Webhook service

To use fulfillment in a production system, you should implement and deploy a webhook service. To handle fulfillment, your webhook service needs to accept JSON requests and return JSON responses as specified in this guide. The detailed processing flow for fulfillment and webhooks is described in the fulfillment overview document (/dialogflow/docs/fulfillment-overview).

Webhook service requirements

The following requirements must be met by your webhook service:

- It must handle HTTPS requests. HTTP is not supported. If you host your webhook service on Google Cloud Platform using a Compute (/products/compute) or Serverless Computing (/serverless) solution, see the product documentation for serving with HTTPS. For other hosting options, see Get an SSL certificate for your domain (https://support.google.com/domains/answer/7630973).
- Its URL for requests must be publicly accessible.
- It must handle POST requests with a JSON `WebhookRequest` (#webhook_request) body.
- It must respond to `WebhookRequest` requests with a JSON `WebhookResponse` (#webhook_response) body.

Authentication

Note: Webhook source IPs are in the subset of those listed at https://www.gstatic.com/ipranges/goog.json (https://www.gstatic.com/ipranges/goog.json) that *excludes* those at https://www.gstatic.com/ipranges/cloud.json (https://www.gstatic.com/ipranges/cloud.json). Beyond this, Dialogflow cannot make any guarantees about IP ranges for machines that send webhook requests. Rather than restricting access via IP range, you should use one of the authentication methods listed below.

It's important to secure your webhook service, so that only you or your Dialogflow agent are authorized to make requests. Dialogflow supports the following mechanisms for authentication:

Login username and password	For webhook settings, you can specify optional login username and password values. If supplied, Dialogflow adds an authorization HTTP header to webhook requests. This header is of the form: `"authorization: Basic <base 64 encoding of the string username:password>"`.
Authentication headers	For webhook settings, you can specify optional HTTP header key-value pairs. If supplied, Dialogflow adds these HTTP headers to webhook requests. It is common to provide a single pair with a key of `authorization`.
Cloud Functions built-in authentication	You can use the built-in authentication when using Cloud Functions (#gcf). In order to use this type of authentication, do not supply login username, login password, or authorization headers. If you do supply any of these fields, these fields will be used for authentication rather than the built-in authentication.
Service identity tokens	You can use service identity tokens (#id-token) for authentication. If you do not supply login username, login password, or a header with a key of `authorization`, Dialogflow automatically assumes that service identity tokens should be used and adds an authorization HTTP header to webhook requests. This header is of the form: `"authorization: Bearer <identity token>"`.
Mutual TLS authentication	See the Mutual TLS authentication (/dialogflow/es/docs/fulfillment-mtls) documentation.

Webhook request

When an intent configured for fulfillment is matched, Dialogflow sends an HTTPS POST webhook request to your webhook service. The body of this request is a JSON object with information about the matched intent.

In addition to the end-user query, many integrations also send some information about the end-user as well. For example, an ID to uniquely identify the user. This information can be accessed via the `originalDetectIntentRequest` field in the webhook request, which will contain the information sent from the integration platform.

See the **`WebhookRequest`** (/dialogflow/docs/reference/common-types#webhookrequest) reference documentation for details.

Here is a sample request:

```
{
  "responseId": "response-id",
  "session": "projects/project-id/agent/sessions/session-id",
  "queryResult": {
    "queryText": "End-user expression",
    "parameters": {
      "param-name": "param-value"
    },
    "allRequiredParamsPresent": true,
    "fulfillmentText": "Response configured for matched intent",
    "fulfillmentMessages": [
      {
        "text": {
          "text": [
            "Response configured for matched intent"
          ]
        }
      }
    ],
    "outputContexts": [
      {
        "name": "projects/project-id/agent/sessions/session-id/contexts/con
        "lifespanCount": 5,
        "parameters": {
          "param-name": "param-value"
        }
      }
    ],
    "intent": {
      "name": "projects/project-id/agent/intents/intent-id",
      "displayName": "matched-intent-name"
    },
    "intentDetectionConfidence": 1,
    "diagnosticInfo": {},
    "languageCode": "en"
  },
  "originalDetectIntentRequest": {}
}
```

Webhook response

Once your webhook receives a webhook request, it needs to send a webhook response. The body of this response is a JSON object with the following information:

- The response (/dialogflow/docs/intents-responses) that Dialogflow returns to the end-user.
- Updates to contexts (/dialogflow/docs/contexts-overview) active for the conversation.
- A follow-up event (/dialogflow/docs/events-overview) to trigger an intent match.
- A custom payload (/dialogflow/docs/intents-rich-messages#custom) to be sent to the integration (/dialogflow/docs/integrations) or detect intent client (/dialogflow/docs/api-overview#detect-intent)

The following limitations apply to your response:

- The response must occur within 10 seconds for Google Assistant (/dialogflow/docs/integrations/aog) applications or 5 seconds for all other applications, otherwise the request will time out.
- The response must be less than or equal to 64 KiB in size.

