BITS Pilani

CLOUD APP– BITSY–THE CHAT BOT

Cloud Computing (CCZG527)

Assignment - 1

Submitted By:

PRASHANT SINGH- 2023MT03125

JALAMANCHILI RAMA SURYAM- 2023MT03101

KAKANI VARSHITHA- 2023MT03002

SRUTHI KRISHNAMURTHY- 2023MT03003

Submitted To:

PROF. ADITYA GOEL

INCLUDEPICTURE "https://upload.wikimedia.org/wikipedia/en/thumb/d/d3/BITS_Pilani-Logo.svg/1200px-BITS_Pilani-Logo.svg.png" * MERGEFORMATINET

Dec 02, 2023

Executive Summary

Using chatbots, you can immediately engage website visitors with personalized messaging. Depending on your target market's demographics, where they came from, the content they are interacting with, and where they are in the purchasing process, you may create customized chatbots for each page of your website or audience.

BITSY is a system which BITS WILP Students can use to choose the subjects they want to study and the teachers they want to study the subjects from. The chatbot provides the answers to the relevant questions.

OBJECTIVE

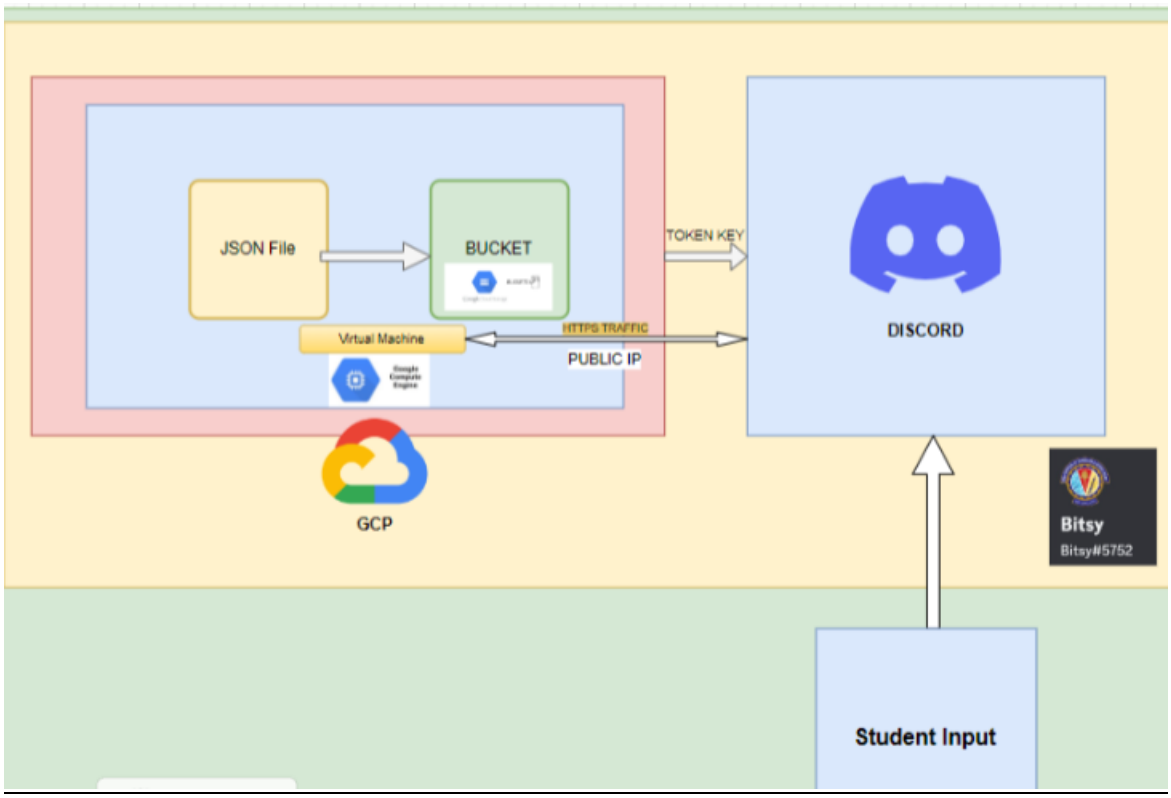
Software programs called chatbots mimic human speech. In order to simulate real-world interactions and respond to consumer inquiries, they adhere to a set of pre-designed rules.

