Microsoft

SharePoint

Deciding between apps for SharePoint and SharePoint solutions

Deciding between apps for SharePoint and SharePoint solutions

Keenan Newton
Microsoft Corporation
May 2013

Applies to: SharePoint 2013 | Office 365

Summary: This white paper describes the issues you should consider when deciding whether to write an app for SharePoint or a SharePoint solution. It covers scenarios in which an app is an ideal choice versus a SharePoint solution.

Contents

Overview of the Cloud App Model 3
- App shapes 3
- Hosting options 4
- App trust and permissions 5

Factors to consider 6
- Server object model vs. client object model 6
- Obsolete APIs 7
- Environment 7
- Deployment 8
- Design pattern 8
- Skill sets 8

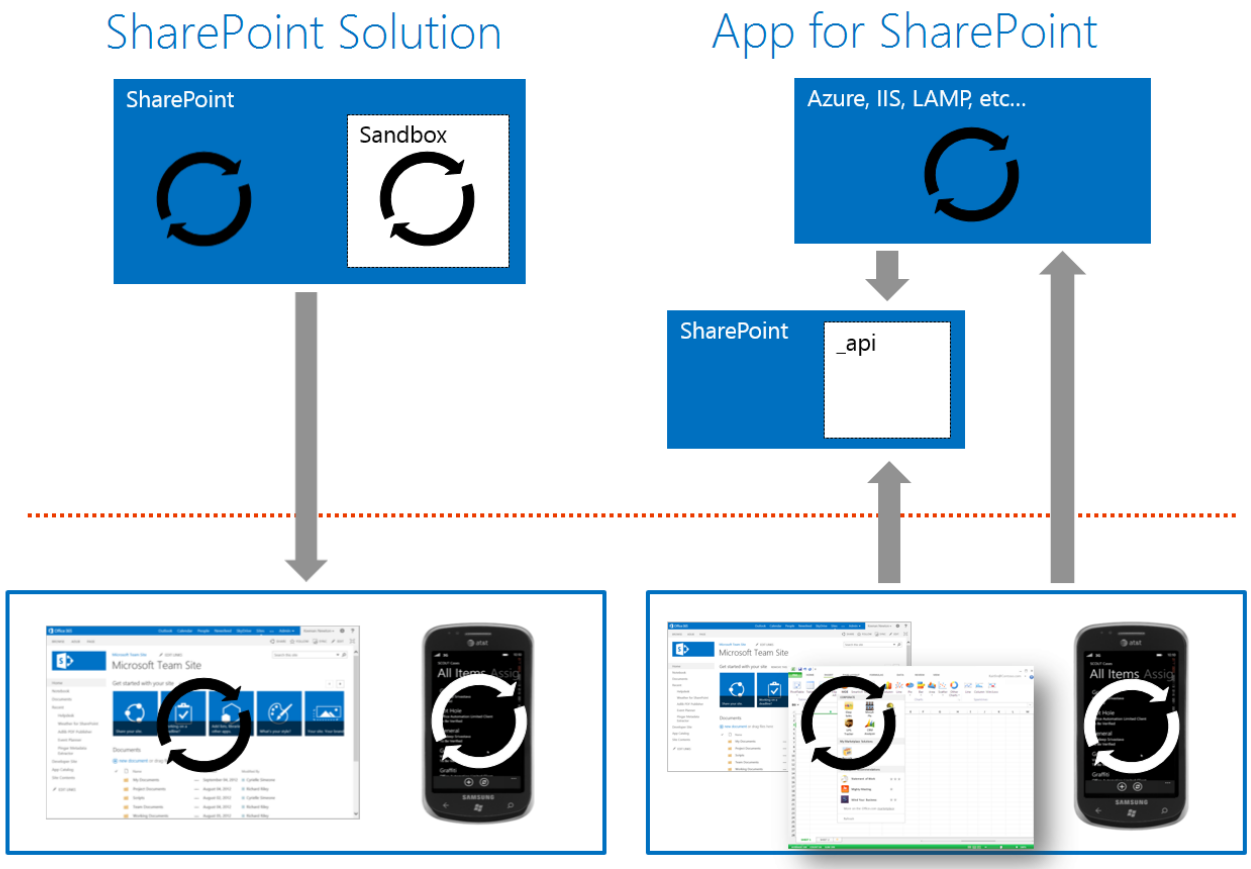
Scenarios 9
- Branding 9
- Line-of-business 10
- Forms and views 10
- Handling events 11
- Search and workflow 11

Making the decision 11

Overview of the Cloud App Model

SharePoint 2013 has introduced a new way to customize SharePoint by using *apps*. This new light-touch, SharePoint-as-a-service and surface approach allows you to create SharePoint customizations that are integrated with the SharePoint experience without the impact that full-trust code SharePoint solutions can have on the life cycle and management of the SharePoint environment.