See the `WebhookResponse` (/dialogflow/docs/reference/common-types#webhookresponse) reference documentation for details.

Text response

Example for a text response (/dialogflow/docs/intents-rich-messages#text):

```
{
  "fulfillmentMessages": [
    {
      "text": {
        "text": [
          "Text response from webhook"
        ]
      }
    }
  ]
}
```

Card response

Example for a card response (/dialogflow/docs/intents-rich-messages#card):

```
{
  "fulfillmentMessages": [
    {
      "card": {
        "title": "card title",
        "subtitle": "card text",
        "imageUri": "https://example.com/images/example.png",
        "buttons": [
          {
            "text": "button text",
            "postback": "https://example.com/path/for/end-user/to/follow"
          }
        ]
      }
    }
  ]
}
```

Google Assistant response

Example for a Google Assistant response (/dialogflow/docs/intents-rich-messages#aog):

```
{
  "payload": {
    "google": {
      "expectUserResponse": true,
      "richResponse": {
        "items": [
          {
            "simpleResponse": {
              "textToSpeech": "this is a Google Assistant response"
            }
          }
        ]
      }
    }
  }
}
```

Context

Example that sets output context (/dialogflow/docs/contexts-input-output#output_contexts):

```
{
  "fulfillmentMessages": [
    {
      "text": {
        "text": [
          "Text response from webhook"
        ]
      }
    }
  ],
  "outputContexts": [
    {
      "name": "projects/project-id/agent/sessions/session-id/contexts/conte
      "lifespanCount": 5,
      "parameters": {
        "param-name": "param-value"
      }
    }
  ]
}
```

Event

Example that invokes a custom event (/dialogflow/docs/events-custom):

```
{
  "followupEventInput": {
    "name": "event-name",
    "languageCode": "en-US",
    "parameters": {
      "param-name": "param-value"
    }
  }
}
```

Session entity

Example that sets a session entity (/dialogflow/docs/entities-session):

```
{
  "fulfillmentMessages": [
    {
      "text": {
        "text": [
          "Choose apple or orange"
        ]
      }
    }
  ],
  "sessionEntityTypes":[
    {
      "name":"projects/project-id/agent/sessions/session-id/entityTypes/frui
      "entities":[
        {
          "value":"APPLE_KEY",
          "synonyms":[
            "apple",
            "green apple",
            "crabapple"
          ]
        },
        {
          "value":"ORANGE_KEY",
          "synonyms":[
            "orange"
          ]
        }
      ],
      "entityOverrideMode":"ENTITY_OVERRIDE_MODE_OVERRIDE"
    }
  ]
}
```

Custom payload

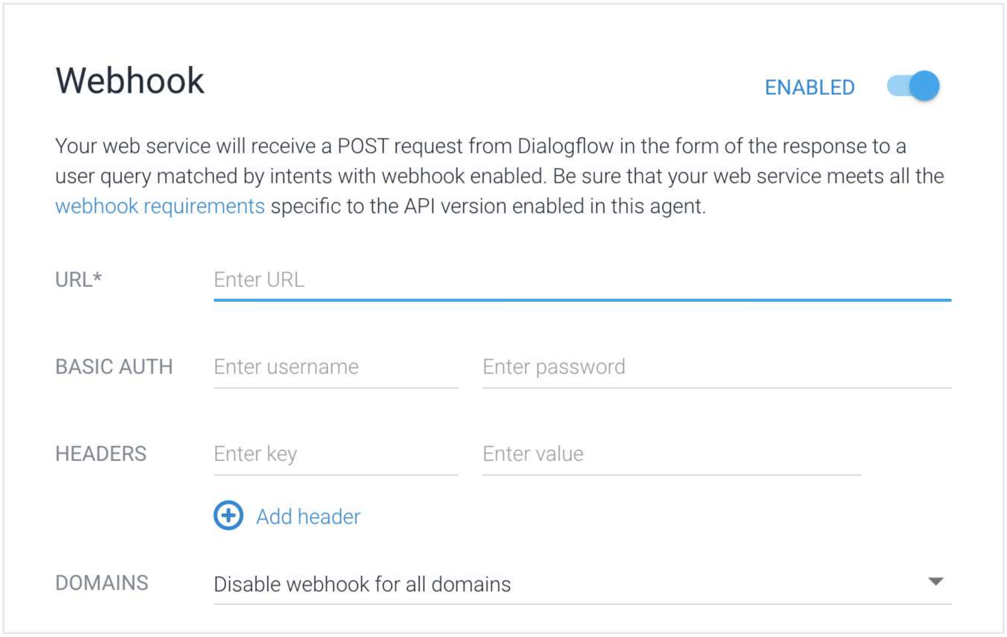
Example that provides a custom payload:

```
{
  "fulfillmentMessages": [
    {
      "payload": {
        "facebook": { // for Facebook Messenger integration
          "attachment": {
            "type": "",
            "payload": {}
          }
        },
        "slack": { // for Slack integration
          "text": "",
          "attachments": []
        },
        "richContent": [ // for Dialogflow Messenger integration
          [
            {
              "type": "image",
              "rawUrl": "https://example.com/images/logo.png",
              "accessibilityText": "Example logo"
            }
          ]
        ],
        // custom integration payload here
      }
    }
  ]
}
```