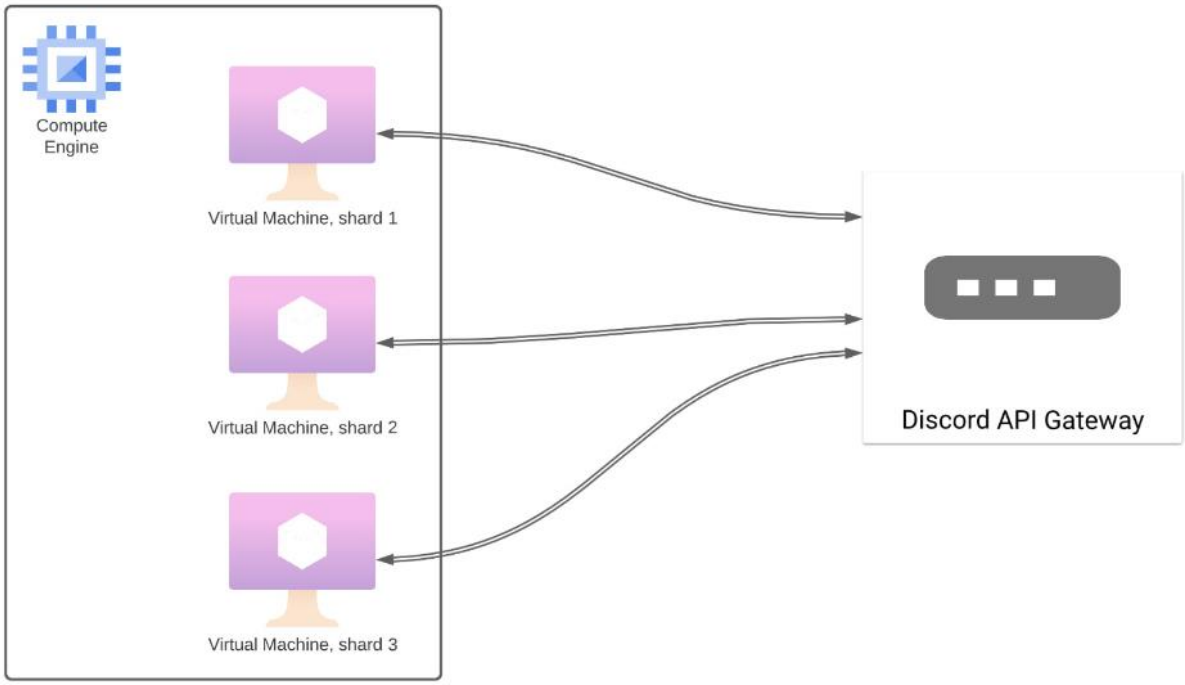
Additionally, chatbots that employ artificial intelligence (AI) and natural language processing (NLP) are able to interpret these exchanges almost as well as a human.

Using chatbots, you can immediately engage website visitors with personalized messaging.

Depending on your target market's demographics, where they came from, the content they are interacting with, and where they are in the purchasing process, you may create customized chatbots for each page of your website or audience.

