**4장. SMS 및 음성 알림 재생**

클라우드 텔레포니(Cloud telephony)는 전화 시스템을 클라우드로 이동시키는 기술이다. 4장은 클라우드 텔레포니에 대한 소개로 시작하며, 파이썬의 텍스트 메시지와 음성 메시지로 비즈니스 사용 사례의 자동화를 다룬다.

4장은 다음을 살펴볼 것이다.

-클라우드 텔레포니 공급자 등록

-텍스트 메시지 전송

-SMS 메시지 수신

-도미노의 SMS 워크플로우

-음성 메시지 전송

-음성 통화 수신

-나만의 고객 서비스 소프트웨어 구축

**소개**

이전 장에서는 평문 및 CSV(comma-separated value) 파일을 사용한 작업을 살펴본 다음 범위를 확장하여 엑셀 워크시트, 워드 문서 및 PDF 파일 작업에 대해 살펴봤다. Excel, Word 및 PDF 파일은 이진 형식으로 제공되며 수학 연산, 표, 차트 및 기타 여러 작업을 지원한다. 파이썬으로 자동화할 수 있는 흥미로운 비즈니스 사용 사례도 살펴봤다.

4장에서는 클라우드 텔레포니의 흥미로운 세계를 살펴본다. 인터넷의 출현으로 기업들은 커뮤니케이션 시스템을 클라우드로 이전했다. 인터넷 기반의 호스팅된 텔레포니는 PBX와 같은 기존의 전화 장비를 대체했다. 이로 인해 클라우드 텔레포니를 사용하여 비즈니스 요구사항을 해결할 수 있는 가능성이 열렸고 파이썬도 마찬가지다. 비즈니스에서 클라우드 텔레포니를 사용하면 여러 통화와 SMS를 동시에 주고받을 수 있다. 통화 전송, 녹취, 대량 SMS와 같은 서비스는 클라우드 텔레포니를 활용할 수 있는 놀라운 기능이다.

클라우드 텔레포니는 품질, 비용 및 추가적인 인프라에 투자하지 않고도 비즈니스 요구사항을 관리할 수 있다.

4장의 레시피는 SMS 메시지 및 음성 통화(voice calls)를 송/수신하는 데 도움이 되는 파이썬 모듈을 살펴볼 것이다. 클라우드 텔레포니 공급자에 등록하고 파이썬 API를 사용하여 흥미로운 비즈니스 플로우를 자동화하는 방법을 다룬다. 특히, Twilio 텔레포니 공급자와 함께 작업할 것이고 4장은 다음 파이썬 모듈을 사용한다.

-Flask(http://flask.pocoo.org/)

-twilio(https://pypi.python.org/pypi/twilio)

<<중요표시시작>>

Twilio 클라우드 텔레포니 공급자를 살펴보겠지만 다른 공급자도 있다. 이들 각각은 훌륭한 API를 가지고 있으며 파이썬 라이브러리로 작동한다. 솔루션을 구현할 경우, http://www.capterra.com/telephony-software/에서 다양한 종류를 살펴볼 수 있다.

<<중요표시끝>>

**클라우드 텔레포니 공급자 등록**

클라우드 기반 텔레포니 서비스를 사용하려면 텔레포니 공급자와 계정을 등록해야 한다. 인터넷을 검색하면 사용할 수 있는 다수의 인기있는 클라우드 공급자가 있다. 4장에서는 Twilio(https://www.twilio.com/)를 사용한다. 계정을 등록하는 방법을 살펴보자.

**준비하기**

클라우드 텔레포니 API를 사용하려면 AccountSID 및 AuthToken을 얻기 위해 Twilio 계정에 등록해야한다. 또한 ‘SMS 및 음성’ 절의 레시피를 위한 번호를 임대해야 한다. 이 레시피에서 Twilio API를 사용하는 방법을 살펴보자.

**실행방법**

1. 컴퓨터에서 가장 좋아하는 브라우저를 열고 <https://www.twilio.com/try-twilio>를 탐색한다.

<그림>

2. 계정을 생성했으면 계정에 로그인하고 Billing 페이지에서 계정 대시보드의 드롭다운 옵션으로 사용할 수 있는 금액을 추가한다. 이미 로그인한 경우 Billing 페이지(https://www.twilio.com/user/billing)를 직접 검색할 수도 있다.

<그림>

3. Twilio API를 호출하려면 AccountSID 및 AuthToken이 필요하다. 드롭다운의 Account 절을 클릭하거나 https://www.twilio.com/user/account/settings를 직접 탐색하여 이러한 세부 정보를 얻을 수 있다. 이 페이지에서는 다음 스크린샷과 같은 API 자격증명(credentials)을 받는다. 지금은 2단계 인증을 걱정말고 계정에서 SSL Certification Validation이 가능한지 확인한다.

<그림>

4. 좋다. 이제 https://www.twilio.com/user/account/phone-numbers/search를 직접 탐색하여 전화 번호를 임대하자.

