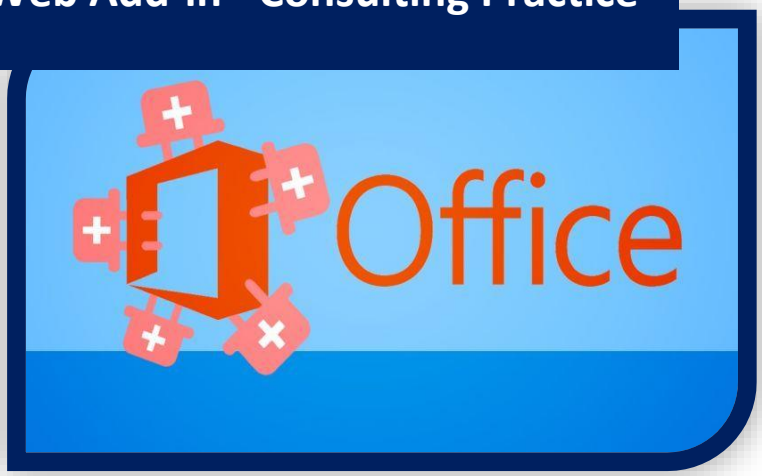
Microsoft Office 365 Word - Web Add-in - Consulting Practice



Cognitive Convergence is Subject Matter Expert in Office 365, Dynamics 365, SharePoint, Project Server, Power Platform: Power Apps-Power BI-Power Automate-Power Virtual Agents.
Our Microsoft Office 365 Consulting, add-in Development, Customization, Integration services and solutions, can help companies maximize business performance, overcoming market challenges, achieving profitability and providing best customer service.

CONTENTS

OBJECTIVE

This case study is a brief introduction about the core concept of MS-Word Web Add-in, its fundamentals, building architecture & its development tracks.

MS WORD ADD-IN

Word add-ins are one of the many development options that users have on the Office Add-ins platform. Users can use add-in commands to extend the Word UI and launch task panes that run JavaScript that interacts with the content in a Word document. Any code that the user can run in a browser, can run in a Word add-in. Add-ins that interact with content in a Word document create requests to act on Word objects and synchronize object state.

Office Add-ins can do almost anything a webpage can do inside a browser. Users can use the Office Add-ins platform to:

- **Add new functionality to Office clients** - Bring external data into Office, automate Office documents, expose third-party functionality in Office clients, and more. For example, use Microsoft Graph API to connect to data that drives productivity.
- **Create new rich, interactive objects that can be embedded in Office documents** - Embed maps, charts, and interactive visualizations that users can add to their own Excel spreadsheets and PowerPoint presentations.

COMPONENTS OF AN OFFICE ADD-IN

An Office Add-in includes two basic components: an XML manifest file, and a user's web application. The manifest defines various settings, including how your add-in integrates with Office clients. User's web application needs to be hosted on a web server, or web hosting service, such as Microsoft Azure.

Manifest

The manifest is an XML file that specifies settings and capabilities of the add-in, such as:

- The add-in's display name, description, ID, version, and default locale.
- How the add-in integrates with Office.
- The permission level and data access requirements for the add-in.

Web app

The most basic Office Add-in consists of a static HTML page that is displayed inside an Office application, but that doesn't interact with either the Office document or any other Internet resource. However, to create an experience that interacts with Office documents or allows the user to interact with online resources from an Office host application, the user can use any technologies, both client and server-side, that the hosting provider supports (such as ASP.NET, PHP, or Node.js). To interact with Office clients and documents, the user uses the Office.js JavaScript APIs.

EXTENDING AND INTERACTING WITH OFFICE CLIENTS

Office Add-ins can do the following within an Office host application:

- Extend functionality (any Office application)
- Create new objects (Excel or PowerPoint)

Extend Office functionality

You can add new functionality to Office applications via the following:

- Custom ribbon buttons and menu commands (collectively called “add-in commands”)
- Insertable task panes

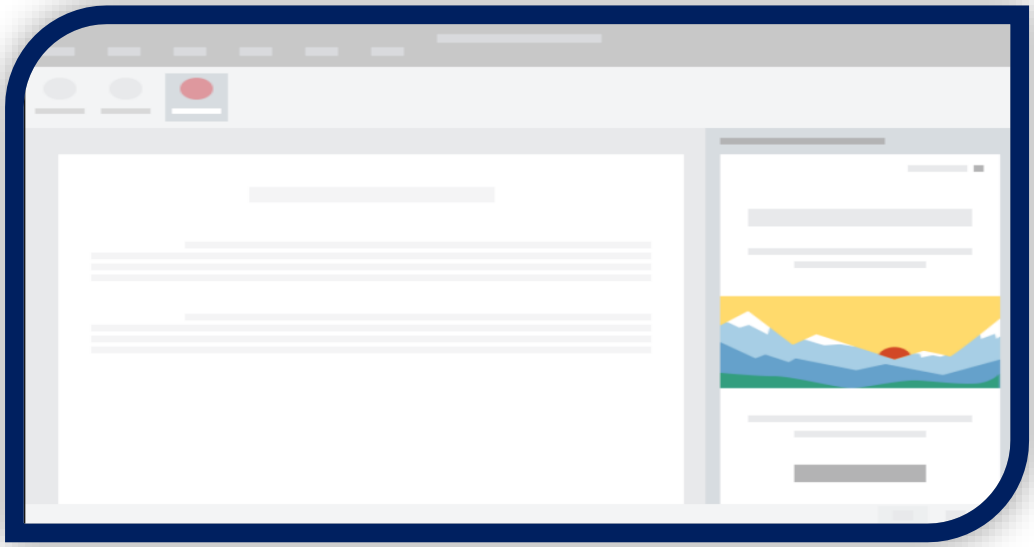
Custom UI and task panes are specified in the add-in manifest.

Custom buttons and menu commands

Users can add custom ribbon buttons and menu items to the ribbon in Office on the web and Windows. This makes it easy for users to access the developed add-in directly from the Office application. Command buttons can launch different actions such as showing a task pane with custom HTML or executing a JavaScript function.