The following illustration shows how an apps execution differs from a typical SharePoint solution.



In a typical SharePoint solution, as shown on the left side of the image, your code executes within the SharePoint execution environment. This means that your custom code could have a negative impact on that SharePoint environment. In the case of apps for SharePoint, shown on the right, your code executes in a completely separate environment, whether it is in the browser or in a separate web application hosting environment. This environment can be anywhere of your choosing as long as it supports the generation of webpages (that is, a web server).

App shapes

SharePoint 2013 allows for multiple ways to interact with and access apps. The key method used to determine how your app will be interacted with is to define the one or more app shapes. App shapes

define the experience for interacting with and using an app. There are currently three app shapes available:

Immersive (or full-page app): This shape provides a fully immersive experience by using the entire page. While this gives you complete control over the app experience, it is important to make sure that your app properly links back to the SharePoint site, so the user experience feels integrated and not lost. To make this simple, we provide a chrome control that not only lets your app automatically point a user back to their SharePoint environment, but it also provides your app with the current SharePoint cascading style sheet (CSS). So when SharePoint changes its CSS, say, through a theming change, your app will change its look and feel as well.

App part: An app part is an IFrame hosted in a Web Part that is represented by the **ClientWebPart** class. The IFrame can take parameters which can change the user experience of the app part. You can think of these app experiences as similar to standard Web Parts, the difference being they do not interact in the same way legacy Web Parts do.

Extension app: Extension apps are just custom actions that can direct to a page in your app. This could be a page hosted on SharePoint, or, in the case of a cloud-hosted app, a remotely hosted webpage. You can think of extension apps as entrance points to your app. They don't provide a user experience in themselves.

Hosting options

Hosting apps is not as simple as hosting SharePoint solutions. With a SharePoint solution, the hosting options are quite clear: it is hosted on a SharePoint server and it runs within the SharePoint Application Domain. While this makes selection very easy, running custom code within the application domain of SharePoint presents a lot risk to a SharePoint environment, and you have to manage everything from application life cycle of your customizations to the upgrade cycle of your SharePoint environment. Apps let you manage the customization independently of the SharePoint environment because they use SharePoint as a service and portal as opposed to running within SharePoint. This requires you to determine how you want to host your app. Hosting is determined based on the needs of your app.

There are two main hosting options for SharePoint: *Cloud-hosted apps* and *SharePoint-hosted apps*.

Cloud-hosted apps: Cloud-hosted apps are apps that are hosted remotely from SharePoint and can contain some sort of server-side logic. They can be hosted on any platform of your choice, whether it is Windows Azure, Internet Information Services (IIS), or even a PHP server running on Linux.

There are two main flavors of cloud-hosted apps: the first is a *provider-hosted app*. In provider-hosted apps, the developer of the app determines where the app is hosted. The other type is an *autohosted app*. This hosting option, which is currently only available on Office365, takes the components that the developer built for the app and deploys them at installation time to a special Windows Azure website and/or database that is designated for the SharePoint instance. This lets a developer develop apps without having to determine where to host the app.

SharePoint-hosted apps: SharePoint-hosted apps contain only declarative content, such as HTML and JavaScript, and don't have any server-side code that is packaged with them. These types of apps store the HTML and other content in SharePoint and simply present the declarative content to the browser.

SharePoint-hosted apps interact with SharePoint via the JavaScript object model or Representational State Transfer (REST) end-points and are client-side based. The design and interaction of these types of apps are similar to client-side design patterns used with sandboxed solutions.

It should be noted that SharePoint-hosted apps are limited to declarative content and cannot contain server side code. However, cloud-hosted apps are not limited to only remotely hosted content. Cloud-hosted apps can also contain declarative content that is hosted in SharePoint, such as webpages, CSS, content types, and so on.

