Building an integrated desktop application ecosystem for Finance

A guide to the tools, products and migration challenges

This paper identifies the core building blocks that are required to build an integrated desktop application ecosystem using a combination of web and legacy technologies – an ecosystem where it is easier to deploy and integrate both your own applications and those from third-party vendors. We look at the challenges involved in moving from legacy desktop technologies to the fast-moving world of web technologies. And finally, we review the various open source and commercial products that form the building blocks of this ecosystem of the future.

A white paper by Colin Eberhardt

SCOTT LOGIC

ALTOGETHER SMARTER

Contents

Introduction

Financial services professionals interact with a wide range of disparate systems throughout the course of their working day. They rely on these to provide access to data, interact with others in the business and perform a multitude of actions across trading, risk, sales and beyond. However, working across this heterogeneous application estate can be a challenge, with inconsistencies in user experience and interface quality, duplicate functionality, and manual intervention required in order to move data between systems – all resulting in workflows that are fractured and extremely inefficient.

While there are a number of reasons why this burgeoning application estate is so hard to tame, technology is a major factor. The tendency is for many disparate internal systems to be implemented to provide tactical solutions, but these often suffer from a lack of investment over many years and rely on legacy development frameworks.

Recently, we've seen an emerging vision for an integrated desktop application ecosystem, with various vendor products providing integration and interoperability. There are also open standards such as FDC3 which provide an agreed framework for communicating data and actions between applications. This vision outlines a better future, with a connected ecosystem of applications that deliver processes efficiently in support of user-centred workflows. These loosely coupled applications all live within a framework that is easy to deploy, update and maintain.

There is broad agreement across the industry that a key component of this vision is a move away from legacy desktop frameworks to web technologies – a move that we are seeing in the wider technology community. The vision is certainly appealing; many business-critical applications, especially those which are not customer-facing, are built with legacy technologies and have meagre development budgets. However, a wholesale migration is not feasible. A more pragmatic approach is to migrate some apps, while retaining and integrating legacy applications into a hybrid desktop ecosystem.

The benefits of an incremental migration towards a hybrid ecosystem are manifold. This approach provides the opportunity to 'break up' monolithic applications and create a new enhanced desktop experience, whilst also leveraging previous investments by making selective decisions around what is (and what is not) migrated, and when.

This paper identifies the core building blocks that are required to build an integrated desktop application ecosystem using a combination of web and legacy technologies – an ecosystem where it is easier to deploy and integrate both your own applications and those from third-party vendors. We look at the challenges involved in moving from legacy desktop technologies to the fast-moving world of web technologies. And finally, we review the various open source and commercial products that form the building blocks of this ecosystem of the future.

The desktop ecosystem of the future

You only have to walk the floor of any financial institution to understand the day-to-day complexity experienced by those who work there. Whether it is risk, trading or operations, you'll find people sat behind walls of monitors, displaying a vast quantity of real-time data. Even the telephones are complex and intimidating to the uninitiated!

There is no doubt that these professionals require immediate access to data, productivity tools and the ability to transact based on a diverse range of data sources. However, there still exist inefficiencies due to a lack of consistency and interoperability between the applications that occupy their desktops. Users are faced with a heterogeneous mix of third-party systems and bespoke applications, which they have to mentally 'map' between.

So what does a better future look like?

For the user, it is one where the applications on their desktop are task-oriented and efficient, where interfaces surface the data a user needs and allow them to respond rapidly; where applications speak the same language, allowing them to seamlessly integrate in support of bespoke user-oriented workflows that remove repetition and re-keying of information.

For the business, a better future is one supported by a framework that allows easy deployment of tools for both internal users and external clients; a framework that provides the ability to integrate third-party products into their internal workflows, removing the need to build their own integrations. It is a future that is ultimately free from constraints of legacy, where the old and new live side by side.

This is a vision that isn't far off, and is one shared by a number of product vendors and the open source community. However, there are a number of obstacles in the way, the most notable being the legacy technologies that occupy the desktop of almost every financial institution.

The desktop legacy challenge

Since the dawn of PCs, productivity and line-of-business applications have been installed directly to the desktop operating system either by the user, or rolled-out by a central IT team. However, as web browser capabilities have grown, these applications have been replaced with HTML5/Single Page Apps (SPA) counterparts. For consumer-facing applications this is the primary delivery mechanism, with users enjoying immediate access (without installation) to their business-critical applications from any device, anywhere in the world.

