Membuat File Aplikasi Google Assistant (GVA) terintegrasi Mycroft Precise pada Ubuntu 16.04 LTS

Note : Metode pemasangan berlaku umum sehingga dapat diterapkan pada OS  
Ubuntu berbagai arsitektur termasuk Ubuntu Arm dan Ubuntu Aarch64

1. Pastikan bahwa terminal sedang berjalan dalam Virtual Environment Python:



1. Buat folder baru:

|  |
| --- |
| $ mkdir google-assistant $ cd google-assistant |

1. Pasang paket library Python terkait google-assistant:

|  |
| --- |
| $ python -m pip install --upgrade google-assistant-sdk[samples] google-api-core |

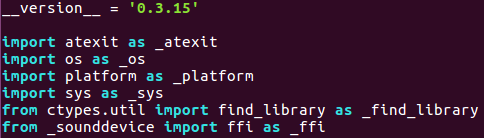
1. Modifikasi file sounddevice pada paket library Python sounddevice  
   (file ini perlu diubah untuk menghindari penggunaan library PortAudio yang tidak sesuai):

|  |
| --- |
| $ nano ../env/lib64/python3.5/site-packages/sounddevice.py |

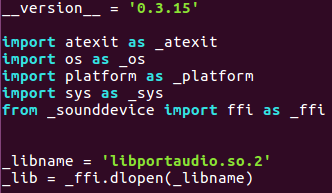
Atau:

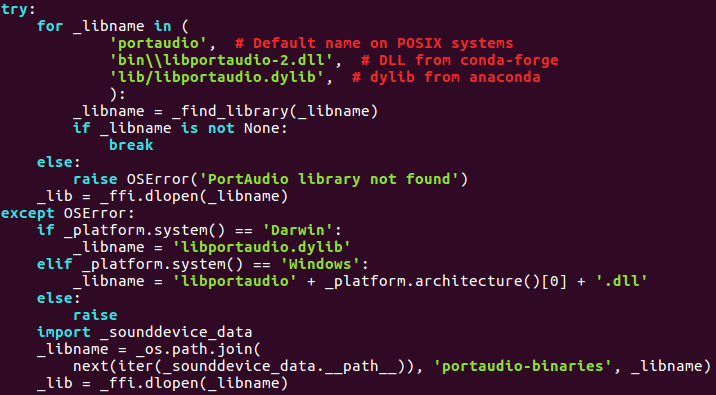
|  |
| --- |
| $ nano ../env/lib/python3.5/site-packages/sounddevice.py |

Perhatikan bahwa sistem 32bit hanya memiliki folder env/lib dan tidak memiliki folder env/lib64.

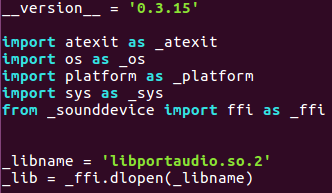


Untuk modifikasi pertama, hapus find\_library dari file:





Untuk modifikasi kedua, hapus proses deteksi library PortAudio:



1. Buat script Python (terlampir):

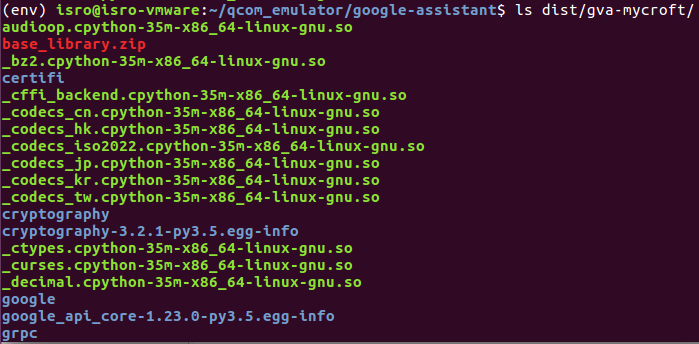
|  |
| --- |
| $ nano gva-mycroft.py |

1. Buat file spec pyinstaller (terlampir):

|  |
| --- |
| $ nano gva-mycroft.spec |

1. Build aplikasi Google Assistant terintegrasi Mycroft Precise:

|  |
| --- |
| $ pyinstaller --clean -y gva-mycroft.spec |



Perhatikan bahwa hasil build tersimpan pada folder gva-mycroft di dalam folder dist. Salin satu folder gva-mycroft secara utuh untuk dapat menjalankan aplikasi gva-mycroft di dalam folder tersebut.