Task panes

Users can use task panes in addition to add-in commands to enable users to interact with the solution. Clients that do not support add-in commands (Office 2013 and Office on iPad) run your add-in as a task pane. Users launch task pane add-ins via the My Add-ins button on the Insert tab.



OFFICE JAVASCRIPT APIS

The Office JavaScript APIs contain objects and members for building add-ins and interacting with Office content and web services. There is a common object model that is shared by Excel, Outlook, Word, PowerPoint, OneNote, and Project. There are also more extensive host-specific object models for Excel and Word. These APIs provide access to well-known objects such as paragraphs and workbooks, which makes it easier to create an add-in for a specific host.

JAVASCRIPT APIS FOR WORD

Users can use two sets of JavaScript APIs to interact with the objects and metadata in a Word document. The first is the Common API, which was introduced in Office 201x. Many of the objects in the Common API can be used in add-ins hosted by two or more Office clients. This API uses callbacks extensively.

The second is the Word JavaScript API. This is a strongly-typed object model that a user can use to create Word add-ins that target Word 201x on Mac and Windows. This object model uses promises and provides access to Word-specific objects like body, content controls, inline pictures, and paragraphs. The Word JavaScript API includes TypeScript definitions and vsdoc files so that you can get code hints in your IDE. Currently, all Word clients support the shared Office JavaScript API, and most clients support the Word JavaScript API.

Microsoft highly recommends starting with the Word JavaScript API because the object model is easier to use. Use the Word JavaScript API if the need is to:

- Access the objects in a Word document.

Use the shared Office javascript API if the need is to:

- Target Word 201x.
- Perform initial actions for the application.
- Check the supported requirement set.
- Access metadata, settings, and environmental information for the document.
- Bind to sections in a document and capture events.
- Use custom XML parts.
- Open a dialog box.

WORD PACKAGE

CLASSES	
Word.Application	Represents the application object.
Word.Body	Represents the body of a document or a section.
Word.ContentControl	Represents a content control. Content controls are bounded and potentially labeled regions in a document that serve as containers for specific types of content. Individual content controls may contain contents such as images, tables, or paragraphs of formatted text. Currently, only rich text content controls are supported.
Word.ContentControlCollection	Contains a collection of Word.ContentControl objects. Content controls are bounded and potentially labeled regions in a document that serve as containers for specific types of content. Individual content controls may contain contents such as images, tables, or paragraphs of formatted text. Currently, only rich text content controls are supported.
Word.CustomProperty	Represents a custom property.
Word.CustomPropertyCollection	Contains the collection of Word.CustomProperty objects.
Word.CustomXmlPart	Represents a custom XML part.
Word.CustomXmlPartCollection	Contains the collection of Word.CustomXmlPart objects.
Word.CustomXmlPartScopedCollection	Contains the collection of Word.CustomXmlPart objects with a specific namespace.
Word.Document	The Document object is the top-level object. A Document object contains one or more sections, content controls, and the body that contains the contents of the document.
Word.DocumentCreated	The DocumentCreated object is the top-level object created by Application.CreateDocument. A DocumentCreated object is a special Document object.

Word.DocumentProperties	Represents document properties.
Word.Font	Represents a font.
Word.InlinePicture	Represents an inline picture.
Word.InlinePictureCollection	Contains a collection of Word.InlinePicture objects.
Word.List	Contains a collection of Word.Paragraph objects.
Word.ListCollection	Contains a collection of Word.List objects.
Word.ListItem	Represents the paragraph list item format.
Word.Paragraph	Represents a single paragraph in a selection, range, content control, or document body.
Word.ParagraphCollection	Contains a collection of Word.Paragraph objects.
Word.Range	Represents a contiguous area in a document.
Word.RangeCollection	Contains a collection of Word.Range objects.
Word.RequestContext	The RequestContext object facilitates requests to the Word application. Since the Office add-in and the Word application run in two different processes, the request context is required to get access to the Word object model from the add-in.
Word.SearchOptions	Specifies the options to be included in a search operation. To learn more about how to use search options in the Word JavaScript APIs, read Use search options to find text in the Word add-in.
Word.Section	Represents a section in a Word document.
Word.SectionCollection	Contains the collection of the document's Word.Section objects.
Word.Setting	Represents a setting of the add-in.
Word.SettingCollection	Contains the collection of Word.Setting objects.
Word.Table	Represents a table in a Word document.
Word.TableBorder	Specifies the border style.
Word.TableCell	Represents a table cell in a Word document.
Word.TableCellCollection	Contains the collection of the document's TableCell objects.
Word.TableCollection	Contains the collection of the document's Table objects
Word.TableRow	Represents a row in a Word document.
Word.TableRowCollection	Contains the collection of the document's TableRow objects.

CREATE THE ADD-IN

Users can create an Office Add-in by using the Yeoman generator for Office Add-ins or Visual Studio. The Yeoman generator creates a Node.js project that can be managed with Visual Studio Code or any other editor, whereas Visual Studio creates a Visual Studio solution.

METHOD 1 - OFFICE 365 WORD WEB ADDIN - YEOMAN GENERATOR – VISUAL CODE

Prerequisites

1. Node.js (the latest LTS version)
2. The latest version of Yeoman and the Yeoman generator for Office Add-ins. To install these tools globally,

```
npm install -g yo generator-office
```

run the following command via the command prompt:

