

**By Daniela Balaniuc on July/2021**

**Web Chat with Watson assistant.**

**Did you know that most companies spend millions in monthly conversations to help consumers been routed in the right path?**

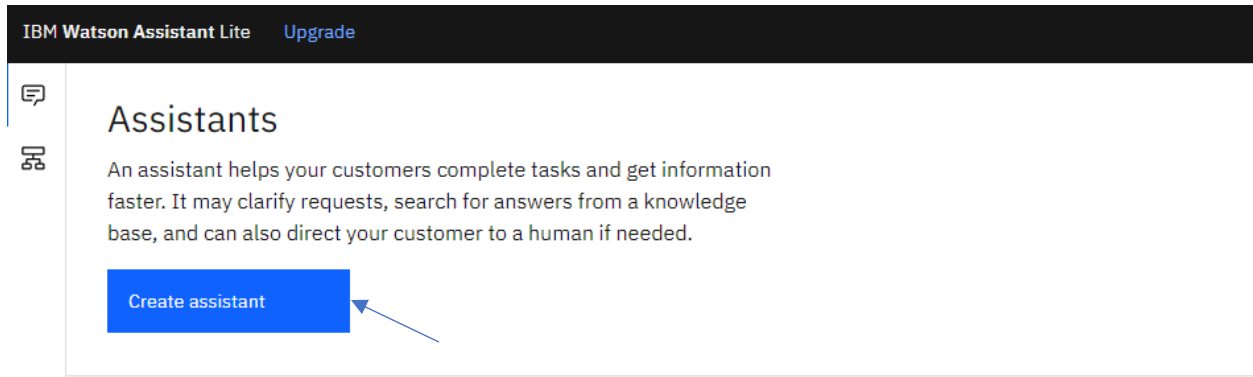
In an economy that the average of most companies is spending millions in monthly conversations to help consumers been routed in the right path. It is very important to get in hand the best way to have an economy more favorable to the companies when in treating the consumer. So, if you want to have the right information at an easy access for the clients' questions, or if you are knowing in programming or don't know anything yet about programming, and want to learn how to get in the programming business, this is the one of the best ways to start to learn. I will show you how to path the IBM Watson Assistant, which is one of the most popular virtual system for information for website, easy and without a cost.

Therefore, the conversational intelligence (AI) is increasingly mainstream capability with which consumers interact daily in their homes, workplaces, and on the go. The Watson Assistant is IBM's chatbot that allows users to interact with business systems using natural language. The purpose of this article is to inform you about how to get the Watson Assistant, and the potential financial impact of Watson Assistant in organizations. The Watson Assistant can be used to serve customer self-service, employee self-service, and agent assist. Without the Watson Assistant these services are provided by human-serviced chat, email, and call services. These methods are slow, and costly. There are a lot of companies out in the market that are in needed for help, it is true that these companies spent millions in monthly conversations to help consumers been routed in the right path, I've myself have been working in customer service helping consumers been routed in the right path. With Watson Assistant you will be able to use the assistant for free in your website's company, so first step to take is to go to the IBM Cloud then open a Lite account, once you signed up for a free IBM Cloud account you click create, then you will click Launch Watson Assistant an assistant, and then create a dialog skill.

As you can see, the Lite will give you the option to work with certain time for free, and of course after that certain time you will have to start make some payments, but believe me if you are not using for a million times, the time available on the Lite will never been removed from your account, but of course only when is expired you will have some time out, and then, after that certain time, you can return to your work for free, but you will have enough time to do your work with small and medium companies, and let me explain better, I am talking about the time that is for your work, not for the use on your website, so if you are fast enough, the Lite time will be enough, thus, is why I am here doing the best I can to work, and to help you to save time by explaining the tutorial and prompts in a fast way, so you can work in fast pace which is important to save time.

Now that you have created your Watson Assistant Lite account, you can take your first step which is to Launch Watson Assistant, you can see the blue icon to click on.

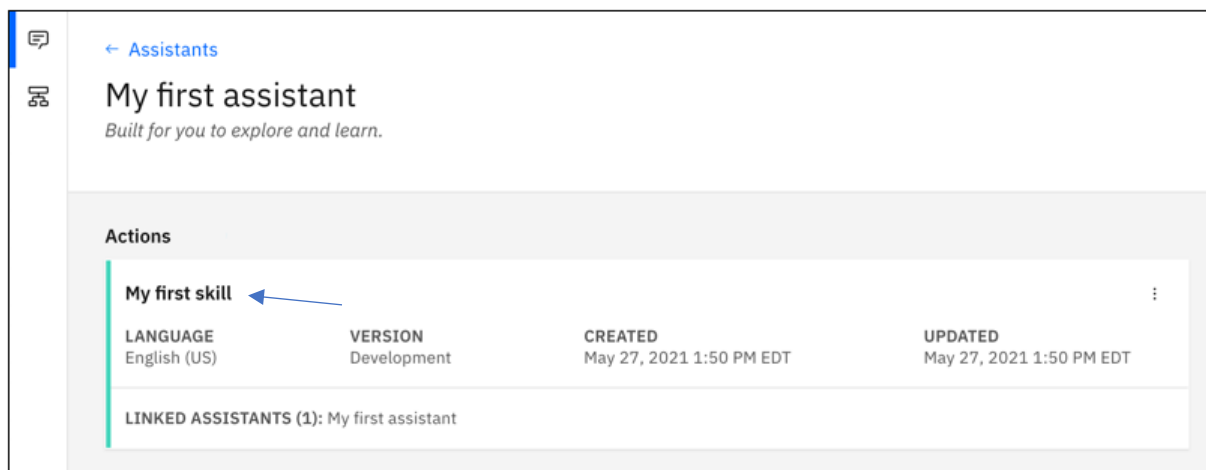
Then click on the blue icon Create assistant, which you will be able to see My first assistant for actions and dialog skills. Skills are the brain of the Watson assistant, where you will be able to create a dialog using natural language.



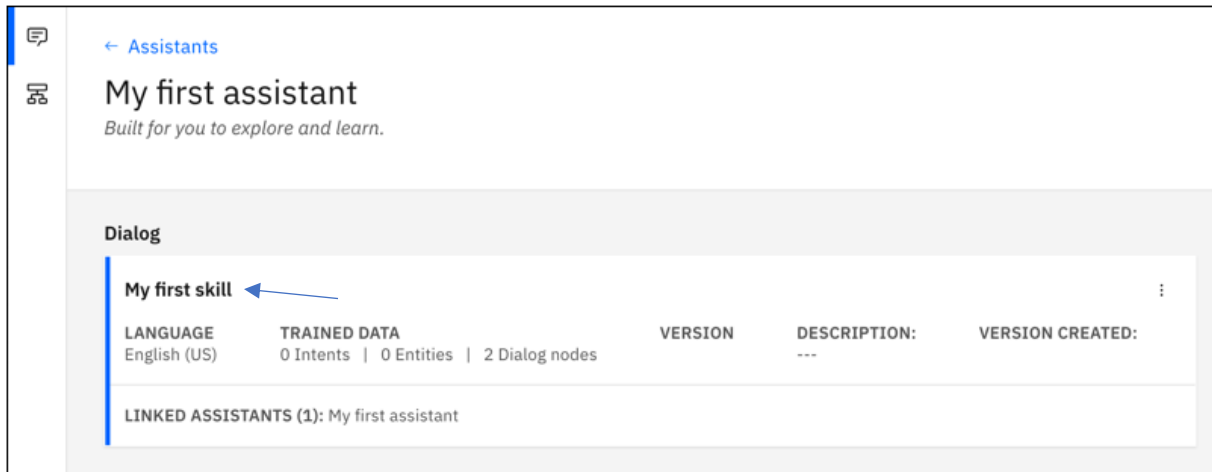
To get started, the highest-level concept Watson assistant is the assistant layer. The assistants affect the face of the entire virtual assistant. This is the piece that is integrated into message channels like slack or Facebook, so you can put on your web page with a host and web chat.