5. Buy a Number를 클릭하면 국가(country) 및 지역번호(prefix) 혹은 지역 코드(area code)로 번호를 임대할 수 있는 페이지가 표시된다. 여기에서 미국(United States)을 국가로 선택하고 510을 지역번호로 사용한다. 또한 SMS 메시지와 음성 통화를 모두 처리할 수 있는 번호를 임대한다.

<그림>

6. 이제, 이 페이지 하단의 Advanced Search 옵션을 클릭하여 다음 스크린샷과 같이 모든 옵션을 펼쳐본다. 기본 설정인 All로 진행하거나 Local 혹은 Toll-Free 번호를 임대할 수 있다. 지역이나 국가의 현지 규칙에 따라 번호를 임대할 때는 주소의 증빙 자료를 제공해야 하지만, 이 모든 정보가 필요하지 않은 Any 옵션을 선택할 수도 있다. Beta Numbers는 특정 국가의 Twilio에 새로 추가된 번호이며, 국가 목록을 지원하기 위해 추가됐다. 이 레시피는 Beta Numbers 옵션에 신경 쓸 필요가 없으므로 빈칸으로 둔다.

<그림>

7. Search를 클릭하면 이 페이지는 결과 화면으로 이동하여 사용 가능한 번호를 구입할 수 있다. 음성 및 SMS를 사용할 수 있는 번호를 구입했는지 확인한다. Toll-free 번호는 비싸기 때문에 이 연습을 위해 Local 번호를 구입하는 것이 가장 좋다.

<그림>

8. 좋아! 이것을 이미 끝냈으면 축하한다! 이제 SMS 메시지 및 음성 통화를 파이썬 API와 함께 사용하는 방법을 살펴본다.

**작동원리**

앞에서 설명한 것처럼 Twilio API를 사용하려면 계정을 등록해야 한다. Twilio는 계정을 생성하고 고유한 AccountSID 및 AuthToken을 제공하는데, 이는 요청을 검증하고 계정에 API호출에 대한 요금이 청구되도록 한다.

Twilio의 전화 번호는 발신자 ID(caller IDs)로 사용되어 텍스트 메시지 혹은 음성 통화를 전송한다. 발신자 ID(발신자 식별로도 알려짐)는 착신자의 장비(유선 전화 혹은 휴대 전화)에서 깜박이는 발신자 번호이다. 이 경우 Twilio에서 임대한 번호가 발신자 ID(caller ID)로 사용된다.

**더 알아보기**

계정을 생성하고, AccountSID 및 AuthToken을 얻고, Twilio를 사용하여 전화 번호를 생성하는 방법을 살펴봤다. 이제 다음 레시피에서는 이것을 사용한다.

**텍스트 메시지 전송**

첫 번째 레시피는 텍스트 메시지 작업을 살펴보자. 이 레시피는 수신자에게 SMS 메시지를 전송한다. 다음 작업 수행을 위해 계정에 비용을 청구해야 할 수도 있다.

**준비하기**

Twilio API를 사용하여 SMS 메시지를 전송하는 것으로 시작한다. 이 절에서는 수행 방법을 살펴보자. 이렇게 하기 전에 파이썬 가상 환경을 생성하고 다음 단계를 따라 flask 및 twilio 모듈을 설치한다.

flask를 사용하여 간단한 웹 서비스를 호스팅할 것이며, 이를 텔레포니 공급자 Twilio가 호출할 것이다. 이 후 flask 앱이 Twilio의 콜백을 기반으로 필요한 비즈니스 작업을 수행한다. 레시피를 살펴보면 이것에 대해 더 알게될 것이다.

가상 환경 설정 및 모듈 설치는 컴퓨터의 커맨드 라인에서 수행해야 한다. 파이썬 pip를 사용하여 flask와 twilio 모듈을 설치한다.

virtualenv ~/book/ch05/

source ~/book/ch05/

pip install flask

Collecting flask==0.10.1

Downloading Flask-0.10.1.tar.gz (544kB)

100% |████████████████████████████████| 544kB

774kB/s

Collecting Werkzeug>=0.7 (from flask==0.10.1)

Downloading Werkzeug-0.11.10-py2.py3-none-any.whl (306kB)

100% |████████████████████████████████| 307kB

1.5MB/s

Collecting Jinja2>=2.4 (from flask==0.10.1)

Downloading Jinja2-2.8-py2.py3-none-any.whl (263kB)

100% |████████████████████████████████| 266kB

2.4MB/s

Collecting itsdangerous>=0.21 (from flask==0.10.1)

Downloading itsdangerous-0.24.tar.gz (46kB)

100% |████████████████████████████████| 49kB

6.2MB/s

Collecting MarkupSafe (from Jinja2>=2.4->flask==0.10.1)

Downloading MarkupSafe-0.23.tar.gz

Building wheels for collected packages: flask, itsdangerous, MarkupSafe

Running setup.py bdist\_wheel for flask

Stored in directory:

/Users/chetan/Library/Caches/pip/wheels/b6/09/65/5fcf16f74f334a215447c26769

e291c41883862fe0dc7c1430

Running setup.py bdist\_wheel for itsdangerous

Stored in directory:

/Users/chetan/Library/Caches/pip/wheels/fc/a8/66/24d655233c757e178d45dea2de

22a04c6d92766abfb741129a

Running setup.py bdist\_wheel for MarkupSafe

Stored in directory:

