**Chatbot through RASA**

1. **Introduction:**

A chatbot is an application that can initiate and continue a conversation using auditory and/or textual methods as a human would do. A chatbot can be either a simple rule-based engine or an intelligent application leveraging Natural Language Understanding. Many organizations today have started using chatbots extensively. Chatbots are becoming famous as they are available 24\*7, provide a consistent customer experience, can handle several customers at a time, are cost-effective and hence, results in a better overall customer experience.

* 1. **Uses**
* Customer support
* Frequently Asked Questions
* Addressing Grievances
* Appointment Booking
* Automation of routine tasks
* Address a query

1. **Prerequisites**

The prerequisites for developing and understanding a chatbot using Microsoft Azure are:

* Python installed
* Microsoft Build tools with visual c++ 14.0 installed. Link: <https://visualstudio.microsoft.com/downloads/>

1. **Introduction to RASA**

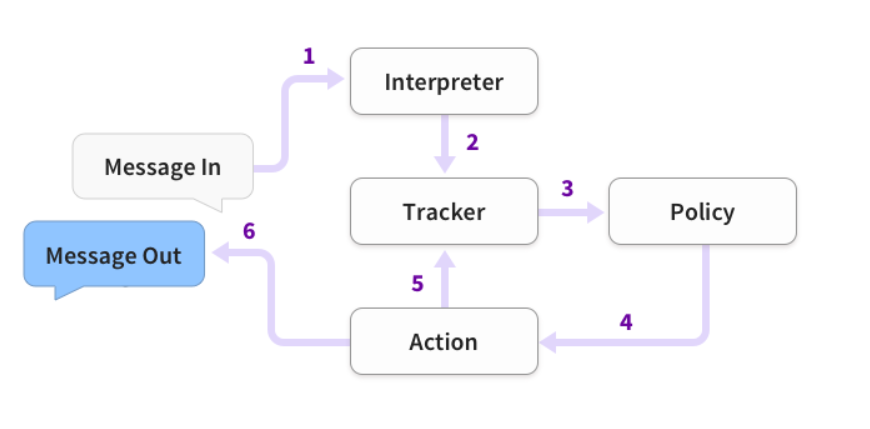
Rasa is an open source machine learning framework for automated text and voice-based conversations. Rasa is helpful in understanding messages, holding conversations and connecting to messaging channels and APIs.

The very good thing about Rasa is you are not tied to any pre-built models or use cases (like Dialogflow, etc). So, due to this very good reason you can customize your usecases which can be a market changer. Rasa is a rule-less framework so, you don’t have to worry about about putting your data on someone’s server or cloud as in Microsoft LUIS, Dialogflow or Amazon Lex.

There are two main components of Rasa: Rasa NLU and Rasa core

Rasa has two main modules:

* **NLU** for understanding user messages
* **Core** for holding conversations and deciding what to do next
* The main reason why open source NLU is used are:
* - All your training data is not dependent on Google, Microsoft, Amazon or facebook.
* - Machine Learning is not one-size-fits all. You can tweak and customize models for your training data.
* - Rasa NLU runs anywhere you want, so you don’t need to make any extra network request for every message.
* Now, let’s understand what is Rasa Core? Rasa Core place a major and the most essential part in generating the reply messages for chatbot. It consider the output of Rasa NLU (intents and entities) as an input and applies machine learning models to response with a bot reply.
  1. **RASA Architecture:**



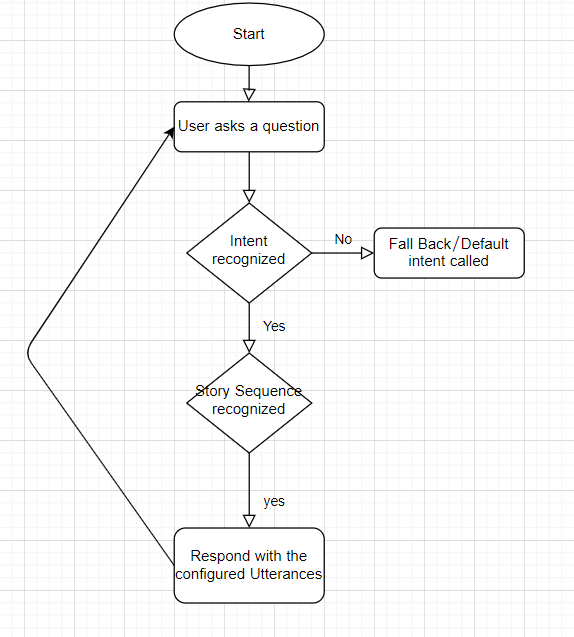
1. **The problem statement**

To build a covid-19 chatbot, which will be able to tell us a number of cases available in any of

the locality by there Pincode or district or state , and it will give all the instruction and preventive

measures.