As you can see now, the skills have actions, and dialog, also you can create on the integrations and webchat, which is where I want to show you the codes so you can copy to add to your website.

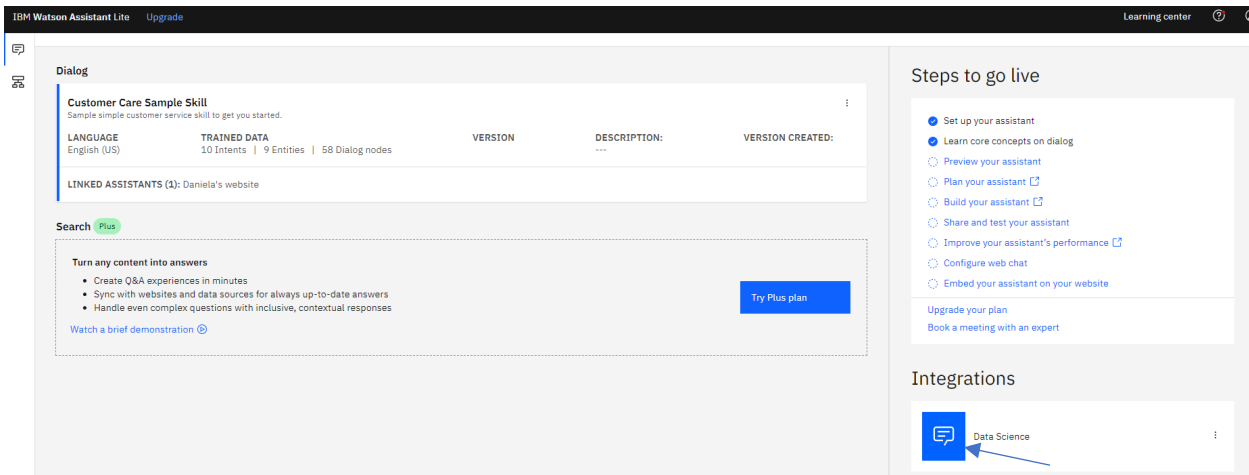
Now you can click on the icon. My first assistant to create an Action, so by creating an action it means that you will be building your first conversation, where you can use the skills that contain actions that represent the tasks you want your assistant to help your customers with. Thus, each action contains a series of steps that represent individual exchanges with a customer. In each action step, you define the user goal that the action will satisfy. So, to recognize when a customer is just saying hello, you can add some examples of ways in which a person might do so.



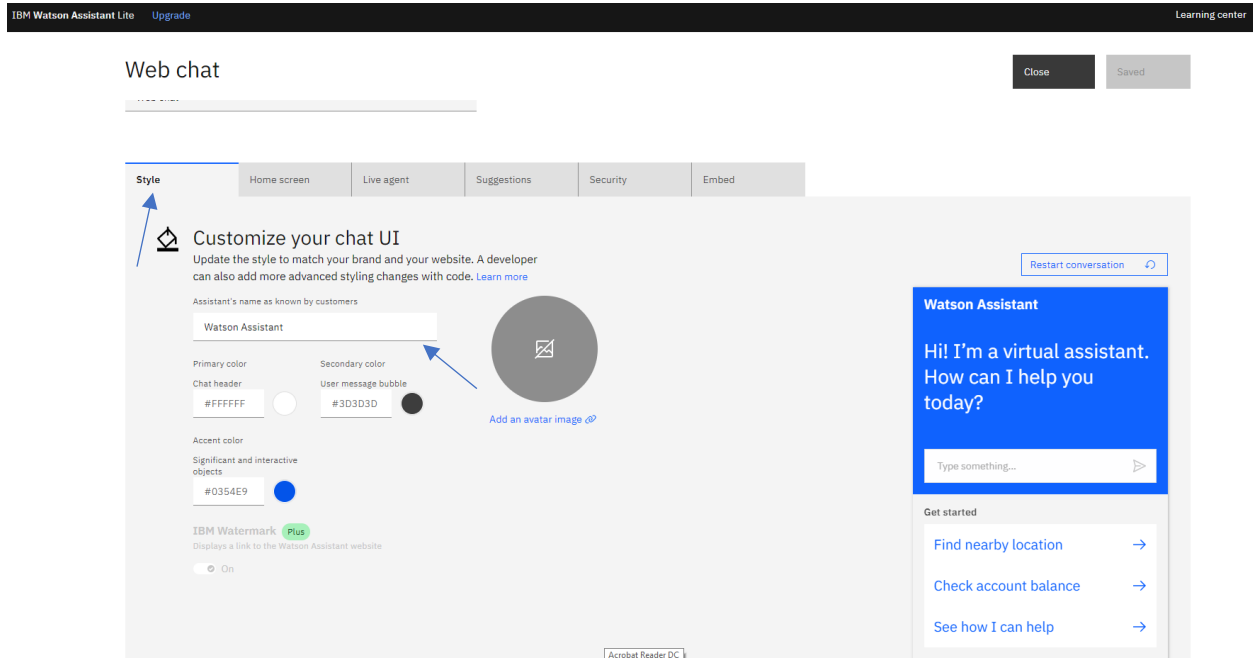
Now you can click on the icon My first assistant to create a dialog for an assistant that helps users with inquiries about your business. You will be able to plan a dialog, define custom intents, add dialog nodes that can handle your intents, add entities to make your responses more specific, add pattern entity, and use it in the dialog to find patterns in user input, and set reference context variables. These objectives tutorial can take 2 to 3 hours to complete, and of course it will be worth it, but as I mention before I am here to show you how to save time by using the web chat which will be faster by copying the codes that are ready for you for add to your webchat.



