Low-code from frontend to backend: Connecting conversational user interfaces to backend services via a low-code IoT platform

IRENE WEBER, Kempten University of Applied Sciences, Germany

Current chatbot development platforms and frameworks facilitate setting up the language and dialog part of chatbots, while connecting it to backend services and business functions requires substantial manual coding effort and programming skills. This paper proposes an approach to overcome this situation. It proposes an architecture with a chatbot as frontend using an IoT (Internet of Things) platform as a middleware for connections to backend services. Specifically, it elaborates and demonstrates how to combine a chatbot developed on the open source development platform Rasa with the open source platform Node-RED, allowing low-code or no-code development of a transactional conversational user interface from frontend to backend.

Versions

1 INTRODUCTION

Chatbots offer a means to use services and control smart home devices connected to the web. They can also serve as conversational user interfaces to application systems in other contexts, including professional work. Chatbots that accomplish tasks for users are termed transactional chatbots, as opposed to conversational chatbots that engage in dialogs with users mainly for social reasons or entertainment [6].This paper addresses transactional chatbots serving as conversational user interfaces.

Compared to graphical user interfaces, conversational user interfaces can bring several advantages: They provide a more natural, user-controlled, and flexible way to interact with computer systems than graphical user interfaces. They make working with computer systems easier, especially when users have to complete

---

Author's address: Irene Weber, irene.weber@hs-kempten.de, Kempten University of Applied Sciences, Kempten, Germany.

tasks only occasionally. As a result, chatbots have become popular as user interfaces for customer self-service functions and the like.

Conversational user interfaces also have great potential in a professional environment as they can help employees complete non-routine tasks more smoothly and efficiently, such as filling request forms, retrieving information from knowledge bases, adding new entries to knowledge bases, filing reports, and more. Conversational user interfaces can serve as a single and unified entry point to multiple system functions and services, eliminating the need for users to become familiar with various application systems and software tools and their respective page, tab, and menu structures.

In order to perform tasks on behalf of users, a transactional conversational user interface, serving as a frontend, needs to access services, devices, databases, and application systems as backend. Consequently, the connectivity of the conversational user interface's language-processing components to external systems and services is critical.

The technical aspects of the overall architecture of a system with a conversational user interface have received rather little attention in research. Attention may increase recently, as it may indicate interesting research opportunities. For instance, Baez et al [1] conducted a literature search and reviewed papers that address architectural aspects of integrating chatbots with existing external systems. They identified seven integration patterns. Their goal is to stimulate research on development aids that specifically support the development of bots' language capabilities for a given integration pattern. They also note that they have found little work on integrating a conversational user interface for specific use cases (namely, *API caller* and *business process interface*), despite the great practical potential that these use cases promise. Rough and Cowan [10] surveyed current chatbot development platforms and frameworks. They found that the platforms and frameworks make it easy to set up the voice and dialog portion of a chatbot, while connecting to backend services and business functions requires significant manual coding effort and programming skills.

This paper proposes a system architecture with a chatbot as frontend using an IoT (Internet of Things) platform as a middleware for connections to backend services. Specifically, it elaborates and demonstrates how to combine a chatbot developed on the open source development platform Rasa with the open source platform Node-RED, allowing low-code or no-code development of a transactional conversational user interface from frontend to backend.

2 ARCHITECTURE OF CONVERSATIONAL USER INTERFACES

2.1 Standard architecture of a conversational user interface

The main components of a chatbot are Language Understanding, Dialog Management, and Language Generation. The chatbot may accept typed input or spoken input and produce written or speech output. When conversing by speech, the architecture has to provide for text-to-speech- and speech-to-text transformation. Figure 1 shows a typical architecture of a conversational interface [6, 10].