Create the add-in project

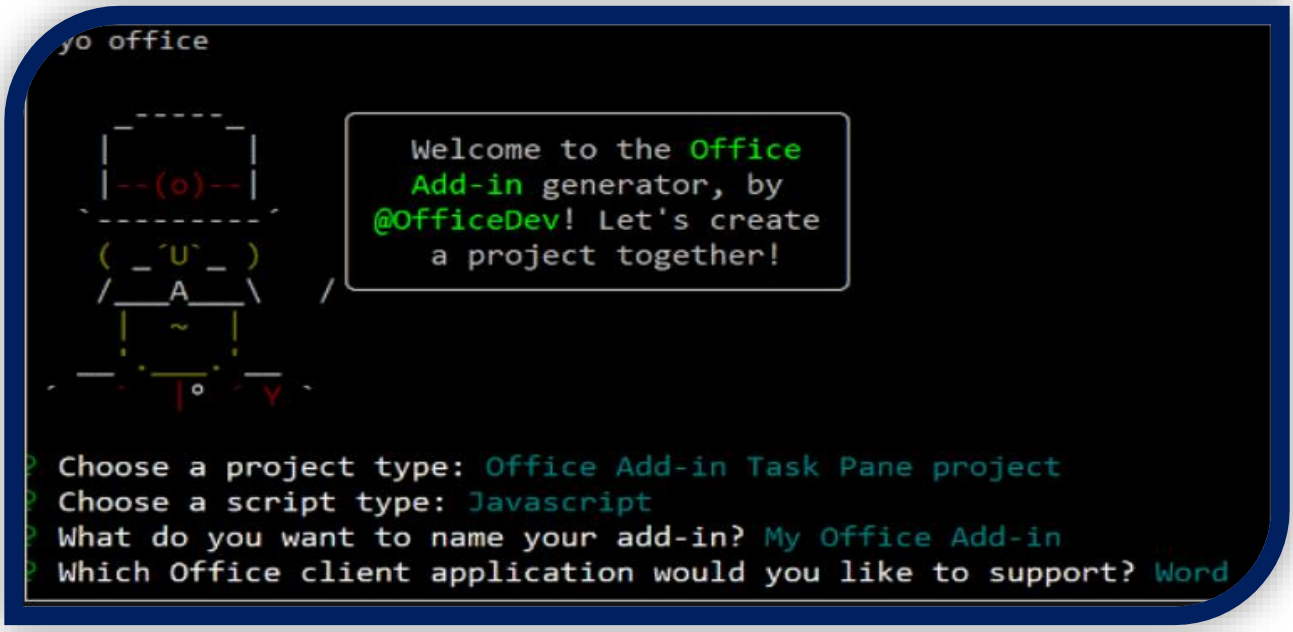
```
yo office
```

Run the following command to create an add-in project using the Yeoman generator:

When prompted, provide the following information to create your add-in project:

1. Choose a project type: Office Add-in Task Pane project
2. Choose a script type: JavaScript
3. What do you want to name your add-in? My Office Add-in

4. Which Office client application would you like to support? Word



After completing the wizard, the generator creates the project and installs supporting Node components.

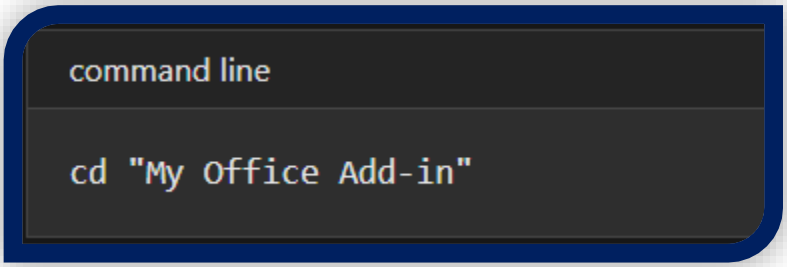
Explore the project

The add-in project that the user has just created with the Yeoman generator contains sample code for a very basic task pane add-in. To explore the components of the add-in project, open the project in the code editor, and review the files listed below.

- The ./manifest.xml file in the root directory of the project defines the settings and capabilities of the add-in.
- The ./src/taskpane/taskpane.html file contains the HTML markup for the task pane.
- The ./src/taskpane/taskpane.css file contains the CSS that's applied to content in the task pane.
- The ./src/taskpane/taskpane.js file contains the Office JavaScript API code that facilitates interaction between the task pane and the Office host application.

Try it out

1. Navigate to the root folder of the project.



2. Complete the following steps to start the local web server and sideload your add-in.

```
command line

npm run dev-server
```

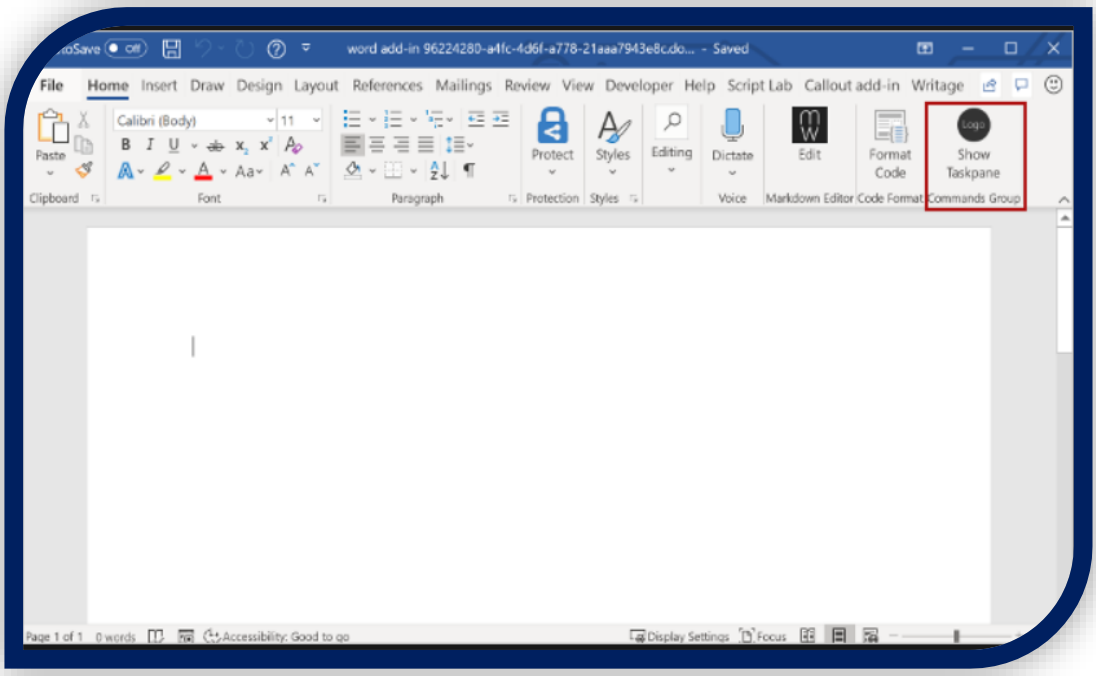
a. To test the add-in in Word, run the following command in the root directory of the project. This starts the local webserver (if it's not already running) and opens Word with the add-in loaded.

```
command line

npm start
```