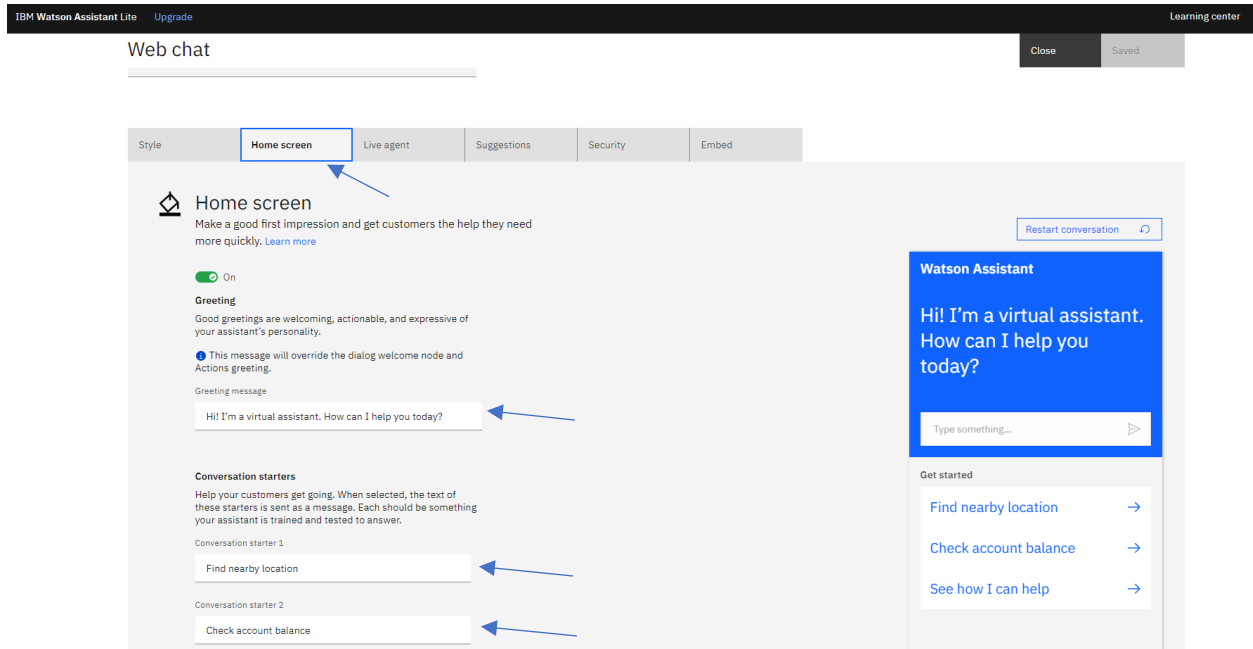
As you can see on the dialog skill, on the right side there is an icon called integrations for the web chat, so first on the three dots create your web chat integration, then click on the blue icon that you have created, so now you can see your Style, Home Screen, Live Agent, Suggestions, Security, and Embed.



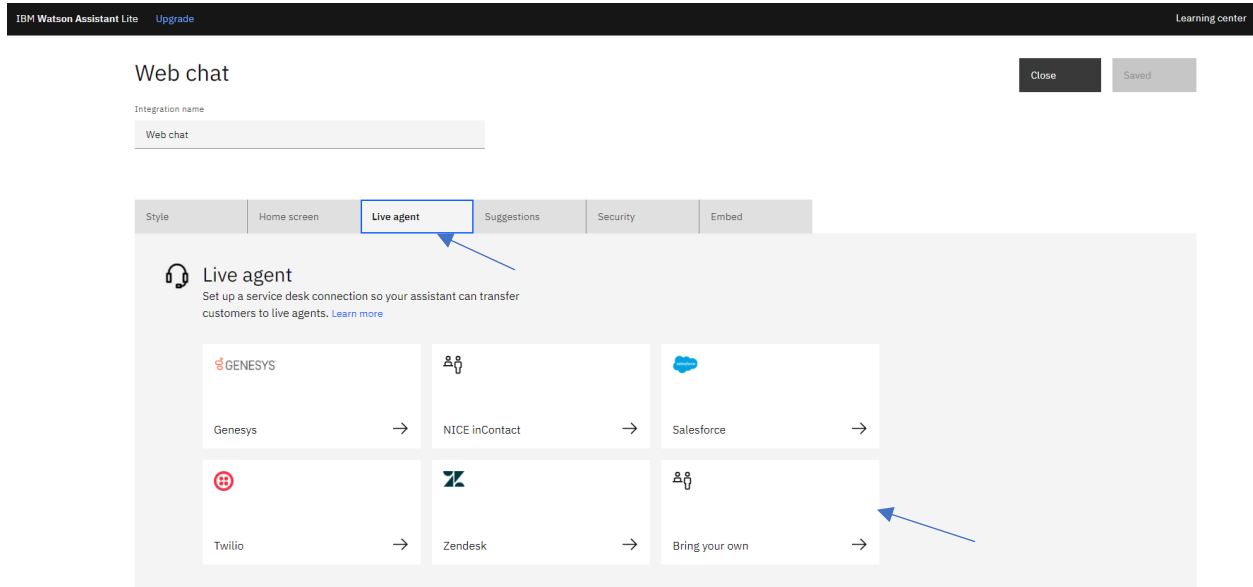
As you can see on the web chat there are several icons called the Style, Home Screen, Live Agent, Suggestions, Security, and Embed. In here you can give a name for your customized chat, then you can move to Home Screen.



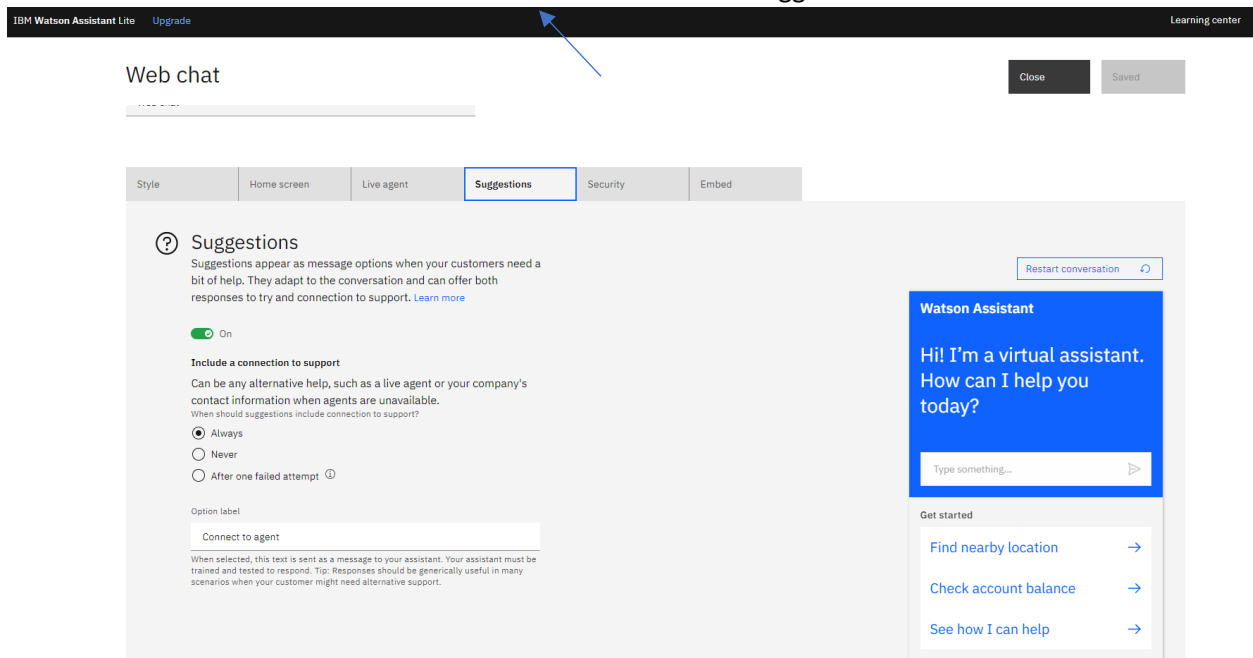
In the Home Screen you will be able to customize your Watson Assistant by introducing you assistant and also writing some options of actions where you think your customer will be looking for information about your product and company.