ARCHITECTURE OF THE APPLICATION

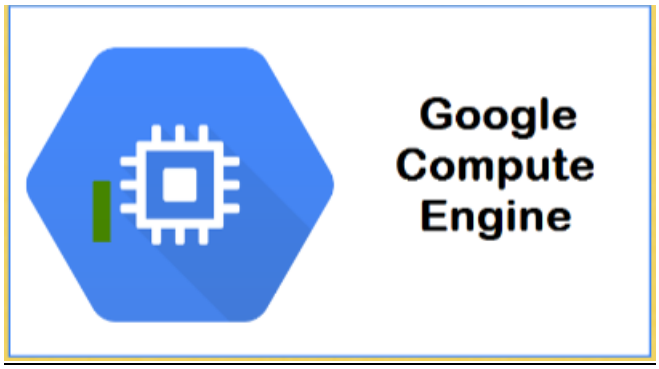


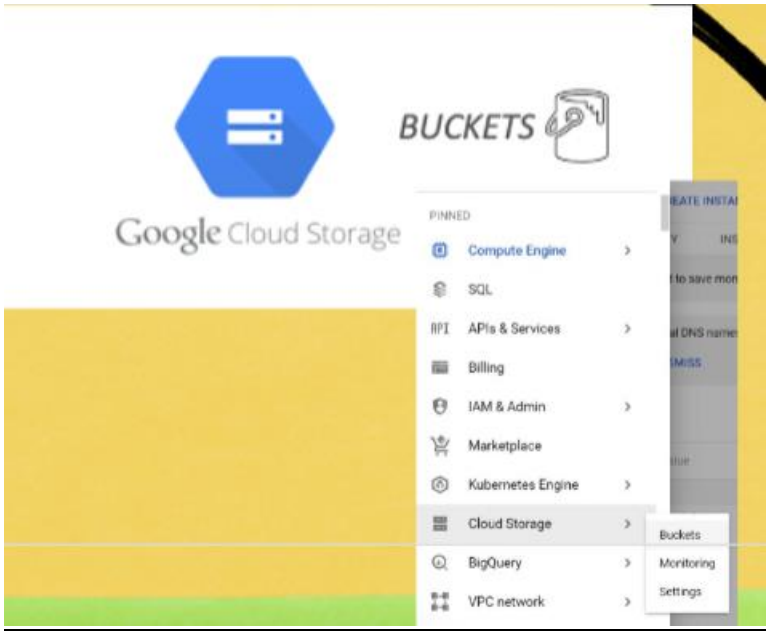


CLOUD PLATFORM USAGE

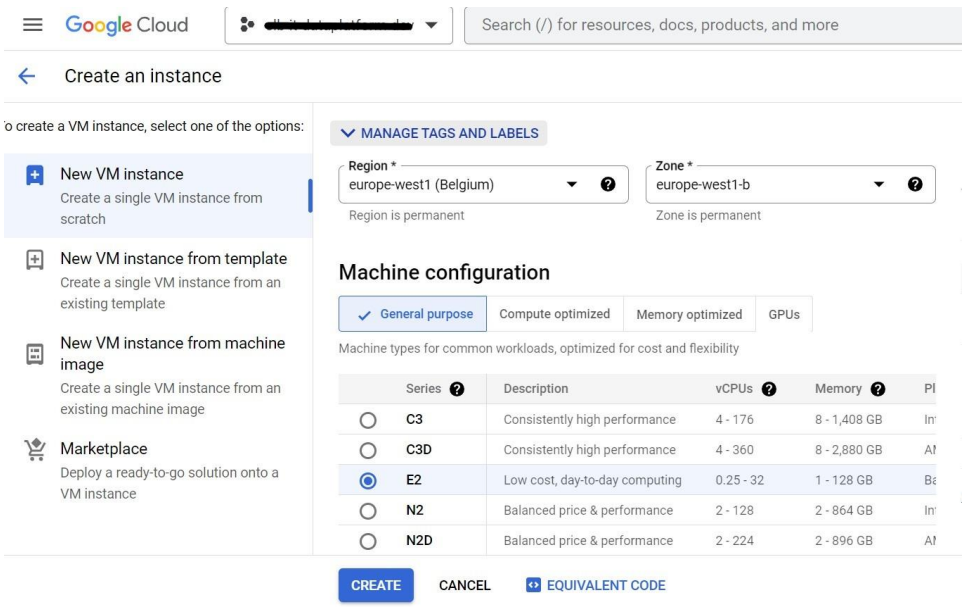
We are using GCP cloud here to host our application which will then connect to DISCORD the messaging app via the API call.

We used the below services in GCP To host with additional configuration





Additional configurations needed for our Application to Run as below to setup EC instance:



Machine type

Choose a machine type with preset amounts of vCPUs and memory that suit most workloads. Or, you can create a custom machine for your workload's particular needs. Learn more

PRESET | CUSTOM

e2-standard-8 (8 vCPU, 4 core, 32 GB memory)

vCPU	Memory
8 (4 cores)	32 GB

ADVANCED CONFIGURATIONS

Availability policies

VM provisioning model
Standard

Choose "Spot" to get a discounted, preemptible VM. Otherwise, stick to "Standard". Learn more

Boot disk

Type	New standard persistent disk
Size	10 GB
License type	Free
Image	Debian GNU/Linux 11 (bullseye)