/Users/chetan/Library/Caches/pip/wheels/a3/fa/dc/0198eed9ad95489b8a4f45d14d

d5d2aee3f8984e46862c5748

Successfully built flask itsdangerous MarkupSafe

Installing collected packages: Werkzeug, MarkupSafe, Jinja2,

itsdangerous, flask

Successfully installed Jinja2-2.8 MarkupSafe-0.23 Werkzeug-0.11.10

flask-0.10.1 itsdangerous-0.24

다음으로, 다음 커맨드를 사용하여 twilio를 설치한다.

pip install Twilio

Collecting twilio

Downloading twilio-5.4.0.tar.gz (193kB)

100% |████████████████████████████████| 196kB

2.2MB/s

Collecting httplib2>=0.7 (from twilio)

Downloading httplib2-0.9.2.zip (210kB)

100% |████████████████████████████████| 212kB

2.0MB/s

Collecting six (from twilio)

Downloading six-1.10.0-py2.py3-none-any.whl

Collecting pytz (from twilio)

Downloading pytz-2016.6.1-py2.py3-none-any.whl (481kB)

100% |████████████████████████████████| 483kB

1.0MB/s

Building wheels for collected packages: twilio, httplib2

Running setup.py bdist\_wheel for twilio

Stored in directory:

/Users/chetan/Library/Caches/pip/wheels/91/16/85/2ea21326cf1aad3e32f88d9e81

723088e1e43ceb9eac935a9b

Running setup.py bdist\_wheel for httplib2

Stored in directory:

/Users/chetan/Library/Caches/pip/wheels/c7/67/60/e0be8ccfc1e08f8ff1f50d99ea

5378e204580ea77b0169fb55

Successfully built twilio httplib2

Installing collected packages: httplib2, six, pytz, twilio

Successfully installed httplib2-0.9.2 pytz-2016.6.1 six-1.10.0

twilio-5.4.0

**실행방법**

1. 먼저 Twilio AccountSID 및 AuthToken을 포함하는 설정을 생성하는 것으로 시작하자. 설정에 발신자 ID(caller ID)에 임대 번호(rented number)를 추가하고 메시지를 전송할 커스텀 번호(custom number)도 추가한다. 이 필드에 나만의 번호를 추가하여 적합한지 확인할 수 있다.

2. 설정 파일은 다음과 같고 config.py로 저장한다.

TWILIO\_ACCOUNT\_SID = 'Account SID'

TWILIO\_AUTH\_TOKEN = 'Auth Token'

CALLERID = '+Rented Number'

MYNUMBER = '+Your Number'

3. 이제 실제로 이 메시지를 전송할 애플리케이션을 작성해 보자. send\_sms.py로 저장하고 config.py의 설정을 임포트 한다.

import config

from flask import Flask

from twilio.rest import TwilioRestClient

app = Flask(\_\_name\_\_)

client = TwilioRestClient(config.TWILIO\_ACCOUNT\_SID,

config.TWILIO\_AUTH\_TOKEN)

message = client.messages.create(

to=config.MYNUMBER,

from\_=config.CALLERID,

body="Hey, this is cool!")

4. 터미널 혹은 좋아하는 편집기에서 이 코드를 실행한다. 휴대 전화를 확인해 보고, 임대 번호로부터 Hey, this is cool!라는 메시지를 받아야 한다. 정말로, 멋지다! 통신사(carrier)와 네트워크 정체에 따라 메시지 도착까지 2-3 분이 걸릴 수 있으므로 기다린다. 수신 메시지에 대한 다음 스크린샷을 살펴본다.

<그림>

**작동원리**

먼저 설정 파일을 생성하고 필요한 모든 필드를 채운다. send\_sms.py 파일에서 config.py를 임포트하여 이 설정 파일을 임포트하고 flask 및 twilio 파이썬 모듈과 같은 필수 모듈도 가져온다.

그런 다음 twilio.rest 모듈에서 임포트된 TwilioRestClient 클래스를 사용하여 twilio 객체인 client를 생성했다. AccountSID 및 AuthToken을 사용하여 객체를 생성한다.

이 후 client.messages 클래스의 create() 메소드를 사용하여 메시지를 전송한다. 여기서 to는 메시지를 수신하는 번호이고 from\_는 발신자 ID(caller ID)이며 body는 전송한 텍스트이다. 왜 from이 아니라 from\_일까? 그 이유는 모듈을 임포트하는 데 사용되는 파이썬의 from 키워드와 충돌하기 때문이다.

**더 알아보기**

간단하다, 그렇지? 텍스트 메시지를 번호로 전송하는 것은 쉬운 일이다. 이제, 나만의 애플리케이션을 작성할 수 있을까? 어디 한번 생각해보자. 아마도, 여러분의 생일 파티에 나를 초대하는 메시지를 전송한다.

그러나 들어오는 메시지를 수신하는 방법을 모른다면 무엇이 재미있을까? 다음 절에서는 이것을 실행하는 방법을 살펴보자.

**SMS 메시지 수신**

SMS 메시지를 전송하는 다양한 사용 사례와 마찬가지로, 텍스트 메시지를 수신하는 기능도 똑같이 중요하다고 생각한다. 따라서, 자동 응답 메시지 앱으로 살펴보자.

**준비하기**

이 레시피는 ngrok 소프트웨어를 사용한다. ngrok(https://ngrok.com/) 소프트웨어는 로컬 머신을 인터넷으로 터널링하는 것을 도와준다. 이는 NAT 혹은 방화벽(Firewall) 뒤에 있는 로컬 서버를 인터넷에 노출시킬 수 있음을 의미한다. 정말로 강력한 유틸리티다! 다음 레시피를 위해, ngrok(https://ngrok.com/download)를 다운로드하고 터미널의 커맨드를 사용하여 포트 5000으로 실행한다. 인터넷에 연결되어 있는 경우 인스턴스가 실행되고 있는 것을 확인할 수 있다.

./ngrok http 5000

다음 스크린샷은 ngrok가 앱을 인터넷과 URL(https://<uuid>.ngrok.io/)로 표시하는 방법을 보여준다.

<그림>

<<중요표시시작>>

아직 ngrok을 시작하지 않아도 된다. 레시피 중에 ngrok을 시작한다.

<<중요표시끝>>

**실행방법**

1. SMS 수신을 위한 서버 작성을 시작하자. recv\_sms.py라고 부를 것이다. 서버의 코드는 다음과 같다.

from flask import Flask

import twilio.twiml

app = Flask(\_\_name\_\_)

@app.route("/insms", methods=['GET', 'POST'])

def respond\_sms():

resp = twilio.twiml.Response()

resp.message("Thanks for your query. We will get back to you soon")

return str(resp)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

2. 다음 커맨드를 사용하여 터미널에서 파이썬으로 서버를 실행한다.

python recv\_sms.py

3. 다음 커맨드로 ngrok을 시작한다.

./ngrok http 5000

4. 좋아, 서버가 작동한다. Request URL을 추가하여 Twilio 번호를 구성하자. 이를 위해 Twilio에 로그인하고 Phone Numbers 절로 이동하고 임대 번호를 클릭한다. 여기서, Messaging 절로 이동하여 다음 스크린샷과 같은 Request URL을 입력한다. URL이 이전 단계에서 실행된 ngrok 인스턴스를 가리키는지 확인한다.

<그림>

이제 끝났다. 따라서 누군가가 임대 번호로 메시지를 전송하면, 그들은 Thanks for your query.We will get back to you soon라는 자동 응답을 수신할 것이다.

대단해! 잘했어, 그런데, 어떻게 작동할까?

**작동원리**

들어오는 메시지를 수신하는 서버는 Flask로 작성되고 포트 5000에서 실행된다. 이렇게 하면 서버가 로컬 머신에서 실행된다. 인터넷에서 사용할 수 있도록, ngrok을 실행하고 포트 5000인 Flask 서버와 동일한 포트에서 실행되도록 한다.

들어오는 메시지를 Flask서버로 라우팅하는 Twilio 전화 번호를 구성한다. 이를 위해, Request URL에 Twilio를 추가한다. 따라서 메시지가 임대 번호로 전송될 때마다, ngrok을 통해 Flask 서버로 라우팅된다. 앱에서는 [https://HYPERLINK "https://%3cngrokid%3e/"<HYPERLINK "https://%3cngrokid%3e/"ngrokIdHYPERLINK "https://%3cngrokid%3e/">](https://%3cngrokid%3e/).ngrok.io/insms로 라우팅됐다.

Flask 서버를 살펴보면 이미 URL /insms로 구성된 루트(route)가 있다. 이 루트는 Thanks for your query. We will get back toyou soon로 응답하는 Twilio 서버(Request URL 설정)에서 POST 호출을 얻는다.

**더 알아보기**

Twilio를 사용하여 텍스트 메시지를 송수신하는 받는 방법을 살펴봤다. 이미 사용 사례를 다루고 있으며 텍스트 메시지를 활용하여 이를 해결할 수 있는 방법을 안다. 식품 소매업의 예제를 살펴보자.

**도미노의 SMS 워크플로우**

미국의 도미노 피자 매장의 소유자인 존(John)은 피자 판매를 향상시킬 방법을 찾는다. 판매를 향상시키는 한 가지 방법은 피자를 주문하는 과정을 보다 쉽게 만드는 것을 살펴보는 것이다. 존은 고객이 주문 및 배송 상태를 업데이트할 수 있도록 워크플로우도 자동화하려고 한다. 존은 인터넷도 좋긴 하지만, 고객들이 네트워크 수신이 어려운 지역에 있을 때도 피자를 주문하고 싶어할 수 있다고 생각한다. 존이 무엇을 해야 한다고 생각하는가?

**준비하기**

사용 사례를 생각해 보고 필요한 것이 무엇인지 적어 볼까? 여기 생각한 몇 가지가 있다.

-들어오는 메시지 수신 기능

-주문 상태 유지 및 조회

-발송 상태 메시지 전송

**실행방법**

솔루션을 살펴보고 도미노 상점의 작동 방법을 파악한다.

다음 코드는 세 가지 주요 측면으로 나뉜다. flask 루트(route)를 통해 들어오는 메시지를 수신하는 기능, 주문을 조회하는 고객의 주문 상태를 유지하는 기능, Flask 앱에서 아웃바운드 메시지를 전송하는 기능이다.

from flask import Flask, request

import twilio.twiml

class Pizza:

def \_\_init\_\_(self):

self.status = None

def setStatus(self, status):

self.status = status

def getStatus(self):

return self.status

app = Flask(\_\_name\_\_)

@app.route("/insms", methods=['GET', 'POST'])

def respond\_sms():

content = request.POST['Body']

resp = twilio.twiml.Response()

if content == 'ORDER':

resp.message("Thanks! We're already on your order!")

pizza = Pizza()

pizza.setStatus('complete')

return str(resp)

if content == 'STATUS':

pizza = Pizza()

status = pizza.getStatus()

if status == 'complete':

resp.message("Your order is ready!")

return str(resp)

else:

resp.message("Sorry! could not locate your order!")

return str(resp)

else:

resp.message("Sorry! Wrong selection")

return str(resp)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**작동원리**

들어오는 메시지를 수신하는 앱을 이미 가졌다. 따라서 이 앱을 사용하고 필요에 따라 확장할 수 있다. 임대 번호는 광고에서 번쩍이는 도미노 번호가 된다.

-이 경우, 도미노 고객들이 사용할 수 있는 두 개의 키워드, 즉 ORDER 및 STATUS를 사용하기로 결정했다.

-고객이 도미노에 ORDER 메시지를 전송하면 원하는 피자를 주문할 수 있다. 피자 가게는 이미 그 주문을 작업중이라고 말함으로써 적극적으로 응답한다.

-고객이 주문 상태를 알고 싶으면 STATUS라는 텍스트를 사용하여 주문 상태를 확인할 수 있다. 우리의 경우, 고객이 주문에 대해 문의를 하면 피자 가게에서 주문을 준비한다는 응답을 받는다. 이 응답을 받았을 때 고객이 느끼는 만족도를 상상해본다.

-고객은 주문 상태를 매우 알기 쉽다고 생각한다. 고객은 더 많은 주문을 시작할 것이다. 글쎄, 적어도 난 그럴 것이다.

-존은 IT 관리자에게 월급을 인상하기로 결정한 것을 너무나 기쁘게 생각하고, 우연히도 바로 여러분에게 적용된다. 좋다, 그렇지?

**더 알아보기**

이제, 만약 피자 가게에 의해 혹은 피자 가게로 전송된 메시지를 살펴보면, 메시지는 임대 번호로 전송될 것이다. DOMP(Domino's Pizza의 약자)와 같은 사용자 정의 코드로 메시지를 전송하는 것은 어떨까? 그래, SMS 단축 코드로 그 효과를 볼 수 있다. 무료로 사용할 수 없기 때문에 비싼 가격에 구입하여야 한다.

**참고사항**

-SMS 메시지의 모든 레시피가 끝났다. 몇 가지를 다뤘고 수익을 위해 몇 가지를 구현하기를 희망한다. 도미노 로고를 사용하여 존경하는 고객에게 오퍼(offer) 같은 MMS 메시지를 전송할 수 있을까? 그것은 여러분이 탐험할 만한 것이다.

-다음 레시피에서는 음성 통화(voice calls)를 시작하고 무엇을 할 수 있는지 살펴본다. 음성 메시지로 멋진 일을 할 수 있을까? 다음 레시피에서 이것을 살펴보자.

**음성 메시지 전송**

VoIP를 들어본 적이 있는가? 그래, VoIP는 Voice over Internet Protocol이다. VoIP(약어)는 인터넷 자체와 같은 인터넷 프로토콜 네트워크를 통해 음성 및 멀티미디어를 전송하는 데 사용되는 기술이다. VoIP는 소비자와 엔터프라이즈 도메인에서 통신 솔루션을 제공하는 스카이프(Skype)와 구글 토크(Google Talk)와 같은 제품과 인터넷에서 통신을 위해 완전히 새로운 세계를 개방했다.

Twilio와 같은 텔레포니 API 공급자는 VoIP 프로토콜을 사용하여 음성 메시지도 전송한다. 이 절에서는 Twilio API를 사용하여 음성 통화(voice calls)를 송신하거나 수신하는 방법을 배운다. 이제 API를 사용하자.

**준비하기**

이 레시피에서는 이전의 레시피에서 사용한 것과 동일한 twilio 및 flask 모듈을 사용한다. 따라서 이 절에는 새로운 설치가 필요하지 않다.

**작동밥법**

1. 먼저 설정 파일을 생성한다. 음성 메시지를 전송하는 것은 텍스트 메시지를 전송하는 것처럼 쉽지 않다. Twilio AccountSID와 AuthToken도 필요하다. 음성 메시지를 전송하려면 발신자 ID(caller ID)와 번호가 필요하다. 설정을 살펴보자.

TWILIO\_ACCOUNT\_SID = '<Account SID>'

TWILIO\_AUTH\_TOKEN = '<Auth Token>'

CALLERID = '<Rented Number>'

MYNUMBER = '<Number to call>'

2. 이제 Flask 서버 코드를 작성하자. 다음 코드는 Twilio 파이썬 API를 사용하여 음성 통화를 생성하는 데 도움이 된다. 파일을 voice\_outbound.py로 저장한다.

import config

from flask import Flask, Response, request

from twilio import twiml

from twilio.rest import TwilioRestClient

app = Flask(\_\_name\_\_)

client = TwilioRestClient(config.TWILIO\_ACCOUNT\_SID,

config.TWILIO\_AUTH\_TOKEN)

@app.route('/call', methods=['POST'])

def outbound\_call():

response = twiml.Response()

call = client.calls.create(

to=config.MYNUMBER,

from\_=config.CALLERID,

record='true',

)

return Response(str(response), 200,

mimetype="application/xml")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

3. 다음 커맨드로 Flask 서버를 실행한다. 기본 포트 5000에서 Flask 서버가 실행될 것이다.

python voice\_outbound.py

4. 포트 5000에서 다음 커맨드로 ngrok을 시작한다. ngrok에서 제공하는 터널링 기능으로 인터넷에서 서버가 이용가능하도록 할 수 있다. ngrok이 실행되고 있는 URL을 복사한다. 이전 절에서 본 것처럼 [https://HYPERLINK "https://%3cngrokid%3e/"<HYPERLINK "https://%3cngrokid%3e/"ngrokidHYPERLINK "https://%3cngrokid%3e/">](https://%3cngrokid%3e/).ngrok.io/ 형식일 것이다.

./ngrok http 5000

이제 서버에서 통화를 할 준비가 되었으므로 [https://HYPERLINK "https://%3cngrokid%3e.ngrok.io/call"<HYPERLINK "https://%3cngrokid%3e.ngrok.io/call"ngrokidHYPERLINK "https://%3cngrokid%3e.ngrok.io/call">HYPERLINK "https://%3cngrokid%3e.ngrok.io/call".ngrok.io/call](https://%3cngrokid%3e.ngrok.io/call)로 이동하여 POST요청을 하면, 구성 파일에 추가된 대로 여러분의 번호로 통화를 수신한다.

정말, 멋지다! 하지만 전화를 수신하면 어떻게 될까? 통화가 끊어진다. 왜? 이는 Twilio에서는 모든 음성 통화에 콜백 URL이 포함되기 때문에 콜백 URL은 일단 선택되면 다음 명령을 실행할 것이다. 이것은 코드에 정의되어 있지 않으므로 오류가 발생하여 통화가 끊어진다. 이 문제를 해결하자.

5. 이제 응답 콜백 URL을 추가하고 서버 코드를 완성하자.

import config

from flask import Flask, Response, request

from twilio import twiml

from twilio.rest import TwilioRestClient

app = Flask(\_\_name\_\_)

client = TwilioRestClient(config.TWILIO\_ACCOUNT\_SID,

config.TWILIO\_AUTH\_TOKEN)

@app.route('/call', methods=['POST'])

def outbound\_call():

response = twiml.Response()

call = client.calls.create(

to=config.MYNUMBER,

from\_=config.CALLERID,

record='true',

url=config.BASE\_URL + '/answer\_url',

)

return Response(str(response), 200,

mimetype="application/xml")

@app.route('/answer\_url', methods=['POST'])

def answer\_url():

response = twiml.Response()

response.addSay("Hey! You are awesome. Thanks for answering.")

return Response(str(response), 200,

mimetype="application/xml")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

6. outbound\_call() 메소드의 url 매개변수를 살펴보면 BASE\_URL을 가리킨다. 이는 /answer\_url 접미사가 추가된 ngrok URL과 동일하다. 이제 [https://HYPERLINK "https://%3cngrokid%3e.ngrok.io/call"<HYPERLINK "https://%3cngrokid%3e.ngrok.io/call"ngrokidHYPERLINK "https://%3cngrokid%3e.ngrok.io/call">HYPERLINK "https://%3cngrokid%3e.ngrok.io/call".ngrok.io/call](https://%3cngrokid%3e.ngrok.io/call)에 POST 요청을 하면 번호는 전화를 수신할 것이다. 전화를 수신하면 [https://HYPERLINK "https://%3cngrokid%3e/"<HYPERLINK "https://%3cngrokid%3e/"ngrokidHYPERLINK "https://%3cngrokid%3e/">](https://%3cngrokid%3e/).ngrok.io/answer\_url에 콜백 POST 요청이 이루어지고 "Hey! You are awesome. Thanks for answering" 메시지가 들릴 것이다. 좋다!

7. 서버 로그는 다음과 같다.

\* Detected change in '/Users/chetan/book/ch05/app.py',

reloading

\* Restarting with stat

\* Debugger is active!

\* Debugger pin code: 608-057-122

127.0.0.1 - - [16/Jul/2016 21:35:14] "POST

/call HTTP/1.1" 200 -

127.0.0.1 - - [16/Jul/2016 21:35:22] "POST

/answer\_url HTTP/1.1" 200 -

**작동원리**

그렇다면, 어떻게 작동할까? 'SMS' 절을 마친 후에는 이해하기가 수월해야 한다. 코드를 단계별로 살펴본다.