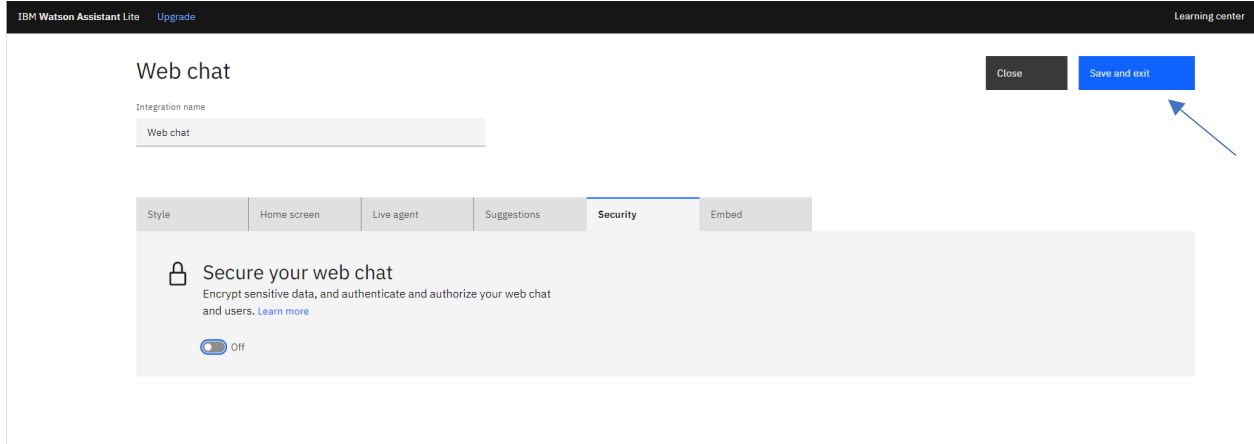
In the Live Agent you have several channels to choose, I personally have used the bring your own where you can try the git hub where you can create your codes to use, which is what I will be using also with the web chat.



In the Suggestions you can give your customers a way to try something else when the current exchange with the assistant isn't delivering what they expect. A question mark icon ? is displayed in the web chat that customers can click at any time to see other topics that might be of interest or, if configured, to request support. Also, customers can click a suggested topic to submit it as input or click the X icon to close the suggestions list.



In the Security all messages that are sent from the web chat re encrypted. When you enable security, your assistant takes an additional step to verify that messages originate from the web chat is embedded in your website only. Also, you can implement the following security measures; ensure that messages sent from the web chat to your assistant come from your customers only, send private data from the web chat to your assistant.



In the Embed is when you create a web chat integration, the Watson Assistant UI gives you a small embed code to add to your website, as in this example. Including are the web chat configuration options that are defined by the parameters for `watsonAssistantChatOptions`. By default, the embed code provided by Watson Assistant configures only the `integrationID`, `serviceInstanceID`, and the region parameters (premium also include `subscriptionID`). These options also include a `onLoad` callback that will be called when the web chat code has been loaded and a new instance has been created. You will need to call `render` on that instance to actually display widget. You can modify `watsonAssistantChatOptions` with additional parameters that control immutable behavior of your web chat instance. But no worries, you can use this code for free. Now by knowing this you can copy and paste this embed code in your website, and by clicking on the blue icon `Learn more`, you will get inside the Watson Assistant web chat configuration, where you will see this script and the configuration options object, and you will be able to navigate on the icons on the left side, where you can see the tutorials for `Getting started`, `Setting context`, `Custom Launcher`, `Custom element`, `Custom response types` `Custom response with React` and `Session History` for you to get more code to add to your website.

IBM Watson Assistant Lite Upgrade Learning center

## Web chat

Integration name  
Web chat

Close Save and exit

Style Home screen Live agent Suggestions Security **Embed**

`</>` Embed on your website  
Ready to launch? It's as easy as copy and paste. [Learn more](#)

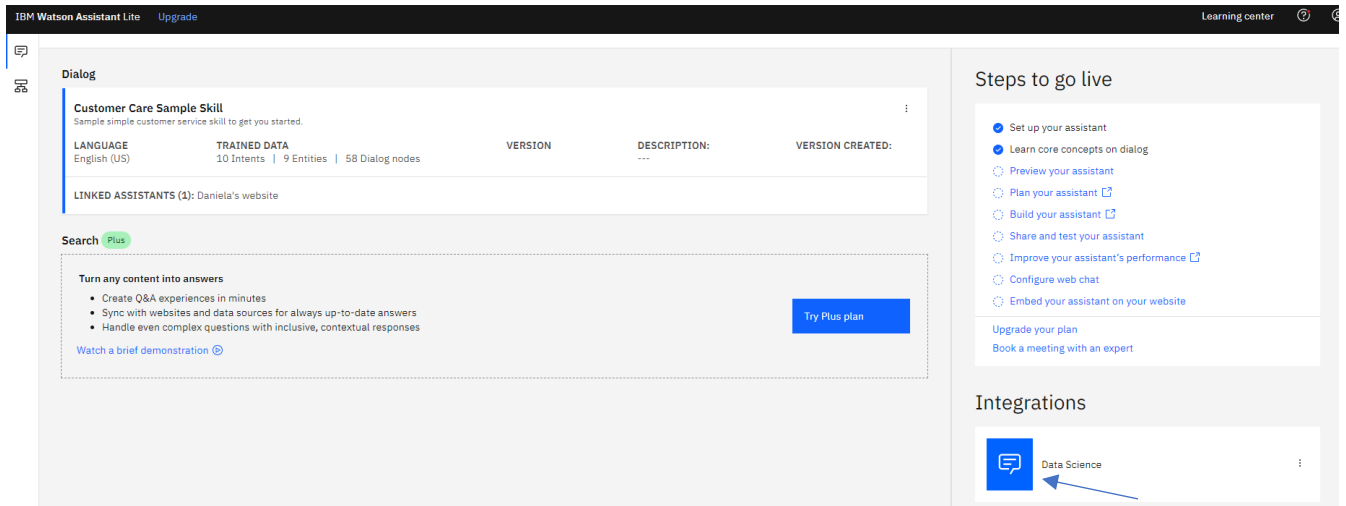
```
<script>
window.watsonAssistantChatOptions = {
  integrationID: "0c2d86f0-e6f8-483e-9252-609e8a3ef168", // The ID of this integration.
  region: "us-south", // The region your integration is hosted in.
  serviceInstanceID: "f3216de5-49c7-45ef-847b-02a26242302c", // The ID of your service instance
  onLoad: function(instance) { instance.render(); }
};
setTimeout(function(){
  const t=document.createElement('script');
  t.src="https://web-chat.global.assistant.watson.appdomain.cloud/loadWatsonAssistantChat.js";
  document.head.appendChild(t);
});
</script>
```