However, for most businesses the internal-facing applications – which facilitate sales, customer relations, operations (and more) – outnumber consumer-facing applications by at least an order of magnitude. Despite the criticality of these applications, they are typically developed using traditional (and now legacy) desktop technologies. So why are so many internal tools being left behind?

In some cases, these tools genuinely do need a level of desktop integration that is not possible through the browser (although as we will see later in this paper, the capability gap is now quite small). In this case, it is still possible to leverage web technologies via the use of a desktop container. However, more often than not, the reason they are left behind is simply the cost of migration. For complex applications, a screen-by-screen migration is a significant undertaking, which is further exacerbated by architectural changes that are needed to, for example, remove direct database connections.

From a wider industry perspective, the use of web technologies[1] for building desktop applications is quite commonplace. The technology ecosystem provides multi-threading, GPU-acceleration and native execution speed – there is very little you can't do with web tech! Popular and widely used productivity apps, such as Visual Studio Code, Slack, Microsoft Teams and Spotify are all built using web technologies. For greenfield desktop application development, it has become the natural choice.

When using web technologies to build desktop applications, you need to replace the browser with some other runtime – and there are numerous options available to you, both commercial and open source. They also offer a range of features that go far beyond a simple browser-replacement, easing the process of migrating legacy applications, by providing adapters/connectors, and delivering additional features and value.

Consumer apps

- Relatively small number
- Web (and some social)
- Easy to distribute
- Available worldwide
- Multiplatform
- High budget

Internal apps

- Numerous
- Diverse tech stack
- Often old/legacy tech (WPF, Java)
- Desktop-only
- Hard to distribute/update
- Hard to maintain
- Meagre development budget
- Siloed applications

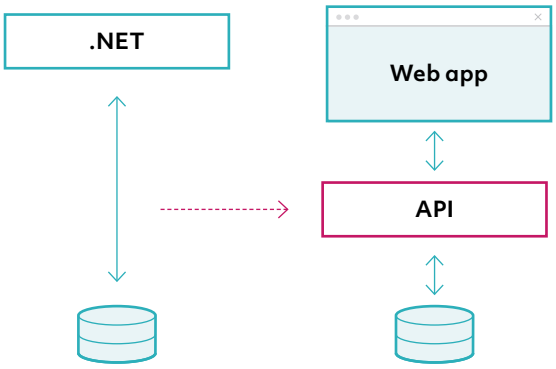
1 Web technologies refers to the four W3C languages (JavaScript, CSS, HTML, WebAssembly) and the extensive array of browser APIs. Some use the term HTML5 to describe these technologies as a collective; however, that term is now quite dated.

The building blocks of a desktop ecosystem

Using web technologies on the desktop isn't an all-or-nothing option. It is possible to mix and match this approach in various configurations in support of a gradual migration. Couple this with the wide range of vendor solutions, each with their own features and differing terminology, and the sheer number of options becomes quite hard to navigate.

Here we'll start by looking at the simplest case, the migration of a single desktop application to web technologies, and use this as a way to introduce the various concepts in a vendor-neutral setting.

A single-application migration



Legacy desktop applications, whether written in Java or C#, using Swing, SWT, Windows Forms or WPF, tend to have similar characteristics. They are often monolithic (or just 'fat'), with features having been added progressively over a number of years, resulting in a bloated and confusing user experience. With consumer applications, analytics can be used to determine which features are most useful, allowing for prioritisation and insight-driven product decisions. However, these metrics are rarely collected for internal tools.