Script Python Aplikasi Google Assistant terintegrasi Mycroft Precise

|  |
| --- |
| from precise\_runner import PreciseEngine, PreciseRunner  from subprocess import call  import json  import logging  import os  import os.path  import pathlib2 as pathlib  import sys  import uuid  import grpc  import google.auth.transport.grpc  import google.auth.transport.requests  import google.oauth2.credentials  from google.assistant.embedded.v1alpha2 import embedded\_assistant\_pb2  from google.assistant.embedded.v1alpha2 import embedded\_assistant\_pb2\_grpc  from googlesamples.assistant.grpc import assistant\_helpers  from googlesamples.assistant.grpc import audio\_helpers  ASSISTANT\_API\_ENDPOINT = 'embeddedassistant.googleapis.com'  END\_OF\_UTTERANCE = embedded\_assistant\_pb2.AssistResponse.END\_OF\_UTTERANCE  DIALOG\_FOLLOW\_ON = embedded\_assistant\_pb2.DialogStateOut.DIALOG\_FOLLOW\_ON  CLOSE\_MICROPHONE = embedded\_assistant\_pb2.DialogStateOut.CLOSE\_MICROPHONE  PLAYING = embedded\_assistant\_pb2.ScreenOutConfig.PLAYING  DEFAULT\_GRPC\_DEADLINE = 60 \* 3 + 5  waiting = 1  class SampleAssistant(object):  """Sample Assistant that supports conversations and device actions.  Args:  device\_model\_id: identifier of the device model.  device\_id: identifier of the registered device instance.  conversation\_stream(ConversationStream): audio stream  for recording query and playing back assistant answer.  channel: authorized gRPC channel for connection to the  Google Assistant API.  deadline\_sec: gRPC deadline in seconds for Google Assistant API call.  """  def \_\_init\_\_(self, language\_code, device\_model\_id, device\_id, conversation\_stream, channel, deadline\_sec):  self.language\_code = language\_code  self.device\_model\_id = device\_model\_id  self.device\_id = device\_id  self.conversation\_stream = conversation\_stream  # Opaque blob provided in AssistResponse that,  # when provided in a follow-up AssistRequest,  # gives the Assistant a context marker within the current state  # of the multi-Assist()-RPC "conversation".  # This value, along with MicrophoneMode, supports a more natural  # "conversation" with the Assistant.  self.conversation\_state = None  # Force reset of first conversation.  self.is\_new\_conversation = True  # Create Google Assistant API gRPC client.  self.assistant = embedded\_assistant\_pb2\_grpc.EmbeddedAssistantStub(channel)  self.deadline = deadline\_sec  def \_\_enter\_\_(self):  return self  def \_\_exit\_\_(self, etype, e, traceback):  if e:  return False  self.conversation\_stream.close()  def is\_grpc\_error\_unavailable(e):  is\_grpc\_error = isinstance(e, grpc.RpcError)  if is\_grpc\_error and (e.code() == grpc.StatusCode.UNAVAILABLE):  logging.error('grpc unavailable error: %s', e)  return True  return False  def assist(self):  """Send a voice request to the Assistant and playback the response.  Returns: True if conversation should continue.  """  continue\_conversation = False  self.conversation\_stream.volume\_percentage = 100  self.conversation\_stream.start\_recording()  logging.info('Recording audio request.')  call(["adk-message-send", "led\_indicate\_direction\_pattern{pattern:1,direction:50}"])  def iter\_log\_assist\_requests():  for c in self.gen\_assist\_requests():  assistant\_helpers.log\_assist\_request\_without\_audio(c)  yield c  logging.debug('Reached end of AssistRequest iteration.')  # This generator yields AssistResponse proto messages  # received from the gRPC Google Assistant API.  for resp in self.assistant.Assist(iter\_log\_assist\_requests(), self.deadline):  assistant\_helpers.log\_assist\_response\_without\_audio(resp)  if resp.event\_type == END\_OF\_UTTERANCE:  logging.info('End of audio request detected.')  logging.info('Stopping recording.')  call(["adk-message-send", "led\_start\_pattern{pattern:16}"])  self.conversation\_stream.stop\_recording()  if resp.speech\_results:  logging.info('Transcript of user request: "%s".', ' '.join(r.transcript for r in resp.speech\_results))  if len(resp.audio\_out.audio\_data) > 0:  if not self.conversation\_stream.playing:  self.conversation\_stream.stop\_recording()  self.conversation\_stream.start\_playback()  logging.info('Playing assistant response.')  call(["adk-message-send", "led\_start\_pattern{pattern:2}"])  self.conversation\_stream.write(resp.audio\_out.audio\_data)  if resp.dialog\_state\_out.conversation\_state:  conversation\_state = resp.dialog\_state\_out.conversation\_state  logging.debug('Updating conversation state.')  self.conversation\_state = conversation\_state  if resp.dialog\_state\_out.volume\_percentage != 0:  volume\_percentage = resp.dialog\_state\_out.volume\_percentage  logging.info('Setting volume to %s%%', volume\_percentage)  self.conversation\_stream.volume\_percentage = volume\_percentage  if resp.dialog\_state\_out.microphone\_mode == DIALOG\_FOLLOW\_ON:  continue\_conversation = True  logging.info('Expecting follow-on query from user.')  elif resp.dialog\_state\_out.microphone\_mode == CLOSE\_MICROPHONE:  continue\_conversation = False  logging.info('Finished playing assistant response.')  call(["adk-message-send", "led\_indicate\_direction\_pattern{pattern:17,direction:0}"])  self.conversation\_stream.stop\_playback()  return continue\_conversation  def gen\_assist\_requests(self):  """Yields: AssistRequest messages to send to the API."""  config = embedded\_assistant\_pb2.AssistConfig(  audio\_in\_config=embedded\_assistant\_pb2.AudioInConfig(  encoding='LINEAR16',  sample\_rate\_hertz=self.conversation\_stream.sample\_rate,  ),  audio\_out\_config=embedded\_assistant\_pb2.AudioOutConfig(  encoding='LINEAR16',  sample\_rate\_hertz=self.conversation\_stream.sample\_rate,  volume\_percentage=self.conversation\_stream.volume\_percentage,  ),  dialog\_state\_in=embedded\_assistant\_pb2.DialogStateIn(  language\_code=self.language\_code,  conversation\_state=self.conversation\_state,  is\_new\_conversation=self.is\_new\_conversation,  ),  device\_config=embedded\_assistant\_pb2.DeviceConfig(  device\_id=self.device\_id,  device\_model\_id=self.device\_model\_id,  )  )  # Continue current conversation with later requests.  