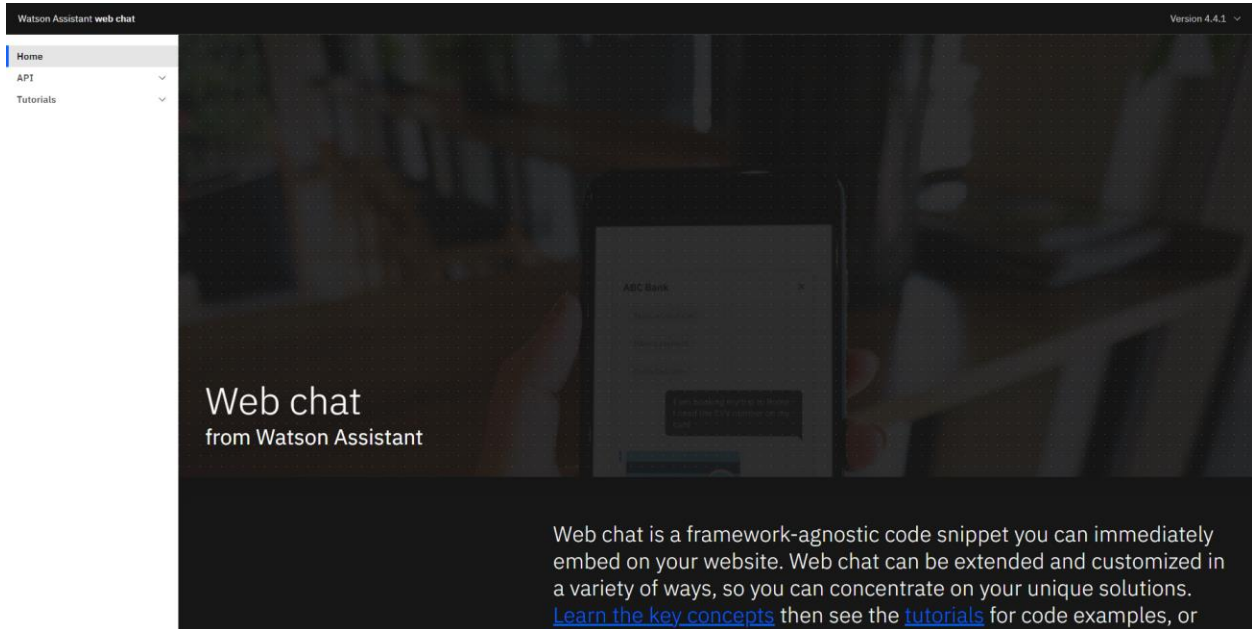
Now let's get into the fast-paced web chat by following the prompt to the ready code. If you have never code with HTML before, you don't have to worry, these are ready codes, all you have to do is copy and paste in your website, but if you are interested in starting coding, here is a good tip it's called git hub which is free you can open an account, to start your coding projects. As you can see on the Live agent you can bring your own agent by creating an account with git hub for free and also to create your webpage. By adding service desk support, the Web chat integration has limits, the usage is measured differently depending on the type of the plan type. For Lite plans, usage is measured by the numbers of /message calls (API) are sent to the assistant from the web chat integration. Lite plan has 10,000 API (approximately 1,000 MAU).

The screenshot shows the IBM Watson Assistant configuration interface for Web chat integration. At the top, there is a navigation bar with "IBM Watson Assistant Lite Upgrade" on the left and "Learning center" on the right. Below this, the "Web chat" configuration page is displayed. The "Integration name" field is set to "Web chat". A "Close" button is on the left and a "Save and exit" button is on the right. A horizontal menu below the integration name includes "Style", "Home screen", "Live agent", "Suggestions", "Security", and "Embed". The "Live agent" tab is selected and highlighted with a blue arrow. The "Live agent" section contains the following text: "Live agent", "Set up a service desk connection so your assistant can transfer customers to live agents. [Learn more](#)", "Bring your own", "Integrate your own preferred service desk by using the starter kit available on GitHub.", and "Change service desk" with a link icon. Below this is a white box with a circular refresh icon, the text "Get started on GitHub", and a right-pointing arrow.

In the Integrations you add the web chat to your website, add your assistant to your company website as a web chat widget that can help your customers with common questions and tasks, and can transfer customers to human agents. When you create a web chat integration, code is generated that calls a script that is written in JavaScript. The script instantiates a unique instance of your assistant. You can then copy and paste the HTML script element into any page or pages n your website where you want users to be able to ask assistant to help.



By navigating on the icons on the left side, you can see the tutorials, click on the tutorials you will see Getting started, Setting context, Custom Launcher, Custom element, Custom response types Custom response with React and Session History for you to get more code to add to your website.



In Getting Started you can see the same first script you saw on embed at the skills, and if you click at the blue icon, it will show you the prompted Watson Assistant.

Watson Assistant web chat Version 4.4.1

Home  
API  
Tutorials  
**Getting started**  
Setting context  
Theming  
Custom launcher  
Custom element  
Custom response types  
Custom responses with React  
Session history

## Getting started

Learn how to quickly embed a web chat on your website and what happens when you do.

### Overview

With just a few lines of code that are provided for you, you can add a working web chat instance to your website.

See it in action

Add an instance of web chat configured to behave like this tutorial to this page.

[Add web chat to see the final result](#)

### Instructions

When you create a web chat integration, the Watson Assistant UI gives you a small embed code to add to your website, as in this example:

Example

```
<doctype html>
<html lang="en">
<body>
<script>
window.watsonAssistantChatOptions = {
  // A GUID like '6438544b-b3e1-4f70-8eff-7149e43e938b' included in the embed code provided in Watson Assistant.
  integrationID: "YOUR_INTEGRATION_ID",
  // Your assistants region e.g. 'us-south', 'us-east', 'jp-tok', 'au-syd', 'eu-gb', 'eu-de', etc.
  region: "YOUR_REGION",
  // A GUID like '6438544b-b3e1-4f70-8eff-7149e43e938b' included in the embed code provided in Watson Assistant.
  serviceInstanceID: "YOUR_SERVICE_INSTANCE",
  // The callback function that is called after the widget instance has been created.
  onLoad: function(instance) {
    instance.render();
  }
};

setTimeout(function(){const t=document.createElement('script');t.src='https://web-chat.global.assistant.watson.appdomain.cloud/loadWatsonAssistantChat.js';document.head.appendChild(t);},0);
</script>
</body>
</html>
```

In the Setting context you can find a basic embed code, and the code below is the pre: send event and updating context. This tutorial uses an example dialog skill that uses a context variable to determine its responses. This skill looks for the boolean context party time; when you ask “Is it party time?“, the skill responds in the negative or affirmative depending on the value of the context variable.