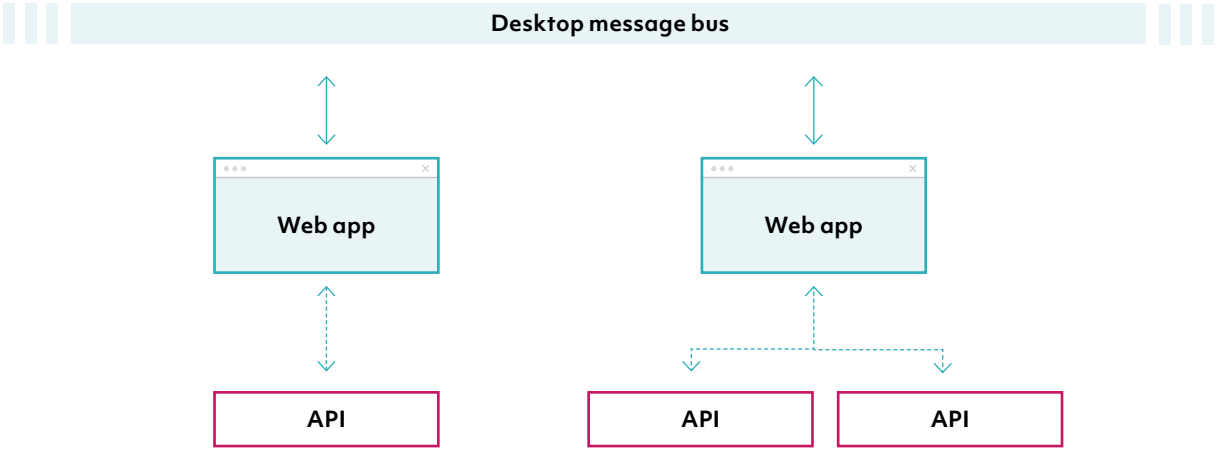
These applications often connect directly to one or more databases, resulting in a tightly coupled architecture where the database schema cannot be evolved. It is these features – a bloated code-base, a poor understanding of usage, and tight coupling – that make these applications so hard to maintain and evolve.

An equivalent application built with web technologies has quite a different architecture. The most obvious changes are in language and framework. Rather than C# or Java, the application is built using JavaScript (or TypeScript), while WPF (or WinForms etc.) is exchanged for React or Angular. For a development team that is not familiar with web development, this technology shift is a significant challenge in itself. Furthermore, web technologies cannot connect directly to databases (or other internal services), and as a result, they integrate with the back-end services via one or more API layers. This approach yields many more benefits beyond simply supporting a web-based front-end; for example, it allows these APIs to become first-class 'services' in their own right.

In order for this equivalent application to be distributed as a desktop application (i.e. installed on a user's machine, launched through an icon and potentially available offline), it needs to be distributed alongside a browser-like environment. This is the **desktop container**, effectively a dedicated browser (with some special extra bells and whistles) for your application.

A like-for-like migration replaces a 'fat' application written in C#/WPF that connects to a database, with an equivalent application written in JavaScript/React, and a suitable API layer. However, in much the same way that a server-side migration from monolith to microservices (or a migration from on-premise to cloud) gives the opportunity to rethink and redesign, so too does the move from legacy desktop technologies to web technologies. It gives the opportunity to split a potentially bloated application into multiple, smaller and more focused applications. Furthermore, there are features of the application that could likely be retired completely[2].

2 Notice that Amazon explicitly details 'Retire' as a cloud migration option among their 6 Rs.



When splitting an application into a cluster of smaller applications, there is a need to support inter-app communication to allow data and events to move seamlessly from one application to another in support of user workflows. A **desktop message bus** provides this connectivity, allowing communication between applications (both bespoke and third party) entirely on the client.

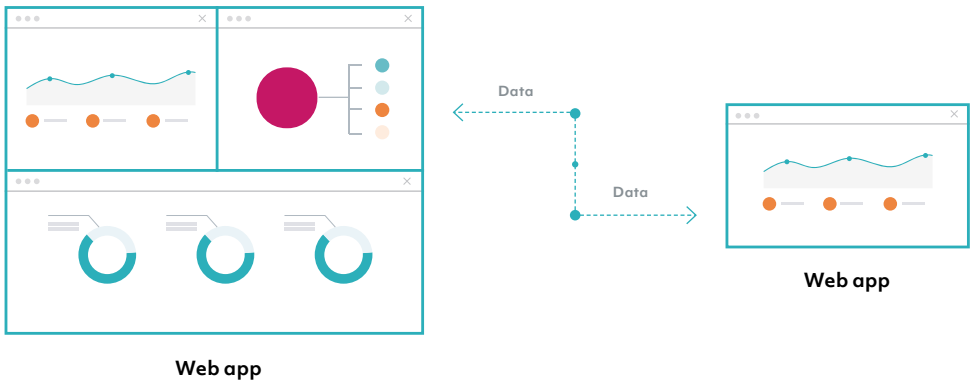
Alongside the ability to exchange data between applications, it can also help to provide some level of **visual integration**, where clusters of application windows are snapped together by the user to act as a single unit. This visual integration isn't purely cosmetic; if used effectively, it can improve workflows and operational efficiency.