App trust and permissions

With full-trust solutions, your code always had the same permissions as the SharePoint environment. Although this provides great power, it also requires a great deal of care and restraint because you can write code that could harm your SharePoint environment. A few examples of practices you have to watch out for in SharePoint full-trust code development are:

- Instantiating an **SPSite** object inside an event receiver
- Enumerating over **SPList.Items** or **SPFolder.Files**
- Deleting multiple versions of a list item
- Using unbounded **SPQuery** objects
- Heavy use of **SPSite** and **SPWeb**
- Always remembering to dispose of SharePoint objects
- Accidential changes of the Web Part signature

Sandboxed solutions helped prevent that, but they had their own limitations as well.

Since apps run outside of the SharePoint execution context, there needs to be some way of trusting them to ensure that an app does only what it is supposed to do.

High-trust applications

A high-trust application is a provider-hosted application for SharePoint for use on-premises, which uses the server-to-server (S2S) protocol. “High-trust” is not the same as “full-trust,” and high-trust does not mean the application has full trust. A high-trust application must still request application permissions. The application is considered “high-trust” because it is trusted to use any user identity that the application needs; the application is responsible for creating the user portion of the access token.

A high-trust application is built for use in an on-premises environment. It is not intended to run in Office 365. Applications that use the server-to-server protocol would typically be installed behind the firewall in instances that are specific to each individual company.

A high-trust application is a provider-hosted application for an on-premises installation of SharePoint. A high-trust application uses a certificate instead of a context token to establish trust. In SharePoint 2013, the **SPTrustedSecurityTokenIssuer** provides access tokens for server-to-server authentication.

Low-trust applications and app permissions

Low-trust applications are those that use a trust broker, such as Windows Azure Access Control Services (ACS), to act as a token issuer between SharePoint and the application. Windows Azure Access Control Services will of course require an Office 365 subscription.

SharePoint requests a context token from ACS, which it then sends to the location hosting the application. The application then uses the context token to extract the refresh token and uses the refresh token to request an access token from ACS. When the access token is received, it is used by the application to talk back to SharePoint.

SharePoint low-trust applications rely on the OAuth authorization code flow ("grant type") to delegate limited rights to applications to act as users. For this to work, both SharePoint and the client application (the SharePoint application) must trust and communicate with a token issuer. SharePoint relies on Windows Azure Active Directory (AD). Windows Azure AD, in turn, must be aware of SharePoint and the client application in order to grant them the necessary tokens to work together.

Low-trust applications can't interact with content outside of the app web, which is the SharePoint sub web that contains your app by default. To do this, you need to specify permissions to the host web. The host web is the SharePoint web that hosts the app web for the app. These requested permissions are then approved or denied by the installer of the app during installation. Denying the permissions (that is, not trusting the app) prevents the app from installing.

Factors to consider

Your decision to create an app versus a SharePoint solution will be determined by two things: first, the capabilities of each customization type, and second, the actual scenario you are trying to solve. Let's take a look and compare apps for SharePoint versus SharePoint solutions. Throughout this section, we will focus on full-trust code (SharePoint solutions) instead of partial-trust code (sandboxed solutions). Partial-trust user code is deprecated as of SharePoint 2013. If you are looking at partial-trust code, you should consider writing an app instead.

Server object model vs. client object model

Writing SharePoint solutions required you to use the SharePoint server object model. This object model offers a very complete coverage of access to your SharePoint content and the SharePoint environment. The downside to this object model is that it requires a great deal of care because your code runs within the context of SharePoint, and if you have a bug or some other performance issue with your code, it could, and typically would, negatively affect the SharePoint environment.

The SharePoint client object model, made popular with sandboxed solutions and now greatly enhanced for apps for SharePoint, provides APIs that remotely connect to and interact with SharePoint. The benefit here is that your code cannot directly affect the SharePoint environment. If your code fails or has performance issues, it affects only your app and not potentially the entire SharePoint environment.

The following table highlights the strengths and capabilities of the server object model (SSOM) and client object model (CSOM).

Table 1. SSOM vs. CSOM

Scenarios	SSOM	CSOM
Administration	Yes	No
Content management	Yes	Yes
Site management	Yes	Yes
Site branding	Yes	No
Synchronous execution	Yes	Yes
Asynchronous execution	No	Yes
Batched requests	No	Yes

Obsolete APIs

With the introduction of the Cloud App Model and the new REST endpoints (/_api/), some of the pre-existing service endpoints are now obsolete. These obsolete APIs include the following web services:

- Listdata.svc
- ASMX web services

Any code that currently uses these obsolete web service endpoints should be refactored to use the equivalent REST endpoint (/_api/) or the Client Side Object Model.