Watson Assistant web chat Version 4.4.1

Home  
API  
Tutorials  
Getting started  
**Setting context**  
Theming  
Custom launcher  
Custom element  
Custom response types  
Custom responses with React  
Session history

## Setting context

### Subscribing to the pre: send event and updating context

```
<doctype html>
<html lang="en">
<body>
<script>
/*
 * Following the v2 message API Response
 * at https://cloud.ibm.com/apidocs/assistant/assistant-v2/send-user-input-to-assistant,
 * we add some items to context.
 */
function preSendHandler(event) {
  // When fetching the Welcome Node on startup, the context won't be defined, yet. If you want to add to
  // context when fetching welcome node, you will need to define the context.
  event.data.context.skills[main skill] = event.data.context.skills[main skill] || { user_defined: {} };
  event.data.context.skills[main skill].user_defined.party_time = true;

  // You can OPTIONALLY return a promise and we will wait to continue processing until the promise is resolved. If
  // you return nothing we will immediately continue processing the event. If you fail the Promise we will cancel
  // sending the message.
  //return new Promise(function(resolve) {
  //  myAsyncThing.then(function(moreData) {
  //    resolve();
  //  });
  //});

  // Do some other manipulation of event.data...
}

window.watsonAssistantChatOptions = {
  integrationID: "YOUR_INTEGRATION_ID",
  region: "YOUR_REGION",
  serviceInstanceID: "YOUR_SERVICE_INSTANCE_ID",
  onLoad: function(instance) {
    // Subscribe to the "pre:send" event.
    instance.on({ type: "pre:send", handler: preSendHandler });
  }
};

setTimeout(function(){const t=document.createElement('script');t.src='https://web-chat.global.assistant.watson.appdomain.cloud/loadWatsonAssistantChat.js';document.head.appendChild(t);},0);
</script>
</body>
</html>
```

In the Theming you can use Carbon themes to quickly change the whole look of your we chat. First create an instance of web chat to get the basic embed code.

The screenshot shows the 'Theming' section of the Watson Assistant web chat configuration interface. The left sidebar has 'Theming' selected. The main content area is titled 'Updating the configuration to switch Carbon themes' and contains a code block for the basic embed code. Below this, there is a section 'Customizing the web chat styling' with a paragraph explaining the use of `updateCSSVariables()` and a link to the 'Carbon Design System' documentation. Another section 'Updating CSS variables' shows a code block with specific CSS variables being updated.

```
<!doctype html>
<html lang="en">
  <body>
    <script>
      window.watsonAssistantChatOptions = {
        integrationID: "YOUR_INTEGRATION_ID",
        region: "YOUR_REGION",
        serviceInstanceID: "YOUR_SERVICE_INSTANCE",
        // Config option to change Carbon themes.
        carbonTheme: 'g90',
        onload: function(instance) {
          instance.render();
        }
      };
      setTimeout(function(){const t=document.createElement('script');t.src='https://web-chat.global.assistant.watson.appdomain.cloud/loadWatsonAssistantChat.js';document.head.appendChild(t)});
    </script>
  </body>
</html>
```

Updating the configuration to switch Carbon themes

Customizing the web chat styling

You can customize the selected theme by using the `updateCSSVariables()` [instance method](#). For a list of supported theme variables and their default values, see the [Carbon Design System](#) documentation.

Updating CSS variables

```
<!doctype html>
<html lang="en">
  <body>
    <script>
      window.watsonAssistantChatOptions = {
        integrationID: "YOUR_INTEGRATION_ID",
        region: "YOUR_REGION",
        serviceInstanceID: "YOUR_SERVICE_INSTANCE_ID",
        carbonTheme: 'g90',
        onload: function(instance) {
          // Instance method to update specific CSS variables
          instance.updateCSSVariables({
            'BASE-font-family': '-times New Roman, Times, serif',
            '$active-primary': '#009900',
            '$focus': '#008000',
            '$hover-primary': '#00F000',
            '$interactive-B1': '#008000'
          });
          instance.render();
        }
      };
    </script>
  </body>
  <script>
    setTimeout(function(){const t=document.createElement('script');t.src='https://web-chat.global.assistant.watson.appdomain.cloud/loadWatsonAssistantChat.js';document.head.appendChild(t)});
  </script>
</html>
```

In the Custom Launcher use your own method of opening the web chat on your page. First create an instance of web chat to get the basic embed code.

The screenshot shows the 'Custom launcher' section of the Watson Assistant web chat configuration interface. The left sidebar has 'Custom launcher' selected. The main content area is titled 'Showing the custom launcher' and contains a code block with CSS keyframes and styles for a custom chat launcher button. The code includes a keyframe animation for sliding in from the right and styles for the button in its default, open, hover, and focus states.

```
<!doctype html>
<html lang="en">
<head>
  <style>
    @keyframes slideInRight {
      from {
        transform: translate3d(100%, 0, 0);
        visibility: visible;
      }
      to {
        transform: translate3d(0, 0, 0);
      }
    }
    button.chatLaunches {
      animation-duration: 0.5s;
      transition-duration: 0.5s;
      position: fixed;
      bottom: 32px;
      right: 32px;
      z-index: 9999;
      border: 4px solid #075cc2;
      padding: 1em;
      border-radius: 8px;
      margin: 0;
      text-decoration: none;
      background-color: #ffffff;
      color: #4d4d4d;
      font-family: sans-serif;
      font-size: 1em;
      cursor: pointer;
      text-align: left;
      webkit-appearance: none;
      -moz-appearance: none;
      width: 284px;
      opacity: 0;
    }
    button.chatLauncher.open {
      animation-name: slideInRight;
      opacity: 1;
    }
    button.chatLauncher:hover,
    button.chatLauncher:focus {
      background-color: rgb(225, 225, 254);
      border: 4px solid #0053ba;
    }
    button.chatLauncher:focus {
      outline: 1px solid #0053ba;
      outline-offset: -4px;
    }
  </style>
</head>
<body>
  <!-- We want to hide this element initially, because web chat isn't ready yet. -->
  <button type="button" class="chatlauncher" style="display:none;">

```

Showing the custom launcher

To integrate with other methods, contact, or to better match your brand, you might want to replace the default web chat launcher icon with your own mechanism for opening the web chat window.

```

-webkit-appearance: none;
moz-appearance: none;
width: 264px;
opacity: 0;
}

button.chatlauncher.open {
  animation-name: slideInRight;
  opacity: 1;
}

button.chatlauncher:hover,
button.chatlauncher:focus {
  background-color: rgb(225, 225, 254);
  border: 4px solid #0053ba;
}

button.chatlauncher:focus {
  outline: 1px solid #0053ba;
  outline-offset: -4px;
}
</style>
</head>
</body>
<!-- We want to hide this element initially, because web chat isn't ready yet. -->
<button type="button" class="chatlauncher" style="display:none;">
  <div>
    <strong>Have questions?</strong> Talk with Karen our Virtual Assistant.
  </div>
</button>
</script>
window.watsonAssistantChatOptions = {
  integrationID: "YOUR_INTEGRATION_ID",
  region: "YOUR_REGION",
  serviceInstanceID: "YOUR_SERVICE_INSTANCE_ID",

  showLauncher: false,
  onLoad: function(instance) {
    // Select the button element from the page.
    const button = document.querySelector('.chatlauncher');

    // Add the event listener to open your web chat.
    button.addEventListener('click', function clickListener() {
      instance.openWindow();
    });

    // Render the web chat. Nothing appears on the page, because the launcher is
    // hidden and the web chat window is closed by default.
    instance.render().then(function() {
      // Now that web chat has been rendered (but is still closed), we make the
      // custom launcher button visible.
      button.style.display = 'block';
      button.classList.add('open');
    });
  }
};

setTimeout(function(){const t=document.createElement('script');t.src='https://web-chat.global.assistant.watson.appdomain.cloud/loadWatsonAssie
Have questions? Talk with Karen our Virtual Assistant.