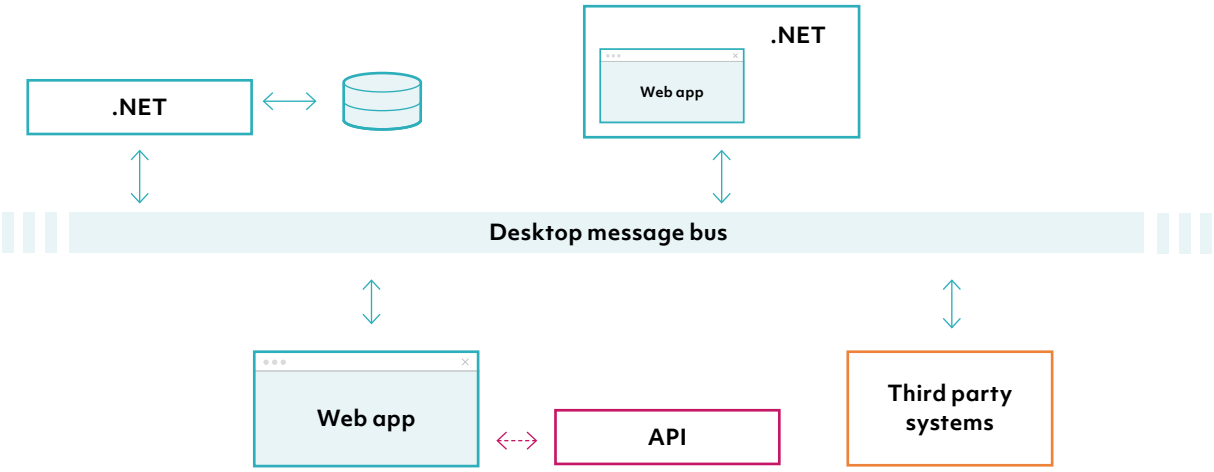
These techniques move beyond a like-for-like migration, delivering a solution that is tangibly better than the original. Users can assemble and connect a collection of applications, creating their own workspaces that are designed for their specific needs.

Migrating a suite of applications

When tackling the process of migrating a suite of applications, things become progressively more complicated. It is unlikely that you will migrate every application at the same time. Instead, high-value applications or functionality will tend to be migrated first, resulting in an application estate where both legacy and web applications co-exist. Furthermore, it may make sense to migrate a sizeable desktop application in 'chunks', following the familiar strangler pattern for gradual migration.

Fortunately, there are several different ways applications can be integrated, both old and new. Furthermore, there are various techniques that can be used to mix and match legacy and web technologies within existing applications.





Through the use of **legacy adapters**, it is possible to exchange data between legacy applications and those running within desktop containers via the **desktop message bus**. Furthermore, various vendors provide adapters for applications that are part of the standard desktop suite, including Excel and Outlook, or finance-specific applications such as Bloomberg and FactSet.

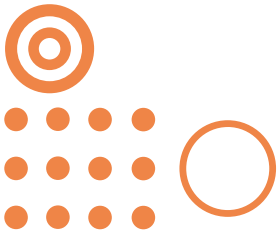
It is also possible to embed web applications within legacy applications through the use of a **web view**. This is effectively a desktop container that is hosted within another application. Creating a consistent user experience (both visually and functionally) in an application that mixes both legacy and web technologies can be quite challenging. Also, notably while it is possible to host web technologies within legacy applications, it is not possible to do the inverse. Web applications cannot host legacy applications.

Some vendor products allow visual integration of both legacy and web applications, allowing users to snap and dock a diverse range of applications in support of their desired workflow.

With desktop applications becoming increasingly integrated, there is a need for vendor-neutral standards in order to accelerate adoption and build an inclusive ecosystem. The FDC3 standard (founded by OpenFin and operated within FINOS) defines a common language for applications that communicate over the desktop message bus. Their standards are adopted by all the tool vendors, ensuring interoperability.

Some vendors also provide various value-add services and capabilities. Examples include an **app store** to facilitate discovery and installation of applications, and a **notification centre** to provide a single location for managing notifications emitted by various applications.

While desktop containers are the main building block for using web technologies on the desktop, there are a great many other tools and technologies which ease migration. They also provide cross-application functionality that takes a disparate collection of siloed applications and turns them into an integrated ecosystem.



From legacy to web technologies

Regardless of which tool you choose as your desktop container, and whether you adopt a message bus, or choose to support visual integration, the most significant change you will be making is in the programming languages and tools you use to build your applications, e.g. from C#/WPF to JavaScript/React.