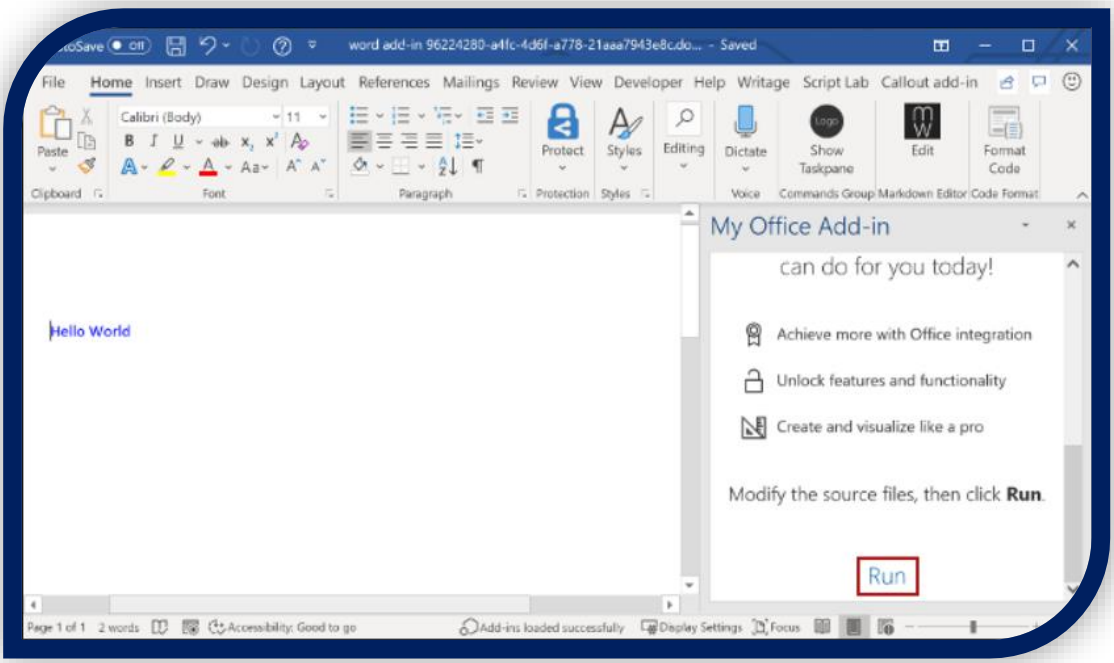
b. To test the add-in in Word on a browser, run the following command in the root directory of the project. When this command runs, the local web server will start (if it's not already running).

```
command line

npm run start:web
```

3. In Word, open a new document, choose the Home tab, and then choose the Show Taskpane button in the ribbon to open the add-in task pane.



4. At the bottom of the task pane, choose the Run link to add the text "Hello World" to the document in blue font.



METHOD 2 - OFFICE 365 WORD WEB ADDIN - VISUAL STUDIO

Prerequisites

a. Visual Studio 2019 with the Office/SharePoint development workload installed
b. Office 2016 or later

Create the add-in project

1. In Visual Studio, choose **Create a new project**.
2. Using the search box, enter add-in. Choose **Word Web Add-in**, then select **Next**.
3. Name your project and select **Create**.
4. Visual Studio creates a solution and its two projects appear in **Solution Explorer**.
5. The Home.html file opens in **Visual Studio.**

Explore the Visual Studio solution

After completing the wizard, Visual Studio creates a solution that contains two projects.

Project	Description
Add-in project	Contains only an XML manifest file, which contains all the settings that describe the add-in. These settings help the Office host determine when the add-in should be activated and where the add-in should appear. Visual Studio generates the contents of this file for you so that the user can run the project and use the add-in immediately. user change these settings any time by modifying the XML file.
Web application project	Contains the content pages of your add-in, including all the files and file references that are required to develop Office-aware HTML and JavaScript pages. While developing the add-in, Visual Studio hosts the web application on the user's local IIS server.

Update the code

1. **Home.html** specifies the HTML that will be rendered in the add-in's task pane. In **Home.html**, replace the <body> element with the following markup and save the file.

```
y>
    <div id="content-header">
        <div class="padding">
            <h1>Welcome</h1>
        </div>
    </div>
    <div id="content-main">
        <div class="padding">
            <p>Choose the buttons below to add boilerplate text to the document by using the Word JavaScript API.</p>
            <br />
            <h3>Try it out</h3>
            <button id="emerson">Add quote from Ralph Waldo Emerson</button>
            <br /><br />
            <button id="checkhov">Add quote from Anton Chekhov</button>
            <br /><br />
            <button id="proverb">Add Chinese proverb</button>
        </div>
    </div>
    <br />
    <div id="supportedVersion"/>
</body>
```