* 1. **Technical stack:**
  + Python
  + Rasa X
  1. **The application flow**



1. **Implementation:**

* Create a new folder for your chatbot project.
* Open that folder using Pycharm
* Create a new environment for your chatbot project from pycharm or from anaconda prompt.
* Run the command **pip install rasa-x --extra-index-url**[**https://pypi.rasa.com/simple**](https://pypi.rasa.com/simple) for installing all the rasa dependencies
* Run the command **pip install spacy** for installing spacy library.
* Then enter the following commands:
  + - **python -m spacy download en**
    - **python -m spacy download en\_core\_web\_md**
    - **python -m spacy link en\_core\_web\_md en**
* After all this command run successfully, enter the command **rasa init** and for all the subsequent actions choose Y (for training the model etc).
* Open the ‘nlu.md’ file from the data folder and enter the following content:
* *## intent:corona\_spread*- how does corona virus spread  
  - how does the virus spread  
    
  *## intent:corona\_food\_spread*- Does corona spread from food  
  - how will corona spread from food  
    
  *## intent:warm\_weather*- will warm weather stop the spread  
  - will it stop with warm weather  
    
  *## intent: high\_risk*- who is at a higher risk of infection  
  - who will be in most danger after affecting with this infection?  
    
  *## intent: symptoms*- What are the symptoms of it?  
  - what are its symptoms  
  - how to verify that i am suffering from coronavirus  
  - any specific symptoms  
    
  *## intent: concern*- Is it a matter of concern?  
  - Is it a serious issue?  
  - Is this a major cause?  
    
  *## intent: prevention*- how to prevent ourselves from getting in contact with corona virus?  
  - What are the steps to prevent ourselves from getting in contact with coronavirus?  
  - Can you tell me how should I protect myself?  
    
  *## intent: Vaccination*- Is there any treatment available for it  
  - What about the vaccination  
  - what medicine should I use If I got infected with coronavirus?  
    
  *## intent: incubation*- What is the incubation period for COVID-19?  
  - What is the incubation period for it?  
  - Is there any incubation period for it?

This file is used to create all the intents and their sample utterances for conversation.

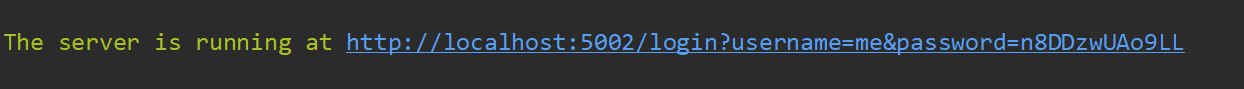
* Open the ‘domain.yml’ file and put the following content:
* session\_config:  
   session\_expiration\_time: 60  
   carry\_over\_slots\_to\_new\_session: true  
  intents:  
  - greet  
  - goodbye  
  - affirm  
  - deny  
  - bot\_challenge  
  - what\_is\_corona  
  - symptoms  
  - concern  
  - prevention  
  - Vaccination  
  - incubation  
  - thanks  
  - corona\_spread  
  - corona\_food\_spread  
  - warm\_weather  
  - high\_risk  
  - my\_name\_is  
  - worldcases  
  - mobile\_number  
  - email\_addr  
  - pin  
  - world\_data  
  - demo  
  - info  
  - state\_covid  
    
