Title: README HLS Between An iOS Apps And A Server On Ubuntu Via MAN

Document Number: 103-5

CTI One Corporation

Table 1a. Document History

2021-09-29	Establish this document, document archive: /media/harry/easystore/backup-2020-2-15/CTI0/3proejcts/3-8-smart-tech/3-8-4-CTI/3-8-4-6-products/AIV200/103-4-tx2-Yolo-tensorRT	YY, ZW (Please add the company's master archive location)

Table 1b. Testing and Release Approval Form

2021-08-31	Tested by ??? and approved for release by ???	Pending for testing and approval

Table 2. References

Number	Name and URL	Note
1.	FFmpeg Options http://underpop.online.fr/f/ffmpeg/help/options-51.htm.gz	

2.	FFmpeg Encode in H.264 http://trac.ffmpeg.org/wiki/Encode/H.264	
3.	x264 FFmpeg Options Guide https://sites.google.com/site/linuxencoding/x264-ffmpeg-mapping	
4.	How To Add a Button in Xcode (Swift) https://www.zerotoappstore.com/how-to-add-a-button-in-xcode-swift.html	
5.	ngrok Setup https://dashboard.ngrok.com/get-started/setup	

Table 3. Prerequisite

Software Prerequisite No.	Description and Version	Note
1.	Ubuntu 18.04	
2.	Python version 3.6.9	On Ubuntu
3.	OpenCV 3.4.2	On Ubuntu
4.	FFmpeg version 4.3.1	On Ubuntu
5.	Django version 3.1.8	On Ubuntu
6.	AIV-100 version 2.0	On Ubuntu To produce HTTP

		communication
7.	macOS Big Sur version 11.5.2	On Mac
8.	Xcode versoin 13.0	On Mac
Hardware Prerequisite No.	Description and Version	
1.	Apple Mac, which supports macOS Big Sur	

1. HLS Video Streaming Algorithm

1.1. Web Server on Ubuntu laptop: Web Server produces the static file folder (cti/static/hls/ipcam) that allows iOS Apps on Macbook development environment, iPad, or iPhone to be able to access;

1.2. HLS server-side program on Ubuntu laptop: HLS server-side program reads the video stream from a IP Cam

1.3. HLS server-side program on Ubuntu laptop: HLS server-side program generates *.m3u8, contains bandwidth, and resolution and *.m4s (video data file) in the static file folder (cti/static/hls/ipcam)

1.4. Ubuntu laptop and ngrok: Ubunt laptop and ngrok, a web service, create a SSH tunnel for Port Forwarding

1.5. iOS Apps on the client-side: iOS Apps reads the basement *.m3u8 file from Ubuntu laptop through ngrok via HTTP, select resolution and read specific resolution *.m3u8 file which contains m4s file name and sequence

1.6. iOS Apps on the client-side: iOS Apps reads m4s files sequentially and play