2. Open the file Home.js at the root of the web application project. This file specifies the script for the add-in. Replace the entire contents with the following code and save the file.

```
;
on () {
ffice.onReady(function() {
    $(document).ready(function () {
        if (Office.context.requirements.isSetSupported('WordApi', '1.1')) {
            $('#emerson').click(insertEmersonQuoteAtSelection);
            $('#checkhov').click(insertChekhovQuoteAtTheBeginning);
            $('#proverb').click(insertChineseProverbAtTheEnd);
            $('#supportedVersion').html('This code is using Word 2016 or later.');
        }
        else {$('#supportedVersion').html('This code requires Word 2016 or later.');} });    });
function insertEmersonQuoteAtSelection() {
    Word.run(function (context) {
        var thisDocument = context.document;
        var range = thisDocument.getSelection();
        range.insertText('"Hitch your wagon to a star."\n', Word.InsertLocation.replace);
        return context.sync().then(function () {
            console.log('Added a quote from Ralph Waldo Emerson.');
        });
    })
    .catch(function (error) {
        console.log('Error: ' + JSON.stringify(error));
        if (error instanceof OfficeExtension.Error) {
            console.log('Debug info: ' + JSON.stringify(error.debugInfo));
        }
    });
}
function insertChekhovQuoteAtTheBeginning() {
    Word.run(function (context) {
        var body = context.document.body;
        body.insertText('"Knowledge is of no value unless you put it into practice."\n', Word.InsertLocation.start);
        retu var console: Console (function () {
            console.log('Added a quote from Anton Chekhov.');
        });
    })
     .catch(function (error) {
         console.log('Error: ' + JSON.stringify(error));
         if (error instanceof OfficeExtension.Error) {
             console.log('Debug info: ' + JSON.stringify(error.debugInfo));
         }
     });
}
function insertChineseProverbAtTheEnd() {
    Word.run(function (context) {
        var body = context.document.body;
        body.insertText('"To know the road ahead, ask those coming back."\n', Word.InsertLocation.end);
        return context.sync().then(function () {
            console.log('Added a quote from a Chinese proverb.');
        });
    })
    .catch(function (error) {
        console.log('Error: ' + JSON.stringify(error));
        if (error instanceof OfficeExtension.Error) {
            console.log('Debug info: ' + JSON.stringify(error.debugInfo));
        }
    });
}
})();
```

3. Open the file Home.css at the root of the web application project. This file specifies the custom styles for the add-in. Replace the entire contents with the following code and save the file.

```css
#content-header {
    background: #2a8dd4;
    color: #fff;
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 80px;
    overflow: hidden;
}

#content-main {
    background: #fff;
    position: fixed;
    top: 80px;
    left: 0;
    right: 0;
    bottom: 0;
    overflow: auto;
}

.padding {
    padding: 15px;
}
```

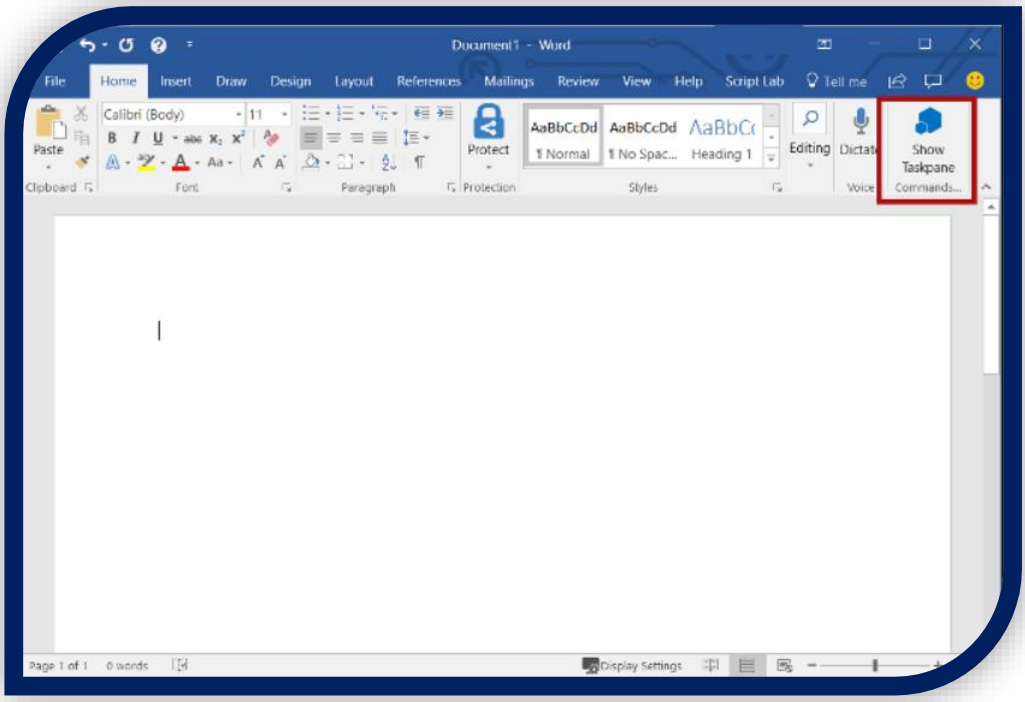
Update the manifest

1. Open the XML manifest file in the add-in project. This file defines the add-in's settings and capabilities.
2. The ProviderName element has a placeholder value. Replace it with your name.
3. The DefaultValue attribute of the DisplayName element has a placeholder. Replace it with My Office Add-in.
4. The DefaultValue attribute of the Description element has a placeholder. Replace it with A task pane add-in for Word.
5. Save the file.