The Language Understanding component analyzes language input and maps it to intents and entities. The Language Generation component composes the output of the chatbot. The Dialog Management component controls the chatbot's behavior. It also holds the slots that implement the chatbot's memory where it stores entities and the state of the dialog.
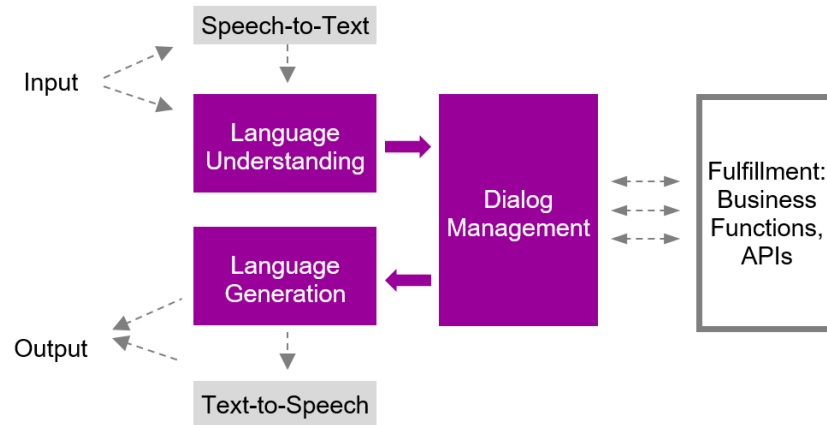
Fig. 1. Typical architecture of a conversational interface, adapted from [6, 10]

The Dialog Management component determines how the chatbot acts upon user inputs, given the recognized intents, entities, slot values, and dialog history so far. In any case, the chatbot eventually provides some output to the user in order to continue the dialog. Before doing so, it may access backend systems to perform a task or retrieve information and include the outcome in its output to the user. The component performing tasks and calling external services constitutes the Fulfillment part of the chatbot.

As [10] have stated, current chatbot development platforms provide good support and tools for developing the Language Understanding, Dialog Management, and Language Generation components, while implementing the Fulfillment part and its interfaces to backend systems requires manual programming skills and effort. This complicates building a conversational user interface, increases cost and development time, and prevents end users from setting up a CUI as their personal conversational assistant. In addition, this is a significant disadvantage for training and hands-on classroom exercises when teaching on conversational user interfaces. This work proposes an approach to alleviate this problem for the chatbot development platform Rasa [9].

2.2 Proposed architecture for Rasa CUIs with Node-RED

In the architecture of a Rasa chatbot, a so-called action server handles fulfillment. The action server runs as a standalone server and bundles and manages calls to external APIs. The Rasa platform provides a Python SDK for developing an action server executing custom actions. While the SDK facilitates the development, Python coding is still necessary.

The approach proposed here replaces the Python-coded action server by an action server hosted on Node-RED. Figure 2 depicts the resulting architecture.

3 IMPLEMENTING THE RASA ACTION SERVER HTTP API ON NODE-RED

This section gives an example on how to implement an action server, i.e. a fulfillment component, for a Rasa chatbot by means of visual programming on the Node-RED platform. It introduces the basics of Node-RED,
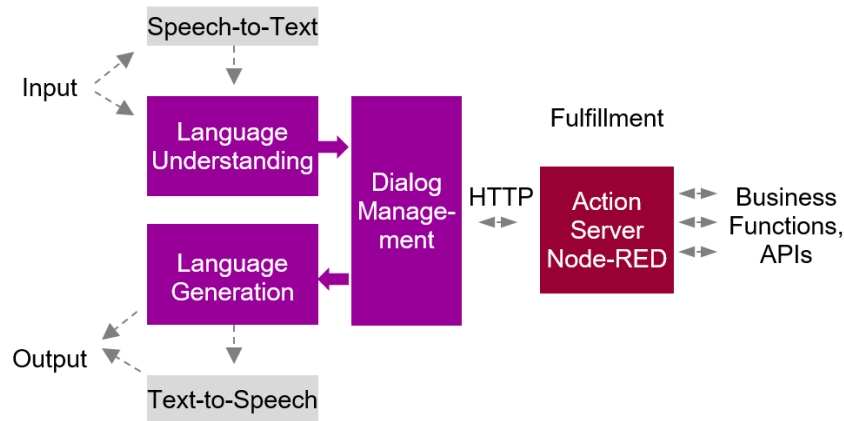
Fig. 2. Architecture of a conversational user interface based on Rasa

explains different types of nodes that are useful for building an action server, and shows a complete example flow.

3.1 Node-RED Basics

Node-RED is a low code platform running on Node.js. It is especially popular for connecting IoT devices and developing, e.g., home automation applications. It brings a browser-based GUI with a visual flow editor and a palette of nodes. Nodes are drag-and-dropped from the palette onto the flow editor and interconnected to flows. When triggered, a flow transports a message object, typically coded in JSON, from node to node, with the nodes augmenting or transforming the message object and, optionally, performing additional activities, such as calls to further services or APIs. Information extracted from the message object can parameterize the API calls. Different types of events can trigger a flow, one of them being an incoming HTTP request. It is also possible to trigger a flow manually for development and testing.