Enable and manage fulfillment

To enable and manage fulfillment for your agent with the console:

1. Go to the Dialogflow ES console (https://dialogflow.cloud.google.com).
2. Select an agent.
3. Select **Fulfillment** in the left sidebar menu.
4. Toggle the **Webhook** field to **Enabled**.
5. Provide the details for your webhook service in the form. If your webhook doesn't require authentication, leave the authentication fields blank.
6. Click **Save** at the bottom of the page.



Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the webhook requirements specific to the API version enabled in this agent.

To enable and manage fulfillment for your agent with the API, see the agent reference (/dialogflow/docs/reference/common-types#agents). The `getFulfillment` and `updateFulfillment` methods can be used to manage fulfillment settings.

To enable fulfillment for an intent with the console:

1. Select **Intents** in the left sidebar menu.
2. Select an intent.
3. Scroll down to the **Fulfillment** section.
4. Toggle **Enable webhook call for this intent** to on.
5. Click **Save**.

To enable fulfillment for an intent with the API, see the intents reference (/dialogflow/docs/reference/common-types#intents). Set the `webhookState` field to `WEBHOOK_STATE_ENABLED`.

Webhook errors

If your webhook service encounters an error, it should return one of the following HTTP status codes:

- `400` Bad Request
- `401` Unauthorized
- `403` Forbidden
- `404` Not found
- `500` Server fault
- `503` Service Unavailable

In any of the following error situations, Dialogflow responds to the end-user with the built-in response configured for the intent currently matched:

- Response timeout exceeded.
- Error status code received.

- Response is invalid.
- Webhook service is unavailable.

In addition, if the intent match was triggered by a detect intent API call (/dialogflow/docs/api-overview#detect-intent), the `status` field in the detect intent response contains the webhook error information. For example:

```
"status": {
    "code": 206,
    "message": "Webhook call failed. <details of the error...>"
}
```

Automatic retries

Dialogflow ES includes internal mechanisms that automatically retry on certain webhook errors to improve robustness. Only non-terminal errors are retried (for example, timeout or connection errors).

To reduce the likelihood of duplicated calls:

- Set longer webhook timeout thresholds.
- Support idempotency in webhook logic or deduplicate.

Using Cloud Functions

There are a few ways to use Cloud Functions for fulfillment. The Dialogflow inline editor (/dialogflow/es/docs/fulfillment-inline-editor) integrates with Cloud Functions (/functions/docs). When you use the inline editor to create and edit your webhook code, Dialogflow establishes a secure connection to your Cloud Function.

You also have the option to use a Cloud Function not created by the inline editor (perhaps because you want to use a language other than Node.js). If the Cloud Function resides in the same project as your agent, your agent can call your webhook without needing any special configuration.

However, there are two situations in which you must manually setup this integration:

1. The **Dialogflow Service Agent** service account (/iam/docs/understanding-service-accounts) with the following address must exist for your agent project:

   ```
   service-agent-project-number@gcp-sa-dialogflow.iam.gserviceaccount.com
   ```

   This special service account and the associated key is normally created automatically when you create the first agent for a project. If your agent was created before May 10, 2021, you may need to trigger creation of this special service account with the following:

   a. Create a new agent for the project.

   b. Execute the following command:

      ```
      gcloud beta services identity create --service=dialogflow.googleapis.co
      ```

2. If your webhook function resides in a different project than the agent, you must provide the **Cloud Functions Invoker** IAM role (/iam/docs/understanding-roles) to the **Dialogflow Service Agent** service account in your function's project.