  entities:  
  - NAME  
  - NUM  
  - EMAIL  
  - ZIP  
  - location  
  - world  
  - states  
  slots:  
   EMAIL:  
   type: unfeaturized  
   auto\_fill: false  
   NAME:  
   type: unfeaturized  
   auto\_fill: false  
   NUM:  
   type: unfeaturized  
   ZIP:  
   type: unfeaturized  
   location:  
   type: text  
   world:  
   type: text  
  responses:  
   utter\_greet:  
   - text: Hey! How may I assist you?  
   utter\_corona:  
   - text: "COVID-19 is a new illness that can affect your lungs and airways. It is  
   caused by a virus called coronavirus. COVID-19 is the disease caused by the  
   new coronavirus that emerged in China in December 2019."  
   utter\_symptoms:  
   - text: COVID-19 symptoms includes Cough, Fever and Shortness of breath.  
   utter\_concern:  
   - text: "yes, from the recent information indicates COVID-19 may be passed from person  
   to person. Community spread is being seen, also. Community spread means people  
   have been infected with the virus in a particular area, including some people  
   who are not sure how or where they became infected. COVID-19 has been detected  
   in people throughout China and in over 100 other countries, including the United  
   States."  
   utter\_prevent:  
   - text: 'Wash your hands frequently and thoroughly for at least 20 seconds, Use  
   alcohol-based hand sanitizer if soap and water aren’t available. Cover coughs  
   and sneezes with a tissue, then throw the tissue in the trash. Avoid touching  
   your eyes, nose or mouth with unwashed hands. Stay home when you are sick. Clean  
   and disinfect surfaces and objects people frequently touch. Cover your face  
   with a mask to prevent yourself from getting in contact with coronavirus. '  
   utter\_Vaccination:  
   - text: Sorry, there is no vaccine.  
   utter\_incubation:  
   - text: It appears that symptoms are showing up in people within 14 days of exposure  
   to the virus.  
   utter\_thanks:  
   - text: You're welcome and Stay safe.  
   utter\_corona\_spread:  
   - text: "This virus was first detected in Wuhan City, Hubei Province, China. The\  
   \ first infections were linked to a live animal market, but the virus is now\  
   \ spreading from person-to-person. It’s important to note that person-to-person\  
   \ spread can happen on a continuum. Some viruses are highly contagious (like\  
   \ measles), while other viruses are less so. The virus that causes COVID-19\  
   \ is spreading from person-to-person. Someone who is actively sick with COVID-19\  
   \ can spread the illness to others. That is why we recommend that these patients\  
   \ be isolated either in the hospital or at home (depending on how sick they\  
   \ are) until they are better and no longer pose a risk of infecting others.\n\  
   How long someone is actively sick can vary so the decision on when to release\  
   \ someone from isolation is made on a case-by-case basis in consultation with\  
   \ doctors, infection prevention and control experts, and public health officials\  
   \ and involves considering specifics of each situation including disease severity,\  
   \ illness signs and symptoms, and results of laboratory testing for that patient.\n\  
   The virus that causes COVID-19 seems to be spreading easily and sustainably\  
   \ in the community (“community spread”) in some affected geographic areas. Community\  
   \ spread means people have been infected with the virus in an area, including\  
   \ some who are not sure how or where they became infected."  
   utter\_corona\_food\_spread:  
   - text: "Coronaviruses are generally thought to be spread from person-to-person through  
   respiratory droplets. Currently there is no evidence to support transmission  
   of COVID-19 associated with food. Before preparing or eating food it is important  
   to always wash your hands with soap and water for 20 seconds for general food  
   safety. Throughout the day wash your hands after blowing your nose, coughing  
   or sneezing, or going to the bathroom."  
   utter\_warm\_weather:  
   - text: It is not yet known whether weather and temperature impact the spread of  
   COVID-19. Some other viruses, like the common cold and flu, spread more during  
   cold weather months but that does not mean it is impossible to become sick with  
   these viruses during other months. At this time, it is not known whether the  
   spread of COVID-19 will decrease when weather becomes warmer. There is much  
   more to learn about the transmissibility, severity, and other features associated  
   with COVID-19 and investigations are ongoing.  
   utter\_high\_risk:  
   - text: Older adults and people of any age who have serious underlying medical conditions  
   may be at higher risk for more serious complications from COVID-19. These people  
   who may be at higher risk of getting very sick from this illness, includes;  
   Older adults, People who have serious underlying medical conditions like...  
   Heart disease, Diabetes, Lung disease.  
   utter\_goodbye:  
   - text: Bye  
   utter\_iamabot:  
   - text: I am a covidbot, which tell about covid updates.

This file is used to configure the bot responses.

