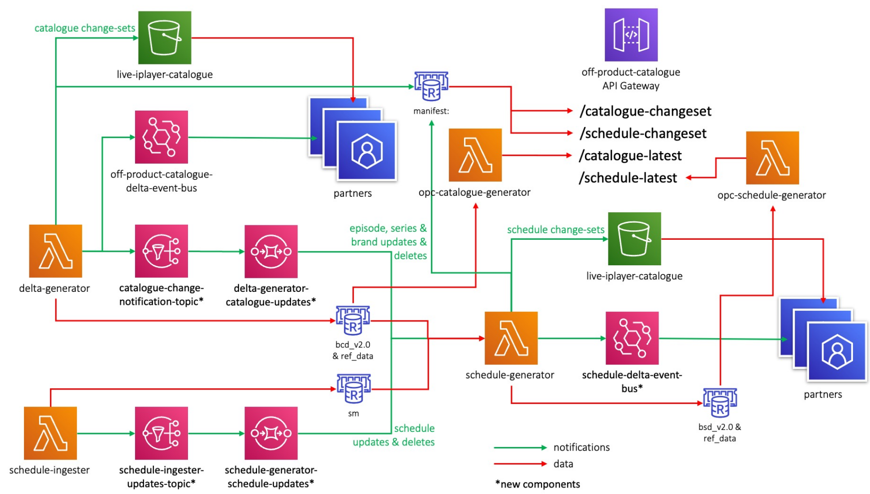

Figure 21: Full diagram of design for schedules pipeline, including future notifications to partners work (Lloyd, 2023).

10.3 Appendix C

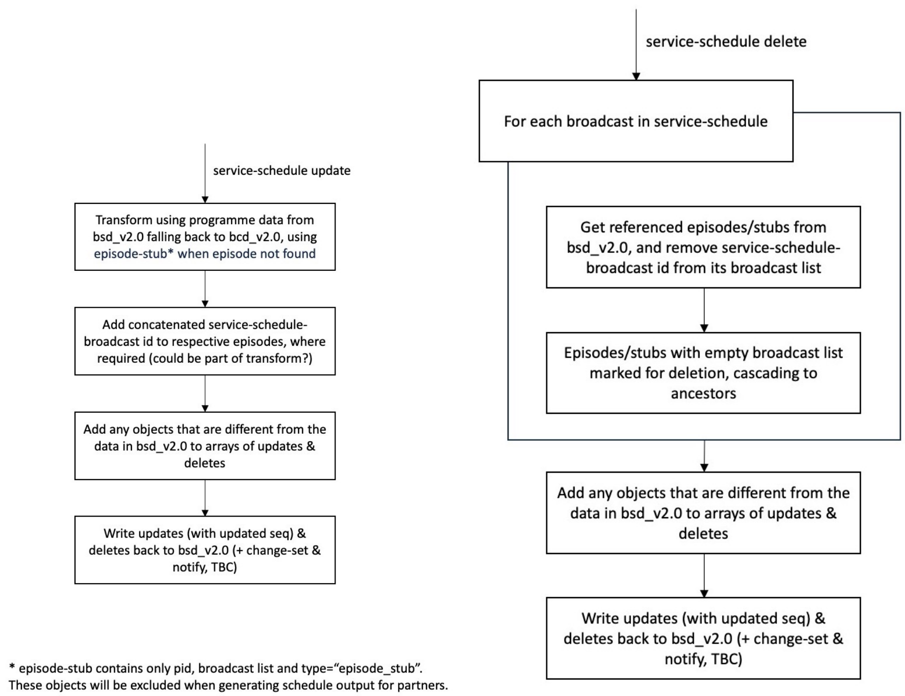

Figure 22: Flow diagrams for schedule events (Lloyd, 2023).

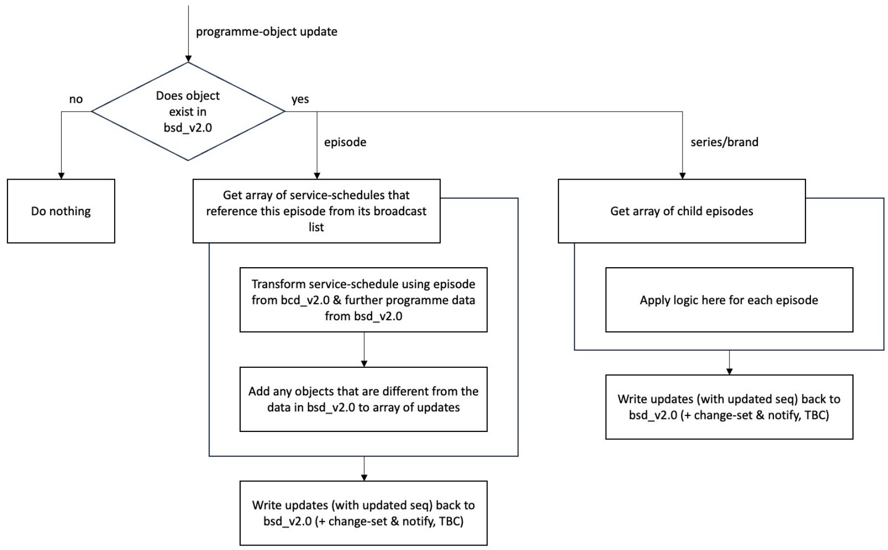

Figure 23: Flow diagram for catalogue/programme events (Lloyd, 2023).

10.4 Appendix D

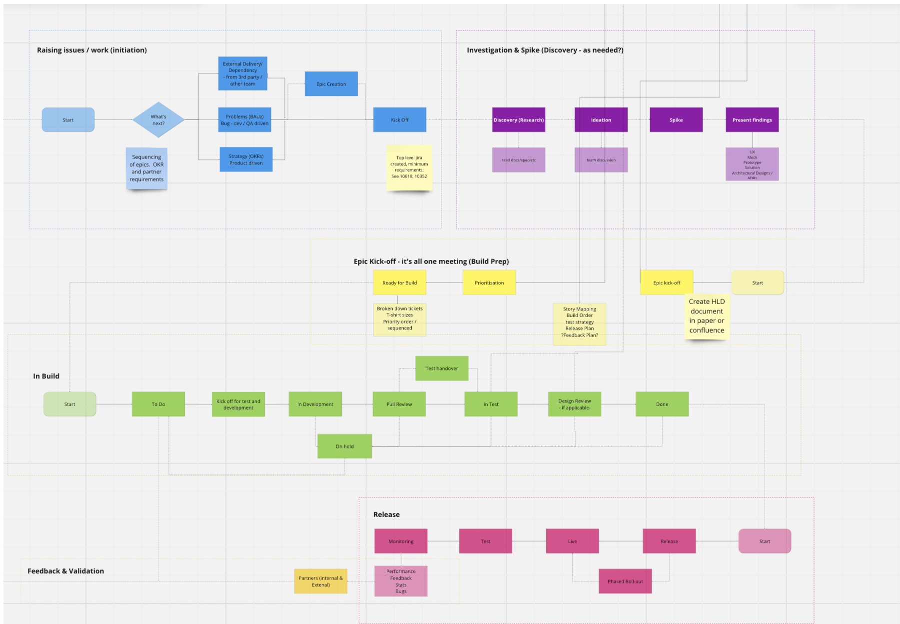

Figure 24: Full ways of working flow diagram used by SpaceChimp.