self.is\_new\_conversation = False  # The first AssistRequest must contain the AssistConfig  # and no audio data.  yield embedded\_assistant\_pb2.AssistRequest(config=config)  for data in self.conversation\_stream:  # Subsequent requests need audio data, but not config.  yield embedded\_assistant\_pb2.AssistRequest(audio\_in=data)  def on\_act():  global waiting  waiting = 0  def main():  """Samples for the Google Assistant API.  Examples:  Run the sample with microphone input and speaker output:  $ python -m googlesamples.assistant  Run the sample with file input and speaker output:  $ python -m googlesamples.assistant -i <input file>  Run the sample with file input and output:  $ python -m googlesamples.assistant -i <input file> -o <output file>  """  # Google Assistant Setting.  api\_endpoint = ASSISTANT\_API\_ENDPOINT  lang = 'en-US'  grpc\_deadline = DEFAULT\_GRPC\_DEADLINE  # Audio Setting.  audio\_sample\_rate = audio\_helpers.DEFAULT\_AUDIO\_SAMPLE\_RATE  audio\_sample\_width = audio\_helpers.DEFAULT\_AUDIO\_SAMPLE\_WIDTH  audio\_iter\_size = audio\_helpers.DEFAULT\_AUDIO\_ITER\_SIZE  audio\_block\_size = audio\_helpers.DEFAULT\_AUDIO\_DEVICE\_BLOCK\_SIZE  audio\_flush\_size = audio\_helpers.DEFAULT\_AUDIO\_DEVICE\_FLUSH\_SIZE  # Setup logging.  verbose = False  logging.basicConfig(level=logging.DEBUG if verbose else logging.INFO)  # Load OAuth 2.0 credentials.  try:  with open('/data/gva-mycroft.json', 'r') as json\_file:  gva\_config = json.load(json\_file)  project\_id = gva\_config["project\_id"]  device\_model\_id = gva\_config["device\_model\_id"]  device\_id = gva\_config["device\_id"]  credentials = gva\_config["credentials"]  device\_config = gva\_config["device\_config"]  engine\_path = gva\_config["engine\_path"]  model\_path = gva\_config["model\_path"]  trigger\_level = gva\_config["trigger\_level"]  sensitivity = gva\_config["sensitivity"]  except Exception as e:  logging.error("Error loading gva-mycroft.json: %s", e)  sys.exit(-1)  try:  with open(credentials, 'r') as f:  credentials = google.oauth2.credentials.Credentials(token=None, \*\*json.load(f))  http\_request = google.auth.transport.requests.Request()  credentials.refresh(http\_request)  except Exception as e:  logging.error('Error loading credentials: %s', e)  logging.error('Run google-oauthlib-tool to initialize new OAuth 2.0 credentials.')  sys.exit(-1)  # Create an authorized gRPC channel.  grpc\_channel = google.auth.transport.grpc.secure\_authorized\_channel(credentials, http\_request, api\_endpoint)  logging.info('Connecting to %s', api\_endpoint)  # Configure audio source and sink.  audio\_device = None  audio\_source = audio\_device = (  audio\_device or audio\_helpers.SoundDeviceStream(  sample\_rate=audio\_sample\_rate,  sample\_width=audio\_sample\_width,  block\_size=audio\_block\_size,  flush\_size=audio\_flush\_size  )  )  audio\_sink = audio\_device = (  audio\_device or audio\_helpers.SoundDeviceStream(  sample\_rate=audio\_sample\_rate,  sample\_width=audio\_sample\_width,  block\_size=audio\_block\_size,  flush\_size=audio\_flush\_size  )  )  # Create conversation stream with the given audio source and sink.  conversation\_stream = audio\_helpers.ConversationStream(  source=audio\_source,  sink=audio\_sink,  iter\_size=audio\_iter\_size,  sample\_width=audio\_sample\_width,  )  if not device\_id or not device\_model\_id:  try:  with open(device\_config) as f:  device = json.load(f)  device\_id = device['id']  device\_model\_id = device['model\_id']  logging.info("Using device model %s and device id %s", device\_model\_id, device\_id)  except Exception as e:  logging.warning('Device config not found: %s' % e)  logging.info('Registering device')  if not device\_model\_id:  logging.error('Option --device-model-id required when registering a device instance.')  sys.exit(-1)  if not project\_id:  logging.error('Option --project-id required when registering a device instance.')  sys.exit(-1)  device\_base\_url = ('https://%s/v1alpha2/projects/%s/devices' % (api\_endpoint, project\_id))  device\_id = str(uuid.uuid1())  payload = {  'id': device\_id,  'model\_id': device\_model\_id,  'client\_type': 'SDK\_SERVICE'  }  session = google.auth.transport.requests.AuthorizedSession(credentials)  r = session.post(device\_base\_url, data=json.dumps(payload))  if r.status\_code != 200:  logging.error('Failed to register device: %s', r.text)  sys.exit(-1)  logging.info('Device registered: %s', device\_id)  pathlib.Path(os.path.dirname(device\_config)).mkdir(exist\_ok=True)  with open(device\_config, 'w') as f:  json.dump(payload, f)  try:  # initiate precise engine with mycroft model  engine = PreciseEngine(engine\_path, model\_path)  # initiate precise runner that will listen, predict, and detect wakeword  runner = PreciseRunner(engine, on\_activation=on\_act, trigger\_level=trigger\_level, sensitivity=sensitivity)  # start runner  runner.start()  except Exception as e:  logging.error("Wake Word Engine Error: %s", e)  sys.exit(-1)  # keep main thread active until user interrupt  try:  with SampleAssistant(lang, device\_model\_id, device\_id, conversation\_stream, grpc\_channel, grpc\_deadline) as assistant:  wait\_for\_user\_trigger = True  global waiting  call(["adk-message-send", "led\_start\_pattern{pattern:7}"])  while True:  if wait\_for\_user\_trigger:  logging.info("Waiting Wake Word")  while waiting == 1:  pass  continue\_conversation = assistant.assist()  wait\_for\_user\_trigger = not continue\_conversation  waiting = 1  except Exception as e:  runner.stop()  logging.error("Google Assistant Error: %s", e)  sys.exit(-1)  if \_\_name\_\_ == '\_\_main\_\_':  main() |