The Node-RED palette holds a basic set of standard nodes. Different types of nodes implement conditional branching or loops, manipulate message objects, communicate with external systems as a server or as a client, catch and handle errors, or display debugging information.

Besides the standard nodes, Node-RED can import and use custom nodes. The Node-RED ecosystem offers a broad variety of community-contributed nodes, including nodes that, e.g., connect to databases, access web APIs, write to spreadsheets, control devices, and more. If needed, experienced programmers may implement own custom nodes.

3.2 Custom nodes implementing the Rasa action server HTTP API

An action server for a Rasa chatbot must implement the HTTP API of Rasa action servers as specified in [8]. The author provides a collection of specialized nodes, node-red-contrib-rasa-actionserver [12], abbreviated as *rasaas,* that facilitates building the action server for Rasa on Node-RED. The *rasaas* nodes are publicly available

for download. Node-RED allows searching and installing these and other custom nodes from its graphical user interface. After installation, the nodes become available in the Node-RED palette.

3.3 Example application

The application is a simple chatbot that provides information about geographical locations, such as cities or regions. Specifically, it can provide information about the current weather or provide a link to the Wikipedia page of a location named by the user. This standard sample application is an extended version of [7]. It is sufficient to demonstrate the approach, since it accesses two external APIs (weatherstack.com and Wikipedia OpenSearch). For better understanding, a sample dialog with the bot is shown in Figure 3.
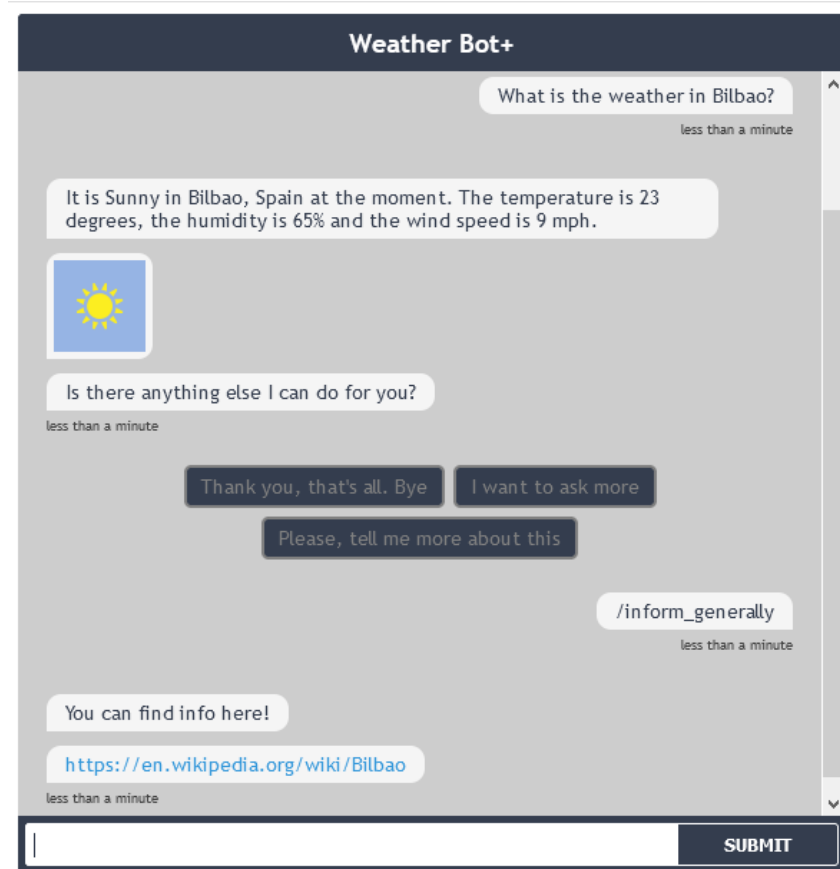

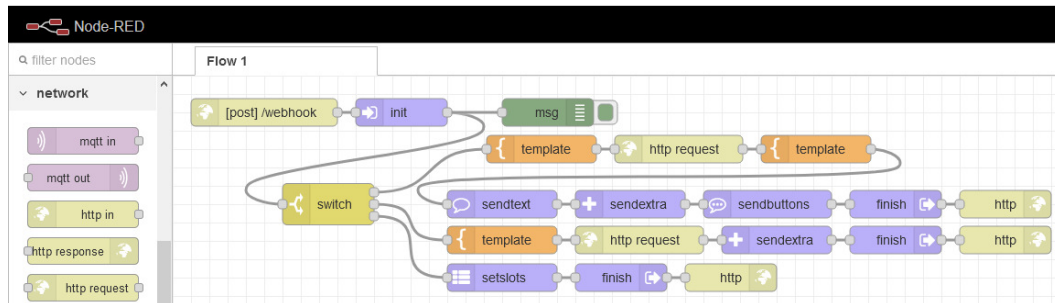Fig. 3. Sample dialog, using the chatroom of [11]

Fig. 4. Node-RED Flow editor with an example flow implementing an action server, i.e., fulfillment component, for a Rasa conversational user interface. To the left, the network section of the Node-RED palette is partially visible.

3.4 Building the action server flow

3.4.1 HTTP Endpoint. Rasa dialog management sends a HTTP request to the action server carrying information about the state of dialog coded in JSON. The basic, most relevant pieces of information are the name of the specific action that the action server has to perform, and slots with their current values.

Enabling communication between the chatbot's Dialog Management and its Node-RED action server is straightforward, as the sole requirement for a custom action server is to provide an HTTP endpoint conforming to the HTTP API specification of Rasa action servers [8]. Node-RED provides a pair of nodes that implement an HTTP endpoint, the *http in* and *http response* nodes. This pair of nodes represent start and end of a Node-RED flow implementing an action server for RASA.

The *http in* node accepts incoming HTTP requests. In Figure 4, the *http in* node is labeled by the HTTP method and the name of the endpoint, "[post]/webhook". The *http response* node delivers the HTTP response. In Figure 4, it is labeled "http" and shows up multiple times as the end of several branches of the flow.

A pair of *rasaas* nodes, *init* and *finish,* respectively, unpack and store the information carried by an incoming Rasa action request or combine and transform results of an action into an HTTP response object coded as JSON according to the Rasa Action Server API. The *init* node follows the *http in* node. A *finish* node precedes each *http response* node, as shown in Figure 4.

3.4.2 Branching into specific actions. The action server in Figure 4 implements three different actions, two of which involve calling an external API (*action_weather* and *action_generalinfo*). A third action, *action_clearlocation*, sets a slot value. The chatbot includes the name of the intended action into its HTTP call to the action server. The *init* node stores the action name into a specific "action" field of the flow's message object, to be evaluated by a *switch* node. The *switch* node is a standard Node-RED node. In the flow in Figure 4, it selects the appropriate branch for a particular action. It has three outgoing branches, each of them corresponding to one of the three actions. To give an impression of how this actually works, Figure 5 hows the configuration view of the *switch* node. The configuration view of a node opens with a double click.

3.4.3 Generating responses and slot-setting events. The HTTP response of the action server contains a list of responses for the chatbot to utter in reply to the user's input, and a list of events manipulating slots. Responses may take the form of text responses, attachments, images, buttons, and more. The *rasaas* node collection
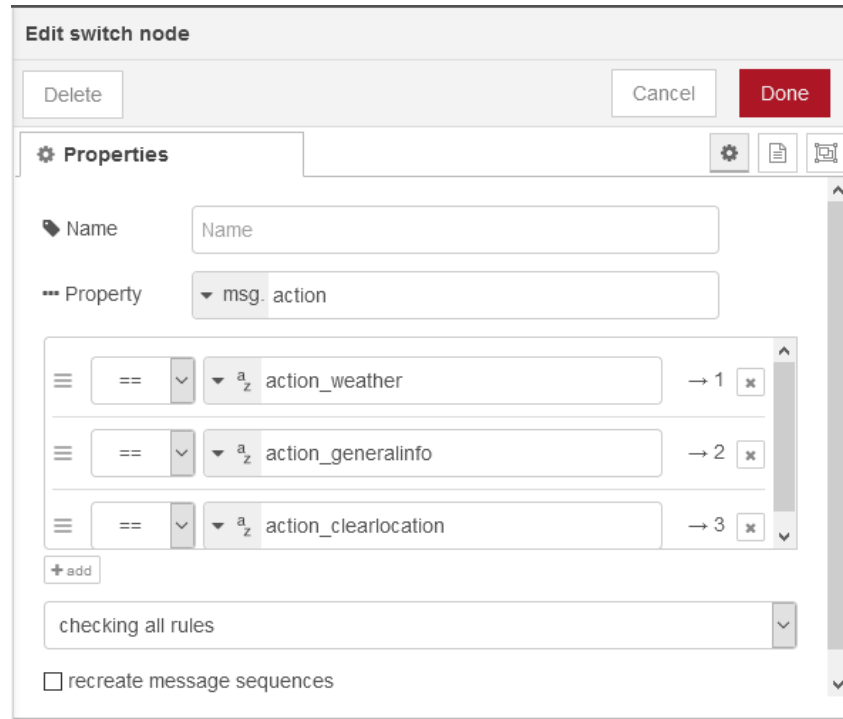
Fig. 5. Example configuration of a *switch* node in Node-RED

provides a node *sendtext* for generating text responses, a node *sendextra* for generating attachment and image responses, and a node type *sendbuttons* for generating button responses. Figure 6 shows the configuration view of a *sendbuttons* node. A *setslots* node generates events setting Rasa slots. A *finish* node collects and assembles the outputs of these nodes as is shown in Figure 4.

3.4.4 Calling external APIs. Calling an external API requires preparing the URL and query parameters of the API. The results of an API need to be analyzed and transformed into a chatbot response. The flow in Figure 4 uses *template* nodes for these tasks.

A *template* node is a standard node type in Node-RED. As its name indicates, the configuration of a *template* node consists of a string template containing placeholders. When the flow is executed, values taken from the message object substitute the placeholders.

A *template* node can hold a template URL string with placeholders for query parameters. The node extracts query parameters from the incoming message object and inserts them into the URL template. Then, an *HTTP request* node suffices for executing the API call. Like *template* nodes, *HTTP request* nodes belong to Node-RED standard nodes. An exemplary template for an API call is:

http://api.weatherstack.com/current?access_key= xxx &query={{slots.location}}

Fig. 6. Example configuration of a *sendbuttons* node in Node-RED

Double curly braces designate placeholders. The expressions within the curly braces specify how to extract the substituting values from the message object.

Another *template* node can hold a response template. When executed, it inserts parts of the API response result into the placeholders and propagates the resulting bot response to a *sendtext* node. An example of such a template string is:

It is {{payload.current.weather_descriptions.0}} in {{payload. location.name}}, {{payload.location.country}} at the moment.

3.5 Developing, deploying and debugging

Developing an action server with Node-RED means adding and configuring nodes and drawing flows. The exemplary action server flow described here implements three actions. It is easy to implement more actions by adding more branches to the flow after the *switch* node.

The flow in Figure 4 contains one other node type, namely, a standard Node-RED *debug* node labelled "msg". When triggered, a *debug* node prints the message object or parts of the message object to a debug tab in the Node-RED GUI, making errors and problems visible.

The Rasa action server API includes two additional services an action server should implement: a */health* service reporting the operational state of the action server, and an */actions* service listing the actions the server is capable to execute. These are easy to implement by flows starting with corresponding *http in* nodes.

Clicking a Deploy button in the Node-RED GUI immediately activates modifications or additions to the flow. This yields very quick and convenient development cycles.

4 DISCUSSION AND FUTURE WORK

This paper proposes a system architecture in which the open IoT platform Node-RED serves as middleware for the Fulfillment component of a conversational UI implemented with the Rasa chatbot development platform. This approach has several benefits.

From a technical viewpoint, Node-RED is a good fit for a Rasa chatbot, as both systems are available as open source and deployable on premise, giving developers and operators full ownership over their systems and data. The overall architecture is clear and simple, as both systems remain separate and communicate via HTTP by a compact API. This differs from the approach of [3], which uses Node-RED as development and deployment platform for chatbots.