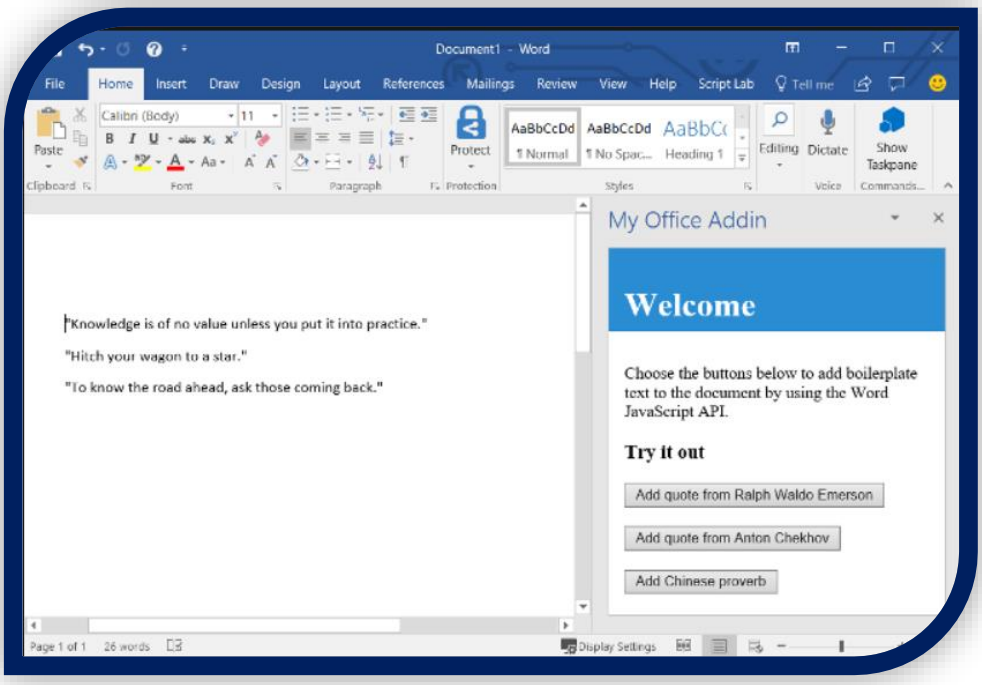
```
<ProviderName>John Doe</ProviderName>
<DefaultLocale>en-US</DefaultLocale>
<!-- The display name of your add-in. Used on the store and various places of the Office UI such as the add-ins dialog. -->
<DisplayName DefaultValue="My Office Add-in" />
<Description DefaultValue="A task pane add-in for Word"/>
```

Try it out

1. Using Visual Studio, test the newly created Word add-in by pressing F5 or choosing the Start button to launch Word with the Show Taskpane add-in button displayed in the ribbon. The add-in will be hosted locally on IIS.
2. In Word, choose the Home tab, and then choose the Show Taskpane button in the ribbon to open the add-in task pane



In the task pane, choose any of the buttons to add boilerplate text to the document.



DEPLOYMENT OF ADDIN

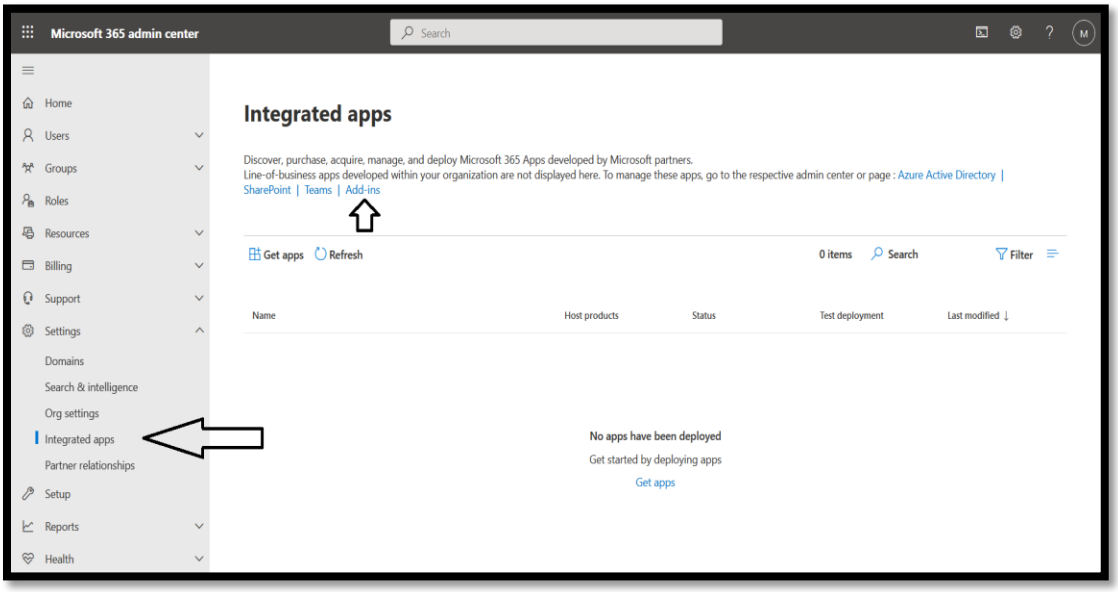
There are multiple methods of deployment but we have followed the Centralized Deployment.

CENTRALIZED DEPLOYMENT

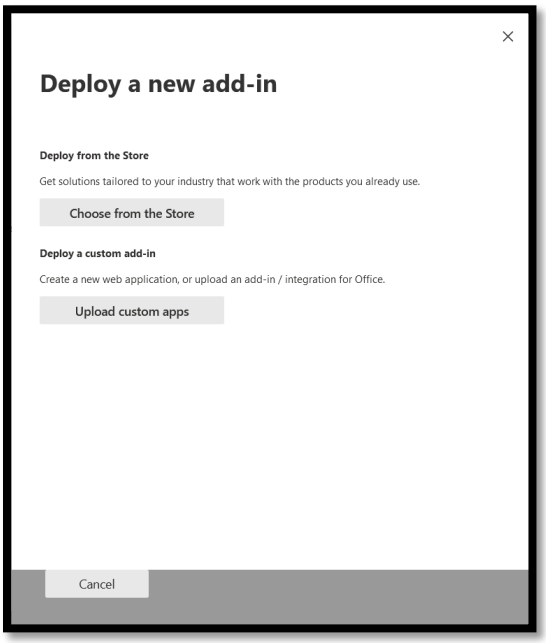
Follow steps below to publish an Office Add-in via Centralized Deployment:

1. Sign in to Microsoft 365 with your work or education account.
2. Select the app launcher icon in the upper-left and choose **Admin**.
3. In the navigation menu, press **Show more**, then choose **Settings** > **Integrated apps**.