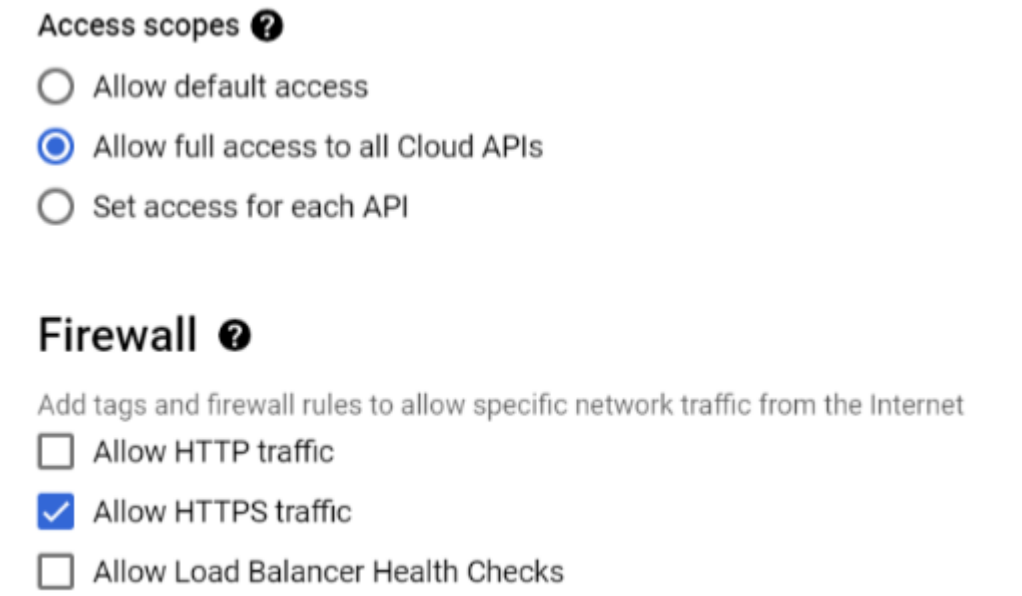
CHANGE

Identity and API access

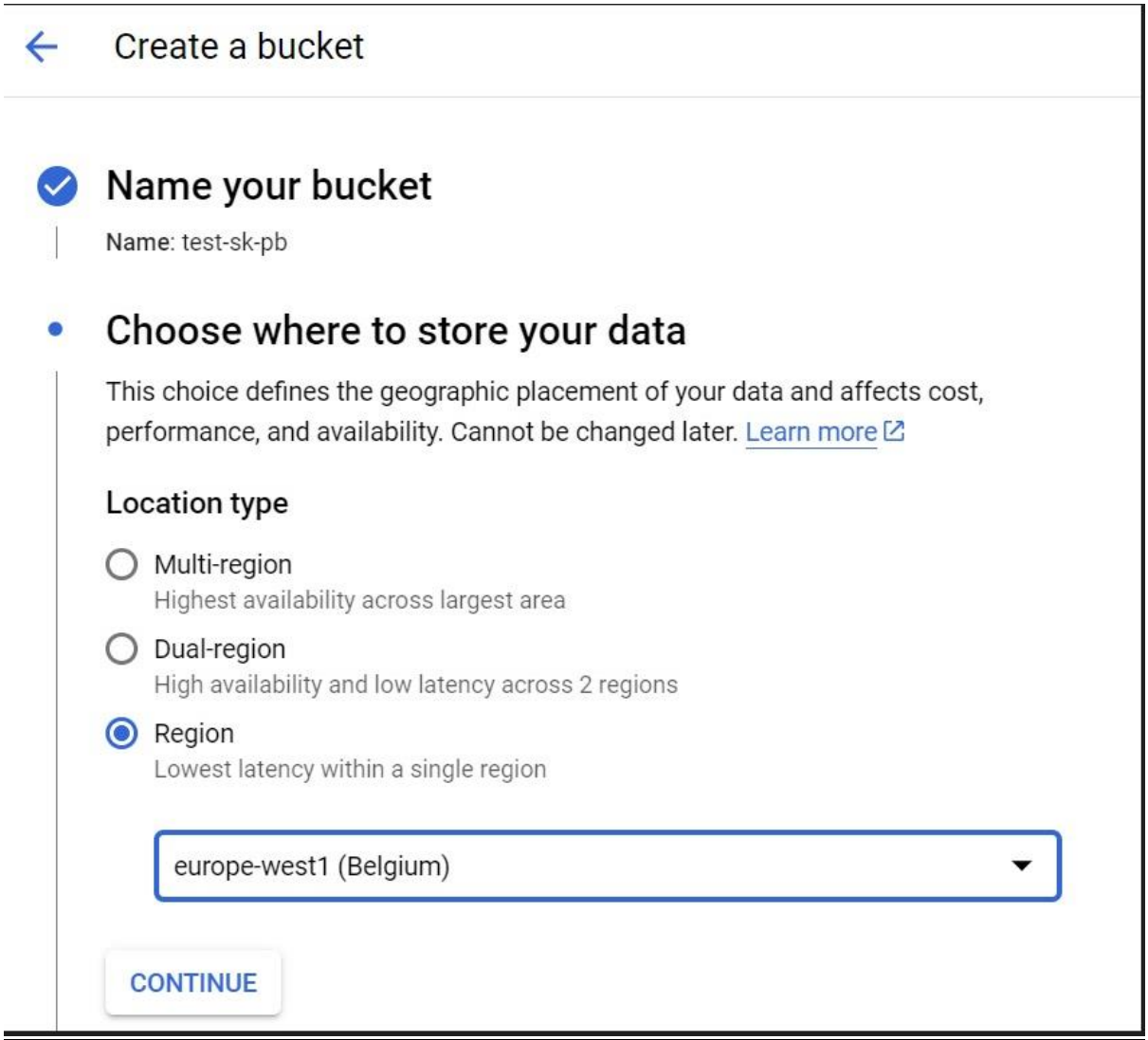
Service accounts

Service account
Compute Engine default service account

Requires the Service Account User role (roles/iam.serviceAccountUser) to be set for users who want to access VMs with this service account. Learn more



Additional configurations needed for our Application to Run as below to storage setup:



Choose a storage class for your data

A storage class sets costs for storage, retrieval, and operations, with minimal differences in uptime. Choose if you want objects to be managed automatically or specify a default storage class based on how long you plan to store your data and your workload or use case. Learn more

- Autoclass
  Automatically transitions each object to Standard or Nearline class based on object-level activity, to optimize for cost and latency. Recommended if usage frequency may be unpredictable. Can be changed to a default class at any time. Pricing details