Environment

The environment where you are running SharePoint is a critical deciding factor. If your SharePoint environment is Office365, you have to use the Cloud App Model and develop all of your customizations as apps for SharePoint. If your SharePoint environment is on-premises, you have the choice of apps or full-trust solutions. Keep in mind though, if you want the flexibility to support both on-premises and cloud environments, then you will want to write your solution as an app because full-trust code is not supported in Office365. The environments you decide to target will also impact how your app will authenticate with SharePoint. Essentially, if you plan to support both cloud and on-premises environments, you will need to make your app low-trust. If you intend to support only on-premises, you have the choice of high-trust or low-trust apps. See the App trust and permissions section earlier for more details on app trust.

Deployment

SharePoint solutions require a significant amount of planning in order to deploy them. Typically, you have to plan an outage in order to reset the application pools. This is never ideal. Apps, on the other hand, can be deployed at any time, since they don't require an outage because they run remotely of the SharePoint environment. This means an app can be deployed and installed at any time of day with little or no impact to the environment. This doesn't mean an app can't affect the content in the SharePoint environment, because it very much can, such as the use of the app event receiver, which can execute code during an installation. It simply means that you don't need to reset your SharePoint environment.

Apps for SharePoint are ultimately deployed to either the Office Store or to the internal app catalog. The Office Store gives the developer the potential to monetize the app or to provide it for free to the public. The app catalog is available within the context of an internal organization's network and only available to the employees of that organization. In either scenario, the app package must be generated in order to deploy the app. The exact type of deployment depends on the type of environment that the app is being deployed to. For example, for an app that will be published as a provider-hosted app, the publication process in Visual Studio 2012 generates the app manifest file (the .app file), as well as multiple files for the web application. The web files and components then have to be deployed to the web application host environment. If you publish your app as an autohosted app, the Visual Studio 2012 publication process generates not only the app manifest .app file, but also the files and script that you will use to publish the host web application, and it embeds the web files and script into the app package. When an autohosted app is deployed to the SharePoint environment, it automatically deploys the web files to the autohosted web environment and executes the SQL scripts against the autohosted SQL Server databases.

Design pattern

SharePoint solutions allow for really only one type of web development pattern, which is web forms. Although this has been a time tested and true way of developing solutions for years, there have been many advances in web applications that may now be architected using the MVC (Model-View-Controller) and MVVM (Model-View-ViewModel) design patterns. Both design patterns separate the view from the logic of an application, thus allowing you to have multiple views. This can be handy to support multiple views for one set of logic. Say an app for SharePoint could be one view, and an app for Excel could be another view. Both of these views, however, can use the same set of domain logic. In the case of web forms, you would end up rewriting a lot of domain and render logic for both the SharePoint solution and the Office solution.

The Cloud App Model allows for a loosely coupled environment between the SharePoint surface and the domain logic itself. This allows you to build multiple views that can use the same logic over and over again. One example would be creating a full immersion (full page) experience and an app part. Both of these experiences or views can use the same domain logic, and you can take advantage of the common and popular frameworks and tools that are available to the development community.

Skill sets

SharePoint solutions require that you have .NET, ASP.NET and SharePoint development skills in order to build a solution. It requires a specialized skill set to build scalable, reliable, and valuable solutions. You have to understand the proven development practices for SharePoint in order to maintain a stable and

reliable environment. The Cloud App Model, on the other hand, gives you incredible flexibility with the languages and technologies used to create the new apps for SharePoint. For example, depending on the app shape used (as described earlier), the application and data logic can reside in the cloud or on-premises whether it is running on Windows Azure and Windows Azure SQL Database, some other cloud service, or running locally on any web server. Because the Cloud App Model supports a tiered architecture approach to building apps, the user interface, logic, and data can be distributed among the various tiers and use different programming languages and design paradigms. You can write apps in virtually any web language that can use standard web-based standards, such as HTML5, JavaScript, OAuth, and REST. This lets you code the application logic of the app for SharePoint in any language and any platform that you are comfortable with. The primary visual and user experience aspects of the app are coded in JavaScript and HTML, which are basic web standards that any developer should know for standardized web development that can be accessed on any browser.

The Cloud App Model also can be used as a platform service to support apps written for other platforms, including Windows Phone 8, Windows 8, iOS, and Android. You can write the presentation layer on these platforms in the native languages and paradigms available on those platforms, and then reference the SharePoint APIs through JavaScript libraries, REST, and WP and .NET libraries.