One of the most challenging aspects of the web technology ecosystem is its fragmented and fluid nature. With C# and WPF for example, you get a fully integrated toolset that supports all aspects of development, test, build and distribution. It is very much a shrink-wrapped solution. In contrast, web development involves a bewildering array of choices: JavaScript or TypeScript? Parcel or Webpack? React or Angular? Creating a productive development environment that fulfils all of your development requirements (from development, test, CI and release) can be quite a challenge for the uninitiated.

A further challenge that often only becomes apparent a few years after adopting web technologies is their rate of change. There is a constant churn, with new frameworks, tools and libraries appearing almost every week, although pragmatism needs to be exercised – new doesn't necessarily mean better. A strategy around adopting web technologies needs to consider this churn; do you keep everything up to date, migrating to the latest frameworks as they emerge? Or do you create an architecture that allows different tools and frameworks to co-exist? There are pros and cons to each approach.

There is much that could be said about how to tackle these challenges, but that is not the focus of this paper. Probably the most important point to consider is the time it takes to make this transition; an investment in training and well-informed decision making will certainly pay dividends.

Tools and products

There are a number of different products, both free and commercial, that provide the various tools that support a desktop ecosystem (desktop container, message bus, legacy adapters etc ...). Here we give a brief overview of the various options.

Browser

The most obvious, yet often overlooked, vehicle for delivering applications to your user's desktops is the browser itself. Core capabilities such as Web Workers, WebGL, WebAssembly, Web Sockets and WebRTC allow the development of complex productivity applications that are far more than just simple data input forms. There is also ongoing work, under Project Fugu, to add features such as multi-monitor support, data-sharing, Near Field Communication (NFC) and much more.

Recently there's been a lot of excitement about Progressive Web Applications (PWA), a collection of features that allow the creation of mobile applications, using web technologies, that deliver a native-like experience. There is also growing PWA support with desktop browsers, allowing for off-line applications, launched from desktop icons, without the associated browser borders and toolbars.

Electron

Electron is a free and open source desktop container based on Chromium, the open source 'core' of the Google Chrome and Microsoft Edge browsers, coupled with the Node.js runtime. Introduced in 2013, it has become widely used for developing consumer-facing desktop applications using web technologies, including Spotify, MS Teams and Slack.

Electron applications are distributed by bundling the desktop container with the (web) application code, both of which are installed locally on the client machine. This mimics the 'traditional' installation experience of desktop applications and allows them to work offline.

When writing Electron-based applications, caution must be taken if any non-trusted content is consumed. The introduction of Node.js APIs means that malicious code can cause significant damage. However, these risks are well understood and documented, with various guides detailing the required mitigations[3].

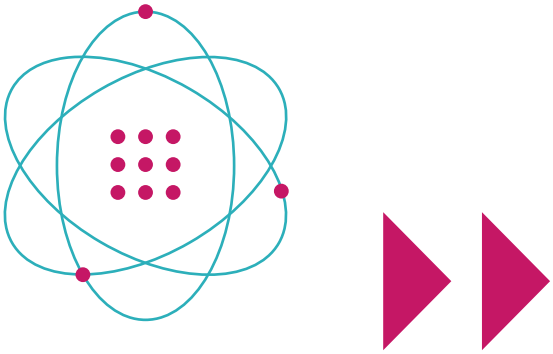
OpenFin

OpenFin was launched in 2010, by the company of the same name, and was originally branded as a Secure HTML5 runtime. Early versions were built on Chromium and as well as providing basic desktop container functionality, they provided a message bus, legacy adapters and Windowing APIs.

A notable feature of OpenFin, when compared to Electron, is its distribution model. End users install a single shared runtime with applications downloaded over HTTP in exactly the same way as browser-based web applications. This 'immediate' distribution model allows the roll-out of updates and new applications far more rapidly than the traditional installer-based model. In 2016 OpenFin moved to Electron, re-implementing their distribution model on top of this open-source codebase.

OpenFin, which has become widely adopted within financial services, has more recently rebranded as the Operating System for Finance, adding more features to their product including snap and dock via their Platform API, workspace management, notifications (which don't exist in Windows prior to v10) and a system tray. More features are on their way with OpenFin Desktop, which adds an app store and user identity management.

3 https://www.electronjs.org/docs/tutorial/security



Finsemble

Finsemble was launched in 2017 by ChartIQ, a company known for their financial charting products. Originally, Finsemble was designed as a desktop application framework on top of OpenFin, layering in capabilities not available at the time – including workspace management, customisable UI, context sharing, event routing, storage, authentication, and data feed management. More recent releases have added further features such as tabbed workspaces, tiling and an application launcher.