Service identity tokens

When Dialogflow calls a webhook, it provides a Google identity token (https://developers.google.com/identity/sign-in/web/backend-auth) with the request. Any webhook can optionally validate the token using Google client libraries or open source libraries like github.com/googleapis/google-auth-library-nodejs (https://github.com/googleapis/google-auth-library-nodejs). For example, you can verify the `email` of the ID token as:

```
service-agent-project-number@gcp-sa-dialogflow.iam.gserviceaccount.com
```

Samples

The following samples show how to receive a `WebhookRequest` and send a `WebhookResponse`. These samples reference intents created in the quickstart (/dialogflow/docs/quick/build-agent).

GoJava (#java)Node.js (#node.js)Python (#python)
(#go)

To authenticate to Dialogflow, set up Application Default Credentials. For more information, see Set up authentication for a local development environment (/docs/authentication/set-up-adc-local-dev-environment).

```
import (
  "encoding/json"
  "fmt"
  "log"
  "net/http"
)

type intent struct {
  DisplayName string `json:"displayName"`
}

type queryResult struct {
  Intent intent `json:"intent"`
}

type text struct {
  Text []string `json:"text"`
}

type message struct {
  Text text `json:"text"`
}

// webhookRequest is used to unmarshal a WebhookRequest JSON object. Note
// not all members need to be defined--just those that you need to process
// As an alternative, you could use the types provided by
// the Dialogflow protocol buffers:
// https://godoc.org/google.golang.org/genproto/googleapis/cloud/dialogflo
type webhookRequest struct {
  Session     string      `json:"session"`
  ResponseID  string      `json:"responseId"`
  QueryResult queryResult `json:"queryResult"`
}

// webhookResponse is used to marshal a WebhookResponse JSON object. Note 
// not all members need to be defined--just those that you need to process
// As an alternative, you could use the types provided by
// the Dialogflow protocol buffers:
// https://godoc.org/google.golang.org/genproto/googleapis/cloud/dialogflo
type webhookResponse struct {
  FulfillmentMessages []message `json:"fulfillmentMessages"`
}

// welcome creates a response for the welcome intent.
func welcome(request webhookRequest) (webhookResponse, error) {
  response := webhookResponse{
    FulfillmentMessages: []message{
      {
        Text: text{
          Text: []string{"Welcome from Dialogflow Go Webhook"},
        },
      },
    },
  }
  return response, nil
}

// getAgentName creates a response for the get-agent-name intent.
func getAgentName(request webhookRequest) (webhookResponse, error) {
  response := webhookResponse{
    FulfillmentMessages: []message{
      {
        Text: text{
          Text: []string{"My name is Dialogflow Go Webhook"},
        },
      },
    },
  }
  return response, nil
}

// handleError handles internal errors.
func handleError(w http.ResponseWriter, err error) {
  w.WriteHeader(http.StatusInternalServerError)
  fmt.Fprintf(w, "ERROR: %v", err)
}

// HandleWebhookRequest handles WebhookRequest and sends the WebhookRespons
func HandleWebhookRequest(w http.ResponseWriter, r *http.Request) {
  var request webhookRequest
  var response webhookResponse
  var err error

  // Read input JSON
  if err = json.NewDecoder(r.Body).Decode(&request); err != nil {
    handleError(w, err)
    return
  }
  log.Printf("Request: %+v", request)

  // Call intent handler
  switch intent := request.QueryResult.Intent.DisplayName; intent {
  case "Default Welcome Intent":
    response, err = welcome(request)
  case "get-agent-name":
    response, err = getAgentName(request)
  default:
    err = fmt.Errorf("Unknown intent: %s", intent)
  }
  if err != nil {
    handleError(w, err)
    return
  }
  log.Printf("Response: %+v", response)

  // Send response
  if err = json.NewEncoder(w).Encode(&response); err != nil {
    handleError(w, err)
    return
  }
}
```

Previous
← Inline editor (/dialogflow/es/docs/fulfillment-inline-editor)

Next
Webhook for slot filling (/dialogflow/es/docs/fulfillment-webhook-slot-filling) →

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/), and code samples are licensed under the Apache 2.0 License (https://www.apache.org/licenses/LICENSE-2.0). For details, see the Google Developers Site Policies (https://developers.google.com/site-policies). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2025-05-05 UTC.