The system architecture also differs from chatbot development platforms like Kore.ai. Kore.ai provides visual programming with flows for defining dialogs. It also allows calling APIs from within the Kore.ai dialogs via built-in service nodes [4]. In contrast, the idea of the system architecture proposed here is to apply visual programming for building a uniform API giving access to a variety of heterogeneous services. This system architecture resembles the architecture of combining a Kore.ai chatbot with Blue Prism Robotic Process Automation via a connector [2, 4, 5]. In the terminology of [1], these architectures correspond to the integration pattern *conversational business process interface*.

The system architecture proposed here is motivated by the intent to facilitate the development of an action server for the Rasa chatbot to the point where even non-programmers can successfully build a working transactional chatbot including its fulfillment component. To date, experience indicates good success: As an assignment in a lab exercise, a class of undergraduate students were asked to develop Rasa chatbots with Node-RED action servers as described in this paper, instructed by a detailed tutorial, and, in a next step, to extend their bots by additional and new functions. Despite lacking an IT background or programming skills, they were able to accomplish this task very satisfactorily with reasonable effort. In conclusion, Node-RED is a feasible solution for the problem of no-code or low-code development of service fulfillment for conversational user interfaces.

The potential of Node-RED goes beyond calling web APIs, since the community has contributed a wealth of custom nodes that connect to various systems including databases, spreadsheets, etc. In sum, however, the ease

of programming an interface also depends on the backend systems. For example, Node-RED has nodes that connect to databases, but retrieving the correct data still requires knowledge of the database query language.

So far, the proposed system architecture has been tested in prototyping and in classroom settings. Next, we intend to explore further integrations and evaluate the approach for real world applications. Rasa chatbot training data and a sample Node-RED flow that demo the approach can be found on GitHub [13].

ACKNOWLEDGMENTS

REFERENCES

[1] Marcos Baez, Florian Daniel, Fabio Casati, and Boualem Benatallah. 2020. Chatbot integration in few patterns. *IEEE Internet Computing* (2020), 1–1. https://doi.org/10.1109/mic.2020.3024605 Publisher: Institute of Electrical and Electronics Engineers (IEEE).

[2] Blue Prism. 2019. Blue Prism Technology Partners Continue to Augment Intelligent.... https://www.blueprism.com/news/blue-prism-technology-partners-continue-to-augment-intelligent-automation-by-adding-ai-powered-chatbot-and-data-analysis-capabilities-1/

[3] Guidone. 2020. guidone/node-red-contrib-chatbot. https://github.com/guidone/node-red-contrib-chatbot original-date: 2016-06-23T09:34:59Z.

[4] Kore.ai. [n.d.]. Build a Bot Using Kore.ai Bot Builder Platform without Coding. Try it. https://kore.ai/platform/design-and-build/dialog-builder/

[5] Kore.ai. 2019. BotKit SDK Tutorial – Blue Prism. https://developer.kore.ai/docs/bots/sdks/botkit-sdk-tutorial-blue-prism/ Section: SDKs.

[6] Michael McTear. 2018. Conversation modelling for chatbots: current approaches and future directions. *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2018* (2018), 175–185. Publisher: TUDpress, Dresden.

[7] Justina Petraityte. 2020. JustinaPetr/Weatherbot_Tutorial. https://github.com/JustinaPetr/Weatherbot_Tutorial original-date: 2018-02-06T08:38:24Z.

[8] Rasa Technologies GmbH. 2021. Rasa Action Server Documentation. https://rasa.com/docs/action-server/pages/action-server-api

[9] Rasa Technologies Inc. 2020. Open source conversational AI. https://rasa.com/

[10] Daniel Rough and Benjamin Cowan. 2020. Don't Believe The Hype! White Lies of Conversational User Interface Creation Tools. In *Proceedings of the 2nd Conference on Conversational User Interfaces (CUI '20)*. Association for Computing Machinery, Bilbao, Spain, 1–3. https://doi.org/10.1145/3405755.3406140

[11] scalableminds. 2020. GitHub - scalableminds/chatroom: React-based Chatroom Component for Rasa Stack. https://github.com/scalableminds/chatroom

[12] weberi. 2021. weberi/node-red-contrib-rasa-actionserver. https://github.com/weberi/node-red-contrib-rasa-actionserver original-date: 2021-02-01T04:51:45Z.

[13] weberi. 2021. weberi/rasaas-demo-flow. https://github.com/weberi/rasaas-demo-flow original-date: 2021-02-02T01:48:53Z.