The power of this flexible development model lets you start developing apps for SharePoint for immediate consumption by either enterprise customers or consumers through the Office Store, without needing to ramp up on a new development language. You can thereby provide value to your enterprise and independent customers immediately, and you have the opportunity to monetize your app through the Office Store, or provide it exclusively in an enterprise isolated app catalog. Either way, you can provide value immediately with your existing skill set, which makes the apps for SharePoint model an obvious choice for value-added, rapidly developed apps.

Scenarios

SharePoint is currently being used throughout enterprises and in many hosted ISV deployments. As part of these deployments, many customizations were added to the platform to make it a more usable and productive environment for its users. In this section, we will delve into several key customization scenarios that have been deployed on the previous versions or SharePoint, and assess the viability of migrating these scenarios to the new apps for SharePoint model.

Branding

Most instances of SharePoint that are currently deployed in the enterprise and ISV spaces have been branded with custom colors, styles, layouts, and logos in order to customize the SharePoint product to the corporate and provider identities. These solutions are provided on the portal-level across the entire SharePoint deployment to offer users a look and feel that is unique to the corporation and help drive the corporate culture and identity. These solutions are often deployed as a collection of master pages, page layouts, styles, and images that are grouped and deployed as a SharePoint solution. Given the scope of these customizations and that they are offered to the users of the entire SharePoint deployment, the solution must reside on the SharePoint servers. These solutions are often housed and deployed from the sandbox to avoid any problematic code from affecting the entire SharePoint farm. Therefore, any branding-type solution cannot by nature be migrated to an app for SharePoint. This doesn’t mean an app can’t provision brandied elements, it just means that it shouldn’t be done in the

same way as a solution. For instance, you could use an app to provision branded content into your SharePoint environment. Another option for branding a SharePoint site would be using the new Design Manager feature in SharePoint.

For apps, the branding can be pulled back into the app for SharePoint instance, and the app can use the custom branding solution already deployed to the SharePoint instance. This is powerful in that the branding investment can be used in the new apps without the need to rewrite any code. For more information about the types of branding and styling options that you can use in the various hosting options, see UX design for apps in SharePoint 2013.

Line-of-business

Another type of common customization performed on top of the SharePoint platform is a line-of-business solution. These solutions often use the capabilities of the SharePoint platform to connect users with in-context resources, such as BI dashboards, business data from external sources, custom workflow processes, platform automations and productivity enhancers for information workers, connections to ERP and CRM systems, and connections through Business Connectivity Services (BCS) in SharePoint to various external data sources and structures. In the past, these types of solutions would have been deployed as custom SharePoint solutions that would have to be added to the SharePoint server and provided with full-trust code access permissions to the SharePoint deployment and cause downtime each time they are updated.

Apps for SharePoint offer a secure alternative by providing the flexibility of having the actual line-of-business application code running in a variety of hosting options, and the in-context app running within the browser for particular users. The hosting options provide an easy alternative to solution hosting that limits the exposure of the SharePoint servers to the application logic and process without eliminating the ability to allow for business-critical applications that must interface with SharePoint. Apps for SharePoint can also access BCS natively through the SharePoint CSOM APIs or REST endpoints and display the data in a custom way through JavaScript. The additional benefit to having the line-of-business application logic hosted outside of SharePoint and only the presentation of the user interface and input surfaced through the app for SharePoint is that the line-of-business application can run on potentially any external server and in any language that has the capability of being exposed through web services. The various remote APIs available through the app model allow the app for SharePoint to communicate with these web services and display the user-relevant business information in the browser. This is incredibly powerful for businesses that have invested resources into building line of business applications on other platforms and languages, but would like to integrate them into their SharePoint deployment.

Forms and views

Forms and views are two more customizations that have been historically widely used within the SharePoint customization options. Forms and views let the enterprise and hosting providers give users a custom way to interact with data hosted inside of SharePoint or available through SharePoint from external data sources. These custom presentation options for data let users determine the most productive way in which to use and manipulate the SharePoint or external data.

Before SharePoint 2013, the common way to create forms and views was either with a custom ASPX webpage and/or using Extensible Stylesheet Language Transformations (XSLT). Apps for SharePoint let

you surface the same type of custom data display and user input through an app rather than a full-trust code solution. An app for SharePoint allows for complete choice in how you render a form or view. You can use server-side rendering like XSLT or custom code, or it can be done client side with HTML. You have this flexibility because your app is not bound by the architecture of SharePoint to execute your code. Instead, you can use the architecture and framework of the web hosting platform that your app resides on. Thus you can easily use common web development patterns, such as Model-View-Controller and Model-View-ViewModel. This provides the same benefits as outlined earlier to the security and flexibility of apps for SharePoint, but it also allows for the same type of data display flexibility and user input customization as previously available in the full-trust code solution model. Therefore, any new forms and views that are available to enterprise users or ISV-hosted users can be created using the new apps for SharePoint model, rather than relying on full-trust code solutions.

