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:

```
isro@isro-vmware:~/qcom_emulator$ source env/bin/activate
(env) isro@isro-vmware:~/qcom_emulator$
```

2. Buat folder baru:

```
$ mkdir google-assistant
$ cd google-assistant
```

3. Pasang paket library Python terkait google-assistant:

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

4. 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.

```
__version__ = '0.3.15'

import atexit as _atexit
import os as _os
import platform as _platform
import sys as _sys
from ctypes.util import find_library as _find_library
from _sounddevice import ffi as _ffi
```

Untuk modifikasi pertama, hapus find_library dari file:

```
__version__ = '0.3.15'

import atexit as _atexit
import os as _os
import platform as _platform
import sys as _sys
from _sounddevice import ffi as _ffi
```

```
try:
    for _libname in (
            'portaudio',  # Default name on POSIX systems
            'bin\\libportaudio-2.dll',  # DLL from conda-forge
            'lib/libportaudio.dylib',  # dylib from anaconda
            ):
        _libname = _find_library(_libname)
        if _libname is not None:
            break
    else:
        raise OSError('PortAudio library not found')
    _lib = _ffi.dlopen(_libname)
except OSError:
    if _platform.system() == 'Darwin':
        _libname = 'libportaudio.dylib'
    elif _platform.system() == 'Windows':
        _libname = 'libportaudio' + _platform.architecture()[0] + '.dll'
    else:
        raise
    import _sounddevice_data
    _libname = _os.path.join(
        next(iter(_sounddevice_data.__path__)), 'portaudio-binaries', _libname)
    _lib = _ffi.dlopen(_libname)
```

Untuk modifikasi kedua, hapus proses deteksi library PortAudio:

```
_libname = 'libportaudio.so.2'
_lib = _ffi.dlopen(_libname)
```

5. Buat script Python (terlampir):

```
$ nano gva-mycroft.py
```

6. Buat file spec pyinstaller (terlampir):

```
$ nano gva-mycroft.spec
```

7. Build aplikasi Google Assistant terintegrasi Mycroft Precise:

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

```
(env) isro@isro-vmware:~/qcom_emulator/google-assistant$ ls dist/gva-mycroft/
audioop.cpython-35m-x86_64-linux-gnu.so
base_library.zip
_bz2.cpython-35m-x86_64-linux-gnu.so
certifi
_cffi_backend.cpython-35m-x86_64-linux-gnu.so
_codecs_cn.cpython-35m-x86_64-linux-gnu.so
_codecs_hk.cpython-35m-x86_64-linux-gnu.so
_codecs_iso2022.cpython-35m-x86_64-linux-gnu.so
_codecs_jp.cpython-35m-x86_64-linux-gnu.so
_codecs_kr.cpython-35m-x86_64-linux-gnu.so
_codecs_tw.cpython-35m-x86_64-linux-gnu.so
cryptography
cryptography-3.2.1-py3.5.egg-info
_ctypes.cpython-35m-x86_64-linux-gnu.so
_curses.cpython-35m-x86_64-linux-gnu.so
_decimal.cpython-35m-x86_64-linux-gnu.so
```

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'