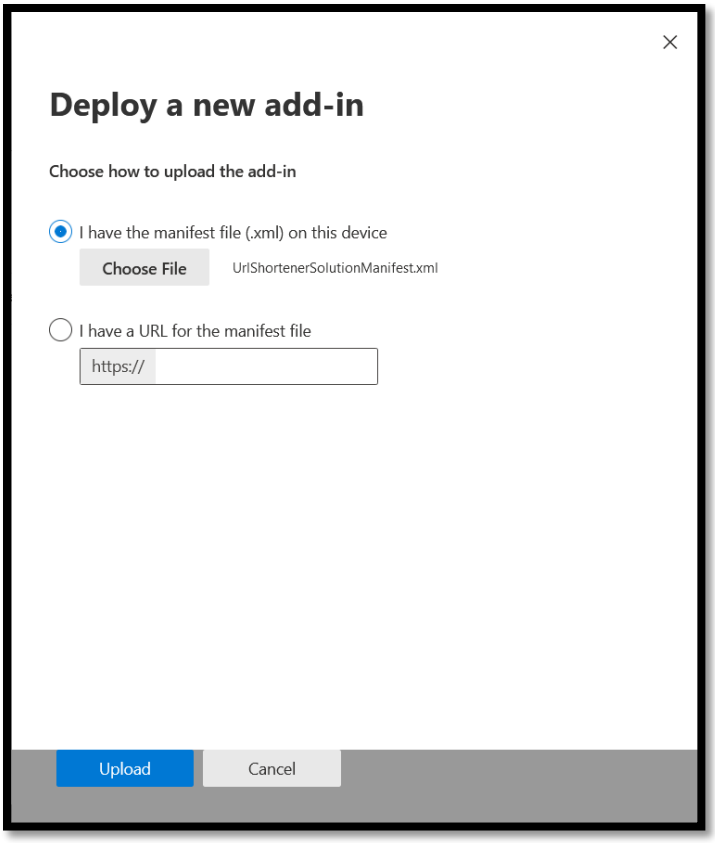
4. Choose **Add-ins** at the top of the page



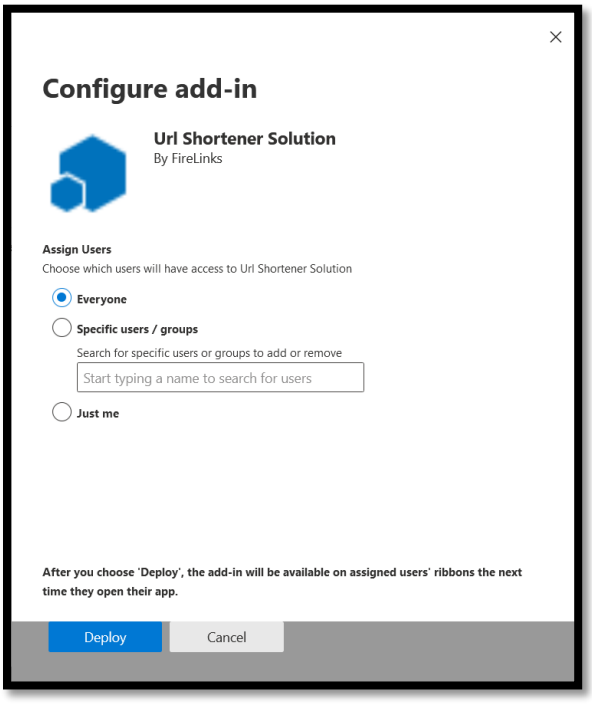
5. Choose **Deploy Add-In** at the top of the page.
6. Choose **Next** after reviewing the requirements.
7. Choose **Upload Custom apps** in the following page



8. Click on **Choose file** and select Add-in manifest file (.xml)
9. Choose **Upload** at the end of the task pane



10. On the **Assign Users** page, choose **Everyone**, **Specific Users/Groups**, or **Only me**. Use the search box to find the users and groups to whom you want to deploy the add-in.



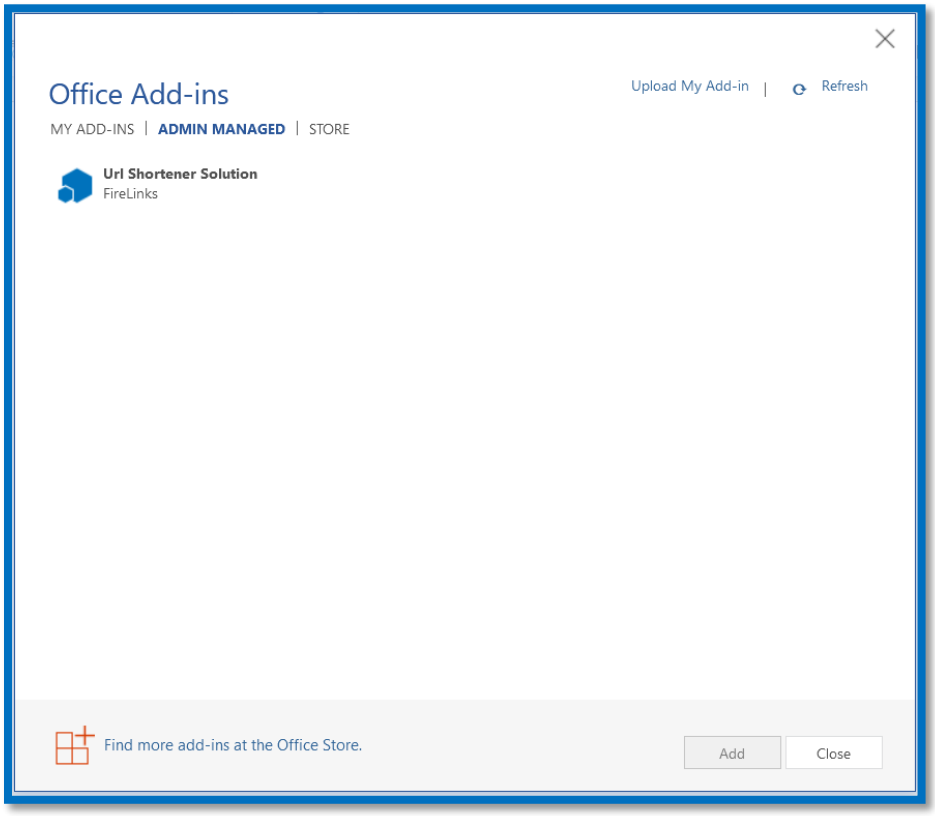
11. When finished, choose **Deploy**. This process may take up to three minutes. Then, finish the walkthrough by pressing **Next**.