1. 우선 twilio.rest 모듈의 TwilioRestClient 클래스를 사용하여 twilio 객체인 client를 생성한다.

2. Flask 앱 루트(route)를 /call로 정의하고 POST 메소드 호출을 받아들인다. 이 루트는 번호로 음성 통화를 한다.

3. 실제 호출은 client.calls 클래스의 create() 메소드를 사용하여 outbound\_call() 루트 메소드에서 수행된다.

4. create() 호출에서는 다음과 같은 매개변수를 정의한다.

-to: 호출되는 모바일/유선전화 번호이다.

-from\_: Twilio의 임대 번호이다.

-record : 통화 기록 여부를 결정한다.

-url: 음성 통화에 응답할 때 호출되는 콜백 응답 url이다.

5. Flask 앱에서는 통화가 수신될 때 호출되는 새로운 루트인 /answer\_url을 정의했다. 지금, 이것을 이해하는 것은 흥미롭다. Twilio는 Twilio 마크업 언어(markup language)라고도 하는 TwiML의 철학에 공을 들인다. 마크업을 살펴보면 XML 마크업과 꽤 흡사하다. TwiML은 들어오는 SMS 혹은 음성 통화와 관련하여 Twilio에게 필요한 것을 알리는 데 사용할 수 있는 지시사항이다. 따라서 addSay() 메소드는 다음과 같다.

<?xml version="1.0" encoding="UTF-8"?> <Response>

<Say>Hey! You are awesome. Thanks for answering</Say>

</Response>

**더 알아보기**

따라서 주어진 번호로 음성 통화를 송신하는 방법과, 전화를 받을 때 응답 콜백 URL이 호출되는 방법을 다뤘다. 이제 들어오는 음성 통화를 처리하는 방법에 대해 살펴보자.

**음성 통화 수신**

음성 통화를 수신하는 것은 클라우드 텔레포니를 사용하여 앱을 개발하는 데 중요한 측면이다. 많은 비즈니스 사례(상상했던대로)가 들어오는 음성 통화에 의존한다. Twilio API를 사용하여 수신 음성 통화를 처리하는 방법을 살펴보자.

**준비하기**

이 레시피에서는 SMS의 이전 레시피에서 사용한 것과 같은 twilio 및 flask 모듈을 사용한다. 따라서 이 절에는 새로운 설치가 필요하지 않다.

**실행방법**

1. 먼저 설정 파일을 생성한다. Twilio AccountSID 및 AuthToken도 필요하다. 이 경우 임대 번호 자체는 발신자 ID(caller ID)이므로 발신자 ID(caller ID)가 필요하지 않다.

2. 이제 Flask 서버를 살펴보자. 코드는 다음과 같다. 이를 voice\_inbound.py라고 한다.

from flask import Flask, Response, request

from twilio import twiml

from twilio.rest import TwilioRestClient

app = Flask(\_\_name\_\_)

client = TwilioRestClient(config.TWILIO\_ACCOUNT\_SID,

config.TWILIO\_AUTH\_TOKEN)

@app.route('/incall', methods=['POST'])

def inbound\_call():

response = twiml.Response()

response.addSay("Thanks for calling our customer

service." "Please hold while we

connect you to our advisors.")

return Response(str(response), 200,

mimetype="application/xml")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

3. 다음 커맨드로 Flask 서버를 실행한다. 그러면 기본 포트 5000에서 Flask 서버가 실행된다.

python voice\_outbound.py

4. 그러면 포트 5000에서 다음 커맨드로 ngrok을 시작한다. 이는 ngrok이 제공한 터널링 기능으로 서버가 인터넷에서 이용가능하도록 할 것이다. ngrok가 실행중인 URL을 복사한다. 이전 절에서 살펴본 것처럼 [https://HYPERLINK "https://%3cngrokid%3e/"<HYPERLINK "https://%3cngrokid%3e/"ngrokidHYPERLINK "https://%3cngrokid%3e/">](https://%3cngrokid%3e/).ngrok.io/ 형식이다.

./ngrok http 5000

5. 이제 Twilio에 로그인하여 들어오는 음성 통화의 임대 번호를 구성한다. Request URL이 ngrok URL을 가리키도록 구성한다. 임대 번호에 Request URL을 추가하는 방법을 알기위해 다음 스크린샷을 살펴본다.

<그림>

6. Twilio의 서버가 실행되고 구성이 설정되면 스카이프(Skype) 혹은 구글 토크(Google Talk)를 통해 임대 번호로 전화한다. 이는 Flask 서버에 POST호출을 하게 되고, Thanks for calling our customer service. Please hold while we connect you to our advisors라고 TwiML 응답으로 다시 응답할 것이다.

**작동원리**

들어오는 메시지를 받아들이는 서버는 Flask로 작성되고 포트 5000에서 실행된다. 이것은 머신의 로컬이고, 인터넷에서 사용할 수 있게 하기 위해서, ngrok으로 터널링을 생성한다.