Handling events

With SharePoint solutions, handling events is done with event receivers. You register an event, such as a list item added or updated, with a block of code, and whenever that list item is updated or something is added to the list, your code executes. It is a very popular feature, but it exposes the same risks that all full-trust code can to your environment.

The Cloud App Model introduces a concept called *remote event receivers*. They behave like event receivers, except that the custom code you write is executed remotely. SharePoint calls a Windows Communication Foundation (WCF) web service that you write, and that service executes code when the event occurs. This custom code can call back into SharePoint using REST endpoints or CSOM APIs just like an app.

SharePoint solutions also have feature receivers that execute when a feature is installed or removed. Apps have an equivalent remote event receiver called an *app event receiver*. These app event receivers have the same execution model as other remote even receivers, except that they execute when an app is installed, upgraded, or uninstalled.

Search and workflow

SharePoint 2013 introduced changes to some SharePoint services and web service end points. First, the Search service and workflow engines have changed: now SharePoint uses a new and improved search engine by default. This requires reevaluation of any existing solutions you have and the approach you take in writing apps or solutions that use search. The workflow engine in SharePoint 2013 is rewritten from the ground up. It follows the new app model and runs remotely from the SharePoint environment. In the past, you could write custom workflow activities with custom code. That is no longer the case in SharePoint 2013: custom activities are now HTTP requests that are defined declaratively. This allows for a scalable and stable workflow environment and will require any custom code to be defined in a web service.

Making the decision

The decision to write your next SharePoint customization as an app or full-trust solution should take into account the scenario you are trying to satisfy as well as the current environment landscape you are trying to target. If you are building for Office365, the choice is pretty clear that apps are the way to go. If you are building a customization for the Central Admin, then a full-trust solution is the way to go. For

some scenarios, you can choose either option. Typically, most organizations want to pick the best scalable method for customizing SharePoint while minimizing the effort to get the customization done. Typically full-trust code must be reevaluated when you are upgrading SharePoint to ensure that the customizations will not negatively impact the upgrade to the next version. Apps do not require you to do this at the same scale as full-trust code due to their loosely coupled nature. With an app, your main concern is to ensure that the data contract on the SharePoint service has not changed. With full-trust code, you have to be concerned with not only the data contract of the API, but also the behavior of the API and the environment your solution is running in. This includes both the SharePoint environment and the impact that any other custom solution may have on the SharePoint environment.

Full-trust code requires a deep understanding of the internal workings of SharePoint to build stable and scalable solutions. Apps require you to just understand the object model you are using because an app will not impact the performance or stability of the SharePoint environment. Another thing to consider is the application lifecycle of developing an app versus a full trust SharePoint solution. It is true that there is always some learning curve involved in learning different technologies, methodologies and programming languages. However, that is not the only factor to consider. Full trust code requires extensive lifecycle management processes and procedures to not only make sure the app itself works as expected, but to insure that the SharePoint environment including any other full trust customizations is also stable. Due to the isolated nature of apps, apps, in most cases, will not require that your application lifecycle processes test the SharePoint environment and other customizations along with your app.

With these considerations in mind, apps should be looked at as the primary choice, and full-trust solutions should be looked at when the capability of the app model does not meet the business requirements. I want to emphasize meeting the *business* requirements and not the *technical* requirements.

It is true that how you implement a full-trust solution is not identical to implementing an app. However, don't get trapped into mapping your server-side object model development skills to the Cloud App Model. There are differences, and the Cloud App Model will require you to think outside of the "full-trust box."

The main reason why you would still use full-trust solutions is that a feature or API you want to use is not yet available through the REST endpoints or CSOM APIs. However, this doesn't mean you can't still use the server object model from an app that is on-premises. It just requires some outside-of-the-box thinking. Instead of writing a full-trust solution that completely covers the entire business scenario you are trying to satisfy, write a full-trust solution that exposes the functionality you are looking for as a REST endpoint and write an app that can use that endpoint. This will allow your solution to scale while reducing the overall exposure of full-trust code to the SharePoint environment.