- Set a default class
  Applies to all objects in your bucket unless you manually modify the class per object or set object lifecycle rules. Best when your usage is highly predictable. Can't be changed to Autoclass once the bucket is created.
  - Standard
    Best for short-term storage and frequently accessed data
  - Nearline
    Best for backups and data accessed less than once a month
  - Coldline
    Best for disaster recovery and data accessed less than once a quarter
  - Archive
    Best for long-term digital preservation of data accessed less than once a year

Choose how to control access to objects

Prevent public access

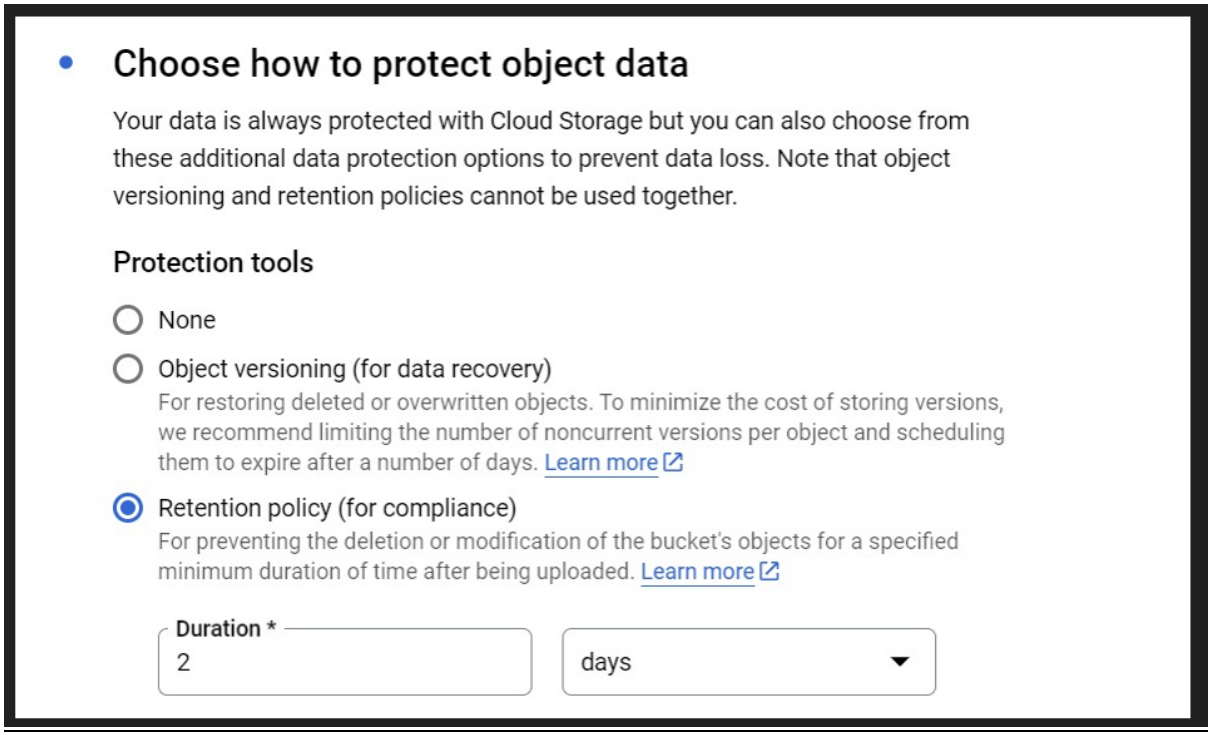
Restrict data from being publicly accessible via the internet. Will prevent this bucket from being used for web hosting. Learn more