```

In the Custom element is to render the web chat widget within your own custom element on the page. Depending on your website design, you might want not want to render the web chat widget in the default DOM element on the page. For example, you might want to render the web chat in a different location, or at a different size; you might want to show the web chat already open, or embed it more deeply in your content.

Configuring a custom element

To change the DOM element where the web chat widget is rendered, we need to use the `element` configuration option. (For more information about the web chat configuration options, see [Configuration](#).)

If you want to render the web chat widget inside a custom element, you should not use the default launcher. Instead, you can use a [custom launcher](#); alternatively, you can choose not to use a launcher at all, and instead just show the web chat always open by using the `openChatByDefault` configuration option. (If you use this approach, you should probably also set `hideCloseButton` to true to prevent users from closing the web chat window.)

**Note:** The web chat widget grows to fill the size of the element you provide. If you do not specify the height and width of the element, the web chat has no fixed size.

The following code snippet updates the configuration to render the web chat inside a custom DOM element. It also renders the web chat widget open by default, and it hides the close button.

Rendering to a custom element

```

<!doctype html>
<html lang="en">
<head>
<style>
  .chatElement {
    height: 600px;
    width: 100%;
  }
</style>
</head>
<body>
  <div class="chatElement"></div>
</body>
</html>
<script>
const element = document.querySelector('.chatElement');
window.watsonAssistantChatOptions = {
  integrationID: "YOUR_INTEGRATION_ID",
  region: "YOUR_REGION",
  serviceInstanceID: "YOUR_SERVICE_INSTANCE",

  // Provide the custom element.
  element: element,
  // Hide the close button since we want it always open.
  hideCloseButton: true,
  // Hide the default launcher.
  showLauncher: false,
  // Make the window open by default.
  openChatByDefault: true,

  onLoad: function(instance) {
    instance.render();
  }
};

setTimeout(function(){const t=document.createElement('script');t.src='https://web-chat.global.assistant.watson.appdomain.cloud/loadWatsonAssistantChat.js';document.head.append

```

In the Custom response types is to use user-defined templates to render your own custom response types in your web chat. In this tutorial you will see an example of dialog skill that responds with a user defined response type, a basic embed code, a handling the customResponse event, a writing your user defined response handler, and a Styling your user defined response handler.

Watson Assistant web chat Version 4.4.1

Home

API

Tutorials

- Getting started
- Setting context
- Theming
- Custom launcher
- Custom element
- Custom response types**
- Custom responses with React
- Session history

Styling your user\_defined response handler

By default, buttons, tables, lists, and text have styling attached to them that matches other web chat content. Web chat also provides some helper classes you can use for other nonstandard content. In this case, we need to wrap our content in a card like the ones that web chat uses for some built-in response types. (For more information about the helper classes, [see the documentation](#).)

```

/**
 * Handler for the color picker template. Writes a <div> to the element provided by web chat that allows the user to change
 * the color of the div text by typing colors into the input field in the div.
 *
 * @param event The event passed from Watson Assistant.
 * @param event.type The type of event, in this case "customResponse".
 * @param event.data.message The original message.
 * @param event.data.message.response.type "user_defined"
 * @param event.data.message.user_defined A free object for you to pass whatever you want into.
 * @param event.data.element An HTML element that is rendered in web chat for you to add your content to.
 */
function handleColorPickerTemplate(event) {
  // Create a text element
  const textElement = document.createElement('p');
  textElement.innerText = 'Type in a color below and hit enter to change the color of this text.';
  textElement.setAttribute('style', 'margin-top: 0;');

  // Create an input element
  const inputElement = document.createElement('input');
  inputElement.setAttribute('type', 'text');
  inputElement.setAttribute('style', 'background-color: #f4f4f4;');
  inputElement.setAttribute('value', event.data.message.user_defined.default_color);
  textElement.style.color = inputElement.value;

  // Add an event listener to the input element
  inputElement.addEventListener('change', function inputChange() {
    textElement.style.color = inputElement.value;
  });

  // Create a card with the 'ibm-web-chat--card' class we will add our content to.
  // This class makes the element look like one of the cards used in web chat.
  const card = document.createElement('div');
  card.classList.add('ibm-web-chat--card');

  // Append the text and input elements to the card
  card.appendChild(textElement);
  card.appendChild(inputElement);

  // Create a div element to hold the card
  const divElement = document.createElement('div');
  divElement.appendChild(card);

  // Append your element to the provided element.
  event.data.element.appendChild(divElement);
}

```

In the Custom response types with React portals, you can use React portals to render your custom response types as part of your application.

Watson Assistant web chat Version 4.4.1

Home

API

Tutorials

- Getting started
- Setting context
- Theming
- Custom launcher
- Custom element
- Custom response types
- Custom responses with React**
- Session history

Custom responses with React

This is a react component you can insert anywhere in your application that will register a listen for custom response events and will render custom response components in React portals attach to those custom response host elements provided by the chat widget. You should only render this component once the chat widget has been rendered and the "instance" is available.

```

/**
 * This is a react component you can insert anywhere in your application that will register a listen for custom response
 * events and will render custom response components in React portals attach to those custom response host elements
 * provided by the chat widget.
 *
 * You should only render this component once the chat widget has been rendered and the "instance" is available.
 */
class UsingWithReactClass extends React.Component {
  constructor(props) {
    super(props);
    // This component state will be used to record all of the custom response events that are fired from the widget.
    // These events contain the HTML elements that we will attach our portals to as well as the messages that we
    // wish to render in the message.
    this.state = { customResponseEvents: [] };
    this.customResponseHandler = this.customResponseHandler.bind(this);
  }

  componentDidMount() {
    // When the component is mounted, register the custom response handler that will store the references to the
    // custom response events.
    const { instance } = this.props;
    instance.on({ type: 'customResponse', handler: this.customResponseHandler });
  }

  componentWillUnmount() {
    // Remove the custom response handler.
    const { instance } = this.props;
    instance.off({ type: 'customResponse', handler: this.customResponseHandler });
  }

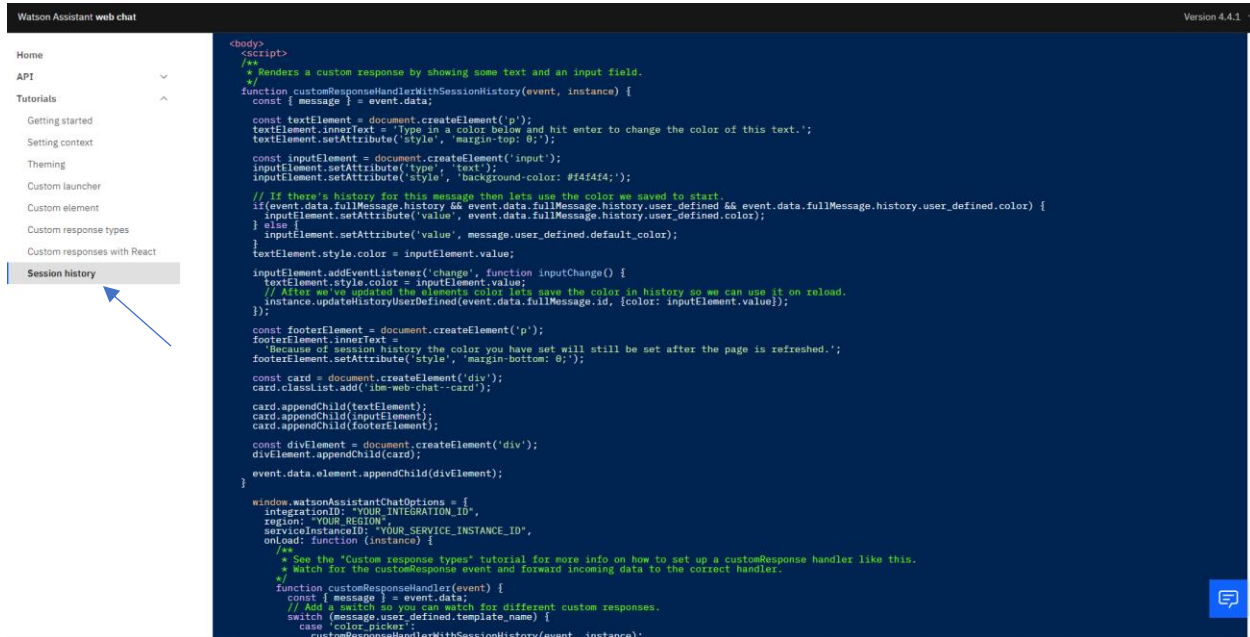
  customResponseHandler(event) {
    // This handler will fire each time a custom response occurs and we will update our state by appending the event
    // to the end of our elements list.
    this.setState(function appendEvent(prevState) {
      return {
        customResponseEvents: prevState.customResponseEvents.concat(event)
      };
    });
  }

  render() {
    // All we need to do to enable the react portals is to render each portal somewhere in your application (it
    // doesn't really matter where).
    return (
      <div>
        {this.state.customResponseEvents.map(function mapEvent(event, index) {
          return (
            <CustomResponseComponentPortal key={index} hostElement={event.data.element}>
              {/* This is your custom response component. It can be whatever you like that renders the given message
              in whatever manner your application needs. */}
              <CustomResponseComponent message={event.data.message} />
            </CustomResponseComponentPortal>
          );
        })}
      </div>
    );
  }
}

/**
 * This is the component that will attach a React portal to the given host element. The host element is the element
 * provided by the chat widget where your custom response will be displayed in the DOM. This portal will attach
 * any React children passed to it under this component so you can render the response using your own React

```

In the Session history your user's conversations will continue seamlessly across page changes and reloads. Session history allows your web chats to maintain conversation history and context when customers refresh the page or change to a different page on the same website.



The screenshot shows the IBM Watson Assistant web chat interface. On the left, there is a navigation menu with the following items: Home, API, Tutorials, Getting started, Setting context, Theming, Custom launcher, Custom element, Custom response types, Custom responses with React, and Session history. The 'Session history' item is highlighted with a blue bar and a blue arrow pointing to it. The main area of the interface is a code editor displaying JavaScript code. The code is for a custom response handler that implements session history. It includes a function `customResponseHandlerWithSessionHistory` that takes an event and an instance as arguments. The function creates a text element and an input field. It checks if there is a history for the message and sets the input value accordingly. It also adds an event listener to the input field to update the history when the user changes the color. The code also includes a footer element and a card element. The code is as follows:

```
<body>
<script>
/**
 * Renders a custom response by showing some text and an input field.
 */
function customResponseHandlerWithSessionHistory(event, instance) {
  const { message } = event.data;

  const textElement = document.createElement('p');
  textElement.innerHTML = 'Type in a color below and hit enter to change the color of this text.';
  textElement.setAttribute('style', 'margin-top: 0;');

  const inputElement = document.createElement('input');
  inputElement.setAttribute('type', 'text');
  inputElement.setAttribute('style', 'background-color: #f4f4f4;');

  // If there's history for this message then lets use the color we saved to start.
  if(event.data.fullMessage.history && event.data.fullMessage.history.user_defined && event.data.fullMessage.history.user_defined.color) {
    inputElement.setAttribute('value', event.data.fullMessage.history.user_defined.color);
  } else {
    inputElement.setAttribute('value', message.user_defined.default_color);
  }
  inputElement.setAttribute('value', message.user_defined.default_color);
  textElement.style.color = inputElement.value;

  inputElement.addEventListener('change', function inputChange() {
    textElement.style.color = inputElement.value;
    // After we've updated the elements color lets save the color in history so we can use it on reload.
    instance.updateHistoryUserDefined(event.data.fullMessage.id, {color: inputElement.value});
  });

  const footerElement = document.createElement('p');
  footerElement.innerHTML =
    'Because of session history the color you have set will still be set after the page is refreshed.';
  footerElement.setAttribute('style', 'margin-bottom: 8;');

  const card = document.createElement('div');
  card.classList.add('ibm-web-chat-card');
  card.appendChild(textElement);
  card.appendChild(inputElement);
  card.appendChild(footerElement);

  const divElement = document.createElement('div');
  divElement.appendChild(card);
  event.data.element.appendChild(divElement);
}

window.watsonAssistantChatOptions = {
  integrationID: "YOUR_INTEGRATION_ID",
  region: "YOUR_REGION",
  serviceInstanceID: "YOUR_SERVICE_INSTANCE_ID",
  onLoad: function (instance) {
    /**
     * See the "Custom response types" tutorial for more info on how to set up a customResponse handler like this.
     * Watch for the customResponse event and forward incoming data to the correct handler.
     */
    function customResponseHandler(event) {
      const { message } = event.data;
      // Add a switch so you can watch for different custom responses.
      switch (message.user_defined.template_name) {
        case 'color_picker':
          customResponseHandlerWithSessionHistory(event, instance);
        }
      }
    }
  }
}
```

Finally, you've got into the IBM Watson Assistant which is one of the most popular virtual system for information for website, at an easy and without a cost. The Watson Assistant can be used to serve customer self-service, employee self-service, and agent assist. Without the Watson Assistant these services were provided by human-serviced chat, email, and call services. So now after following the prompts, you can have in hand in a fast-paced work the Watson assistant web chat by using these steps to get all the codes that are available for you to add into your webpage to run your Watson Assistant.