Perhatikan bahwa '/data/gva-mycroft.json' adalah letak file konfigurasi.

File Spec PyInstaller Aplikasi Google Assistant terintegrasi Mycroft Precise

|  |
| --- |
| # -\*- mode: python ; coding: utf-8 -\*-  block\_cipher = None  from PyInstaller.utils.hooks import collect\_data\_files  grpc\_datas = collect\_data\_files('grpc')  a = Analysis(['gva-mycroft.py'],  pathex=['.'],  binaries=[('/usr/lib/x86\_64-linux-gnu/libxcb.so.1','.')],  datas=grpc\_datas,  hiddenimports=[],  hookspath=[],  runtime\_hooks=[],  excludes=[],  win\_no\_prefer\_redirects=False,  win\_private\_assemblies=False,  cipher=block\_cipher,  noarchive=False)  pyz = PYZ(a.pure, a.zipped\_data,  cipher=block\_cipher)  exe = EXE(pyz,  a.scripts,  [],  exclude\_binaries=True,  name='gva-mycroft',  debug=False,  bootloader\_ignore\_signals=False,  strip=False,  upx=True,  console=True )  coll = COLLECT(exe,  a.binaries,  a.zipfiles,  a.datas,  strip=False,  upx=True,  upx\_exclude=[],  name='gva-mycroft') |

Perhatikan bahwa letak file '/usr/lib/x86\_64-linux-gnu/libxcb.so.1' akan berbeda untuk arsitektur sistem yang berbeda, yaitu:

* Ubuntu Desktop (x86\_64) : '/usr/lib/x86\_64-linux-gnu/libxcb.so.1'
* Ubuntu Arm (ARM 32) : '/usr/lib/arm-linux-gnueabihf/libxcb.so.1'
* Ubuntu Aarch64 (ARM 64) : '/usr/lib/aarch64-linux-gnu/libxcb.so.1'