* Open the ‘stories.md’ file from the data folder and put the following content:
* *## world cases*\* worldcases{"location": "London"}  
   - slot{"location": "London"}  
   - action\_covid\_country  
     
  *## story 1*\* worldcases  
   - utter\_ask\_location  
  \* worldcases{"location": "italy"}  
   - slot{"location": "italy"}  
   - action\_covid\_country  
     
  *## email send*\* email\_addr{"EMAIL":"support@ineuron.ai"}  
   - slot{"EMAIL":"support@ineuron.ai"}  
   - action\_send\_email  
     
  *## world cases*\* world\_data{"world": "global"}  
   - slot{"world": "global"}  
   - utter\_worldmap  
   - action\_covid\_world  
   - utter\_tell\_email  
  \* email\_addr{"EMAIL":"support@ineuron.ai"}  
   - slot{"EMAIL":"support@ineuron.ai"}  
   - action\_send\_email  
    
  *## demographic*\* demo  
   - utter\_demograhic  
   - utter\_ask\_state  
  \* state\_covid{"states": "delhi"}  
   - slot{"states": "delhi"}  
   - action\_covid\_state  
    
  *## mograp*\* demo  
   - utter\_demograhic  
   - utter\_ask\_state  
  \* state\_covid{"states": "karnataka"}  
   - slot{"states": "karnataka"}  
   - action\_covid\_state

This file is used to create the conversation flows.

* After all this, just enter the command ‘**rasa train’** to train the model with new conversation elements.
* After the training is completed, enter the command ‘**rasa x**’ to test your chatbot in the web UI. You’ll see :



* Copy this URL in your web browser and you’ll see the web UI for your chatbot:

There you can talk to your ui bot.

Adding the Custom image with Links to the Rasa Chatbot

**Introduction**

Images are basically the graphical source of information and it can consist of many data/information in a single image, for now let’s consider our example from the where the chatbot want to show the demoghraphic of covid 19 cases. Now let’s understand how does that happens.

**Working**

Here what happens is we want to add image to the Rasa chatbot but the rasa chatbot doesn’t directly support the image and it won’t take image as an input. So to provide the image to the rasa chatbot we have parameter that is “image” that we need to set in the domain file with the link to the image either from the server or from the local drive and that is written with the utter\_ response for the chatbot. Rasa core take care of this link and redirect to the link and display the image from the link on the chatbot.

Here, one thing is very clear that we have to make changes to the domain.yml file as we are making changes to the rasa chatbot responses. So the domain.yml file that we have now is:

utter\_demograhic:  
- image: https://www.statista.com/graphic/1/1103458/india-novel-coronavirus-covid-19-cases-by-state.jpg  
 text: Total cases in India by demographic way.

# Form action with your Rasa Chat-bot

**What are Forms?**

One of the most common conversation patterns is to collect few pieces of information from a user in order to do something (like booking a restaurant, calling an API, searching the database, etc.). This is also called **slot filling**. If you want to collect multiple pieces of information in a row, we recommended that you to create a FormAction. This FormAction is the single action which have the logic to loop over the required slots and ask the user for the information until all the slots are filled. So, this is the most important and the required logic in the chatbot to save many lines of code and also to save efforts for the same process.

So, how do we do it to with rasa chatbot to make it effective, reliable with less efforts. For the form action we need to make the updation in the project files, which are, config.yml, actions.py, stories.md and domain.yml.

Let’s start one by one and understand the basic functionality of each of them:

**Setup Configuration**

Firstly, we need to include the FormPolicy in to the configuration file to make the whole process function properly. In this way,

policies:   
 - name: FormPolicy

The FormPolicy is extremely simple and just always predicts the form action. This comes into action when the form action is called for the first time.

# **Adding the stories**

*## user info*\* info  
 - form\_info  
 - form{"name": "form\_info"}  
 - form{"name": null}  
\* affirm  
 - utter\_iamabot

# So, above is the happy path for calling the FormAction. In this story “**info**” is the user intent to which the bot will redirect to the FormAction which is “**form\_info**”. Here, **“form{“name”: “form\_info”}”**is used to activate the form and **“form{“name”: null}”**is used to deactivate the form again. Once the FormAction is activated, the boty can execute any kind of action outside the form. The above “user info” means that once the FormAction is active it can fo outside of the form and perform any action until all requested slots are filled without interruption.

**Updating the actions file**  
In the above story you have seen we have added the happy path as per the form action. Now we will learn what actually is happening when the form action is called. To to this you first need to define three methods:

* **name**: the name of this action (form\_info in our case)
* **required\_slots**: a list of slots that need to be filled for the submit method to work.
* **submit**: what to do at the end of the form, when all the slots have been filled.

Here, what happens is firstly when the FormAction is called for the first time, then form gets activated and FormPolicy jumps in. The FormPolicy is extremely simple and just always predicts the form action. To identify which form has been called the first method is to be added to the to the actions.py file, i.e.,

def name(self) -> Text:  
 *"""Unique identifier of the form"""* return "form\_info"

# When the bot identifes which FormAction is to be called then it moves to the next static method which is requested\_slots where all the slots are set in the order to be called when requested and here with respect to these requested slot names an utter\_ask\_{slot\_name} is called from the domain.yml file where all the bot response are set. The static method that is to added to the action file is :

@staticmethod  
def required\_slots(tracker: Tracker) -> List[Text]:  
 *"""A list of required slots that the form has to fill"""* return ["NAME", "EMAIL", "NUMBER", "ZIP"]

Here you can see we are returning the list for the static method and this list specifies the order of the slots to be requested to fill values for each of them. So, it simply means firstly utter\_ask\_NAME will be called and vice versa.  
Once all the slots are filled, submit() method is called where we can use the collected information in whatever way we want use it. For example, asking for booking confirmation with your details. This method will be added in actions.py file,

def submit(  
 self,  
 dispatcher: CollectingDispatcher,  
 tracker: Tracker,  
 domain: Dict[Text, Any],  
) -> List[Dict]:  
 *"""Define what the form has to do  
 after all required slots are filled"""* # utter submit template  
 dispatcher.utter\_message(template="utter\_submit",  
 name=tracker.get\_slot('NAME'),  
 number=tracker.get\_slot('NUMBER'),  
 email=tracker.get\_slot('EMAIL'),  
 zip=tracker.get\_slot('ZIP'))  
 return []

**Slot Mapping**  
If you do not define slot mappings in the actions file, slots will be only filled by the entities with the same name as the slot that are picked up from the user input. Slots can be picked with the single entity but FormAction supports inputs like yes/no questions and free-text input. The slot\_mappings method defines how to extract slot values from user responses. Add the below method to your action file to extract the required information from the user response.

def slot\_mappings(self) -> Dict[Text, Union[Dict, List[Dict]]]:  
 *"""A dictionary to map required slots to  
 - an extracted entity  
 - intent: value pairs  
 - a whole message  
 or a list of them, where a first match will be picked"""* return {  
 "name": [self.from\_entity(entity="NAME", intent='my\_name\_is')],  
 "number": [self.from\_entity(entity="NUMBER", intent='mobile\_number')],  
 "email": [self.from\_entity(entity="EMAIL", intent='email\_addr')],  
 "zip": [self.from\_entity(entity="ZIP", intent='pin')],  
 }

**Updating Domain file**

Now to link the core and nlu to the action file add the following line of code to your existing chatbot in domain.yml file

forms:  
- form\_info

slots:  
 EMAIL:  
 type: unfeaturized  
 auto\_fill: false  
 NAME:  
 type: unfeaturized  
 auto\_fill: false  
 NUM:  
 type: unfeaturized  
 ZIP:  
 type: unfeaturized

utter\_ask\_NUM:  
- text: What's your mobile number?  
utter\_ask\_EMAIL:  
- text: What's your email address?  
utter\_ask\_ZIP:  
- text: what is your zipcode where you stay?  
utter\_wrong\_NUMBER:  
- text: Mobile Number should be a positive integer and 10 digit number, please try  
 again..  
utter\_wrong\_ZIP:  
- text: Zipcode should be a positive integer and 6 digit number, please try again..  
utter\_submit:  
- text: "Your details.\nYour First name : {name}\nemail : {email}\nZipcode : {zip} \nMobile Number : {number}"

# Fallback Actions | Rasa

Chatbot is what we have been working with and we have already build different chatbots as well as the voice bots in our previous blogs, but there we may have faced one problem and that is the major concern while building a chatbot or voice bot. So while interacting with the chatbot or the voice bot if you say anything wrong or the text is out of the context. So, in that case either it will reply with the wrong message or the outpur message is not up to the mark. Also, if the reply is wrong it wont ask you to correct it instead it will move to the next step as per the story. So, here we need a policy that can handle such situation and also instead of avoiding the problem and replying with the wrong messages, it will ask you to rephase the input message and to follow the story from the same state.

**What is Fallback Policy?**

A policies that are used to handle the fallback cases by adding either the FallbackPolicy or the TwoStageFallbackPolicy with respect to the confidence value of the nlu and the core and to avoid the situation of breaking the flow of the conversation.

The FallbackPolicy has one fallback action, which will be executed if the confidence of intent recognition is below the nlu\_threshold or if none of the dialogue policies predict an action with confidence above the core\_threshold.

Adding the fallback policy to your rasa project is very easy and very interesting, you simply have to add the following lines to your config.yml, inside the policies:

- name: FallbackPolicy  
 nlu\_threshold: 0.4  
 core\_threshold: 0.4  
 fallback\_action\_name: "action\_custom\_fallback"

In the above fallback policy you can accordingly adjust the values of nlu\_threshold and core\_threshold as per your project requirement for fallback cases. Here action\_default\_fallback is the default fallback action in rasa core which sends the utter\_default response to the user. So in this fallback situation to get the response you have to mention utter\_default in the domain.yml to get the response eg: “Sorry, Didn’t understand. Please say again”. Also with this response it will be reverted back to the state of the conversation before the user message that caused the fallback, so that it will not influence the future actions.

This process will executes the fallback action and goes back to the previous state of the dialogue.

Also, add the template in the domain.yml file as,

responses:

utter\_custom:  
- text: Sorry, I didn't get that please try again...

utter\_default:  
- text: Sorry, I didn't understand. Try again...

In action.py

class ActionCustomFallback(Action):  
  
 def name(self) -> Text:  
 return "action\_custom\_fallback"  
  
 def run(self, dispatcher: CollectingDispatcher,  
 tracker: Tracker,  
 domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:  
  
 dispatcher.utter\_template("utter\_custom", tracker)  
  
 return [UserUtteranceReverted()]

The most import thing to keep in mind while creating a custom fallback action you have to return the UserUtteranceReverted in the action file, to avoid the inaccurate next predictions of the bot, as it is very likely that the fallback action is not present in your stories.

Ading database in Rasa:

All conversations are stored within a tracker store. Rasa Open Source provides implementations for different store types out of the box. If you want to use another store, you can also build a custom tracker store by extending the TrackerStore class.

## [SQLTrackerStore](https://rasa.com/docs/rasa/api/tracker-stores/#id5)

SQLTrackerStore can be used to store the conversation history in an SQL database. Storing your trackers this way allows you to query the event database by sender\_id, timestamp, action name, intent name and typename.

To set up Rasa Open Source with SQL the following steps are required:

1. Add required configuration to your endpoints.yml:
2. tracker\_store:  
    type: SQL  
    dialect: "sqlite" *# the dialect used to interact with the db* url: "sqlite:///./covid\_db.db"  
    db: "covid\_db.db" *# path to your db* username: *# username used for authentication* password: *# password used for authentication* query: *# optional dictionary to be added as a query string to the connection URL* driver: my-driver

2 . To start the Rasa server using your SQL backend, add the --endpoints flag, e.g.:

rasa run -m models --endpoints endpoints.yml

**Telegram Integration:**

* + Download ngrok from<https://ngrok.com/download>
  + After extracting the zip file, open the ngrok file and run it.
  + In ngrok, enter the command ‘**ngrok http 5005** ’:
* Then go to telegram and create your own bot using Botfather:

1. Open the telegram app and search for botfather(it is an inbuilt bot used to create other bots)
2. Start a conversation with botfather and enter /newbot to create a newbot.
3. Give a name to your bot
4. Give a username to your bot, which must end in \_bot.This generates an access token.

* Open ‘credentials.yml’ and enter:

telegram:

access\_token: "obtained from telegram"

verify: "your bot username"

webhook\_url: "https://<ngrokurl>/webhooks/telegram/webhook"

* Go to terminal and enter the command ‘rasa run’
* Open one more terminal and run the command ‘rasa run actions’
* Now, you can chat with your bot from Telegram.

**References:**

1. Rasa Official documentation <https://rasa.com/docs/rasa/user-guide/installation/>
2. CDC Corona FAQ.