The core of Finsemble is centred around Workspaces, which provides window and workspace management, context sharing (including FDC3), global search, themeable UI and actionable notifications. Additional functionality is divided into three distinct categories: Flow, which provides a suite of UI capabilities; Connect, which provides various integration capabilities with existing infrastructure; and Native, which allows visual and logical integration of legacy apps into the workspace.

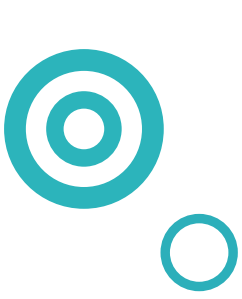
In 2019, Finsemble replaced the underlying OpenFin container and runtime with their own policy-based security wrapper based on Electron. In 2020, Finsemble open-sourced their Secure Electron Adapter through FINOS.

Glue42

Glue42, by the company Tick42, was launched in 2012 with an initial focus on desktop interoperability. The original product facilitated trading workflows, supported by the integration of applications built using Microsoft .NET.

In 2013, a Chromium-based desktop container was added to the product. The features also expanded to include a global search service, workflow management with swimlanes, notification services, application connectors, and various other tools for building a desktop ecosystem. They also provide connectors for various third-party products including Eikon, Bloomberg and FactSet.

More recently, Glue42 has moved to an Electron-based container. They are also acknowledging the impact that Progressive Web Applications could have on the desktop, with their new open-sourced product, Glue42 Core, being based on this technology.



Conclusion

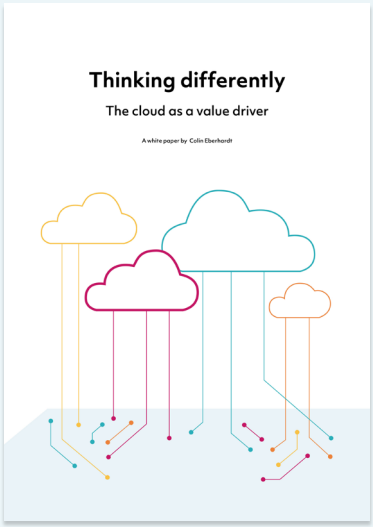
Managing the suite of tools which are so critical to the running of every financial services firm is a challenge. Meeting user expectations with a heterogeneous collection of legacy applications and a meagre development budget is an impossible challenge.

Web technologies, whether used within the browser or a desktop container, are the de facto choice for application development. However, we've seen throughout this paper that this is not an all-or-nothing option. You don't have to migrate each and every application; instead, you can selectively migrate individual applications (or parts of), leveraging various open source libraries and vendor products to knit these together into a streamlined application ecosystem.

So how do you get started on this migration journey? In simple terms, the first step is to form the strategy itself – determine your level of investment, which application(s) should and can be tackled first. In tandem, it is very important that your teams become fluent and comfortable in web technologies – there is no questioning their importance for the future. Finally, consider tools and product options that support your strategy and vision.

White papers by Scott Logic

You'll find more white papers, practical guides and technical articles on our website – please visit **blog.scottlogic.com**



Thinking differently

One of the greatest technology enablers of the past decade is public cloud. The strategic importance of this has been widely accepted by the industry; however, the prevailing focus on the cloud as a means to reduce costs, is overlooking its greatest capability: agility! This paper encourages you to think differently and see the cloud as a value driver, providing a platform for change and a foundation for business agility.

→

The journey to DevOps

The case for the public sector to innovate around service delivery, whilst driving cost-savings and improving efficiency, has never been greater. This paper is written for technology managers within such organisations, who face the significant challenge of designing and delivering transformative digital services.

→

If you'd like a copy of these sent to your inbox, please email **colin@scottlogic.com**

Notes

Want to discuss how to plan your integration journey?

At Scott Logic, we've supported a wide range of clients in establishing and cultivating an integrated desktop application ecosystem, boosting staff productivity.

If you'd like to discuss how your organisation can take a pragmatic, incremental approach to creating the desktop ecosystem of the future, we're always happy to chat.

Please contact Colin Eberhardt on:

+44 333 101 0020

colin@scottlogic.com

SCOTT LOGIC / ALTOGETHER SMARTER

3rd Floor, 1 St James' Gate
Newcastle upon Tyne
NE1 4AD

+44 333 101 0020

scottlogic.com