DEPLOY ON WEB

Now the add-in has been deployed in your tenant and you can test it on Word web application following steps below:

1) Open a **new blank document** in Word web application
2) Go to **Insert** tab
3) Choose **Add-ins/Office Add-ins** in top navigation
4) Choose **Admin managed** in the top of Office Add-ins modal
5) Select your deployed add-in from the list
6) Choose **Add** at the end of the modal

3



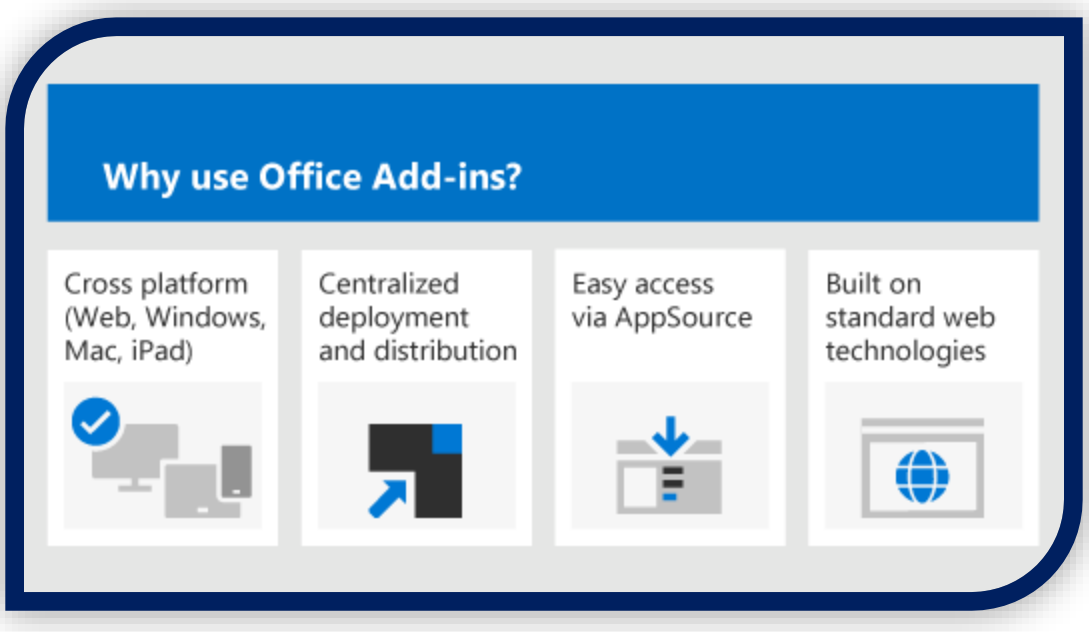
DEPLOY ON DESKTOP

You can test the solution on desktop Word application by following the steps below:

1) Open a **new blank document** in desktop Word application
2) Select **File** > **Account**
3) If you're not already signed in, click **Sign In**, else choose **Switch account**
4) Choose **Sign-in with a different account**
5) Provide **login** credentials of Office account on which add-in has been deployed in Office 365
6) Follow the steps 2 to 6 from heading “Test on Web”

HOW ARE OFFICE ADD-INS DIFFERENT FROM COM AND VSTO ADD-INS?

COM or VSTO add-ins are earlier Office integration solutions that run only on Office on Windows. Unlike COM add-ins, Office Add-ins don't involve code that runs on the user's device or in the Office client. For an Office Add-in, the host application, for example, Excel, reads the add-in manifest and hooks up the add-in's custom ribbon buttons and menu commands in the UI. When needed, it loads the add-in's JavaScript and HTML code, which



executes in the context of a browser in a sandbox.

Office Add-ins provide the following advantages over add-ins built using VBA, COM, or VSTO:

- **Cross-platform support**. Office Add-ins run in Office on the web, Windows, Mac, and iPad.
- **Centralized deployment and distribution**. Admins can deploy Office Add-ins centrally across an organization.
- **Easy access via AppSource**. You can make your solution available to a broad audience by submitting it to AppSource.
- **Based on standard web technology**. You can use any library you like to build Office Add-ins.

EXAMPLES OF WORD WEB ADD-IN IN APP SOURCE

DocuSign for Word

Company: DocuSign, Inc

App Source: URL

Description: Enhance productivity by electronically signing or sending any document from Microsoft Word

Save time by signing or sending documents for eSignature without ever leaving Microsoft Word! With this newest release, you can now access the DocuSign for Word add-in directly from the ribbon – allowing you to request signatures or sign and return critical documents in seconds.

After editing a document, you can use the DocuSign add-in to securely sign a document yourself, or send the document to someone else to complete and sign. Use DocuSign's simple drag and drop functionality to specify tags where recipients need to sign or provide information or add your signature.

Pro Word Cloud

Company: Orpheus Technology Ltd.

App Source: URL

Description: Create word clouds from your text. A great way of visualizing a piece of text or a news feed. The cloud gives greater prominence to words that appear more frequently in the source text. You can tweak your clouds with different sizes, fonts, layouts, and color schemes. The images you create are yours to use as you like.

Add-in capabilities

- When this add-in is used, it
- Can read from your document
- Can send data over the Internet

Bjorn's Word Clouds

Company: Bjorn Backlund

App Source: URL

Description: Create amazing animated word clouds (even as animated GIFs). Supports various image types - packed circles, treemaps, or clouds. For Word, calculates word frequencies from the text. For Excel, loads word frequencies from cell data. For PowerPoint, load word frequencies from imported data (paste, CSV, or TSV). Add-in capabilities

When this add-in is used, it

- Can read and make changes to your document
- Can send data over the Internet

CONCLUSION

In this case study, a brief introduction about the Microsoft Office 365 Word Web add-in, its architecture, development & deployment ways & a demo of the introductory level is discussed in detail.

Our Microsoft Office 365 Consulting, add-in Development, Customization, Integration services, and solutions, can help companies maximize business performance, overcoming market challenges, achieving profitability, and providing the best customer care service.

CONTACT US

Shahzad Sarwar

Cognitive Convergence Team