이제 임대 번호가 누군가에게 수신을 받을 때, Twilio는 Request URL을 찾고 이 URL에 요청하여 임대 번호로 걸려오는 전화가 있음을 알려준다.

flask 서버는 임대 번호가 들어오는 통화를 수신할 때 호출되는 /incall(Request URL과 일치) 루트를 정의한다. /incall 루트는 차례로 <Response> 마크업에 <Say>를 추가하는 TwiML 응답을 생성하고, 발신자(caller)는 <Say> XML에 추가된 메시지를 얻는다.

다음 스크린샷은 Twilio에서 TwiML 응답이 어떻게 나타나는지 보여준다. 그런데, Twilio 인터페이스에서 수신하거나 송신된 모든 통화 혹은 SMS를 볼 수 있다.

<그림>

**나만의 고객 서비스 소프트웨어 구축**

폴(Paul)은 회사에서 고객 서비스를 담당한다. 회사는 채팅으로 고객 불만이나 질문을 받을 수 있는 기능을 갖춘 멋진 웹 사이트를 운영한다. 폴은 고객이 제품 문제에 직면했을 때 회사 직원과 연락하고 신속하게 해결하고자 할 때 채팅 시스템이 유용하지 않다는 의견을 종종 받는다. 폴의 삶을 더 편하게 할 수 있을까?

**준비하기**

사용 사례에 대해 생각하고 필요한 것이 무엇인지 적어보는 것이 어떨까? 생각할 수 있는 몇 가지가 있다.

-들어오는 통화 수신 기능

-고객 지원 엔지니어에게 전화 연결

**실행방법**

그 해결책을 살펴보고 그것이 폴에게 어떻게 작용할지 이해한다.

이 코드에서 임대 번호로 걸려 오는 전화를 받을 수 있는 기능을 추가하고 통화 전송 기능도 추가할 것이다.

import config

from flask import Flask, Response, request

from twilio import twiml

from twilio.rest import TwilioRestClient

app = Flask(\_\_name\_\_)

client = TwilioRestClient(config.TWILIO\_ACCOUNT\_SID,

config.TWILIO\_AUTH\_TOKEN)

@app.route('/call', methods=['POST'])

def inbound\_call():

call\_sid = request.form['CallSid']

response = twiml.Response()

response.dial().conference(call\_sid)

call = client.calls.create(to=config.MYNUMBER,

from\_=config.CALLERID,

url=config.BASE\_URL +

'/conference/' + call\_sid)

return Response(str(response), 200,

mimetype="application/xml")

@app.route('/conference/<conference\_name>',

methods=['GET', 'POST'])

def conference\_line(conference\_name):

response = twiml.Response()

response.dial(hangupOnStar=True).conference(

conference\_name)

return Response(str(response), 200,

mimetype="application/xml")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**작동원리**

들어오는 통화를 수신하는 앱을 이미 생성했다. 고객이 임대 번호로 전화를 걸면 inbound\_call() 메소드로 정의한 /call 루트로 POST호출을 수행하도록 이 앱을 사용 사례로 확장한다.

flask 루트는 들어오는 통화를 받아서 TwiML 명령어의 도움으로 컨퍼런스(conference)에 추가한다. 여러분도 아시다시피, 컨퍼런스는 하나의 컨퍼런스에서 서로 연결된 통화로 이루어진다.

response.dial().conference(conference\_name) 메소드는 컨퍼런스에 콜 레그(leg)를 추가하는 데 도움이 되는 메소드다. 이것은 TwiML이 보이는 방법이다. <Dial>과 <Conference> 태그가 있는 <Response> 태그를 확인할 수 있다.

<그림>

flask 루트는 고객 지원 엔지니어에게 즉시 통화할 수 있도록 한다(MYNUMBER로 식별됨). 고객 지원 엔지니어에 대한 발신 통화는 url 매개변수('발신 음성 통화' 절에서 보았듯이 응답 URL)로 구성된다. 따라서 지원 엔지니어가 전화를 받으면 콜백 응답 URL이 호출되고 엔지니어 콜 레그는 들어오는 콜 레그와 동일한 컨퍼런스에 추가됐다.

콜 레그와 이제 고객에게 걸려오는 통화와 지원 엔지니어와의 통화가 모두 하나의 컨퍼런스로 연결되어 대화를 나눌 수 있다. 고객은 자신의 질문을 신속하게 해결하고 폴은 행복하다. 좋아!

**더 알아보기**

클라우드 텔레포니 API를 사용하여 나만의 SMS 및 음성 애플리케이션을 생성하는 방법을 살펴봤다. 그러나 필요에 맞게 이미 구축된 솔루션을 활용하는 데 정말로 관심이 있다면 합리적인 비용으로 효율적으로 사용 사례 자동화에 도움이 되는 CallHub(https://callhub.io/)와 같은 일부 표준 소프트웨어 애플리케이션을 살펴볼 수 있다. API로 나만의 콜 센터 솔루션을 구축할 수도 있다. 그래서 다음에 무엇을 만들 계획인가?

나는 여러분이 4장을 즐긴 것을 확신한다. 5장은 좀 더 재미있게 즐기자. 우리가 가지고 있는 것을 살펴보자.