- [x] Enforce public access prevention on this bucket

Access control

- Uniform
  Ensure uniform access to all objects in the bucket by using only bucket-level permissions (IAM). This option becomes permanent after 90 days. Learn more
- Fine-grained
  Specify access to individual objects by using object-level permissions (ACLs) in addition to your bucket-level permissions (IAM). Learn more

CONTINUE


Choose how to protect object data
Your data is always protected with Cloud Storage but you can also choose from these additional data protection options to prevent data loss. Note that object versioning and retention policies cannot be used together.
Protection tools
None
Object versioning (for data recovery)
For restoring deleted or overwritten objects. To minimize the cost of storing versions, we recommend limiting the number of noncurrent versions per object and scheduling them to expire after a number of days. Learn more
Retention policy (for compliance)
For preventing the deletion or modification of the bucket's objects for a specified minimum duration of time after being uploaded. Learn more
Duration *
2
days

CODE SNIPPET DEPLOYED ON THE COMPUTE ENGINE

```
import discord
import key
from neuralintents import GenericAssistant
import nltk
import gcsfs
import json



gcs_file_system = gcsfs.GCSFileSystem(project="gcp_project_name")
gcs_json_path = "gs://test-sk-pb/intents-bits.json"
with gcs_file_system.open(gcs_json_path) as f:
  json_dict = json.load(f)

client = discord.Client()
@client.event
async def on_ready():
    print("connected")


@client.event
async def on_message(message):
    if message.author == client.user:
        return
    if message.content.startswith("!bitsy"):
        responce = chatbot.request(message.content[9:])
        await message.channel.send(responce)




if __name__=="__main__":
    nltk.download("omw-1.4")
    chatbot = GenericAssistant("intents-bits.json")
    chatbot.train_model()
    chatbot.save_model()
    print("model done training and saving")



    client.run(key.TOKEN)
~
```

STEPS TO EXECUTE THE BOT:

1. Start the VM on GCP and Let the BOT train as we are calling the train_model() method from the from bucket data stored in Json.
2. Launch the DISCORD server and connect to the bot using TOKEN.
3. Once connection established and BOT is running and completed learning, BITS BOT On DISCORD will become online
4. Ask the questions as shown in the output below.

BOT LEARNING FROM DATA AND GETTING READY FOR INTERACTIVE SESSION

```
Epoch 191/200
56/56 [==============================] - 0s 1ms/step - loss: 0.0969 - accuracy: 0.9713
Epoch 192/200
56/56 [==============================] - 0s 1ms/step - loss: 0.0838 - accuracy: 0.9713
Epoch 193/200
56/56 [==============================] - 0s 1ms/step - loss: 0.0794 - accuracy: 0.9642
Epoch 194/200
56/56 [==============================] - 0s 1ms/step - loss: 0.1299 - accuracy: 0.9642
Epoch 195/200
56/56 [==============================] - 0s 1ms/step - loss: 0.0499 - accuracy: 0.9857
Epoch 196/200
56/56 [==============================] - 0s 1ms/step - loss: 0.1007 - accuracy: 0.9677
Epoch 197/200
56/56 [==============================] - 0s 1ms/step - loss: 0.0508 - accuracy: 0.9892
Epoch 198/200
56/56 [==============================] - 0s 1ms/step - loss: 0.1103 - accuracy: 0.9677
Epoch 199/200
56/56 [==============================] - 0s 1ms/step - loss: 0.0731 - accuracy: 0.9677
Epoch 200/200
56/56 [==============================] - 0s 1ms/step - loss: 0.0441 - accuracy: 0.9892
model done training and saving
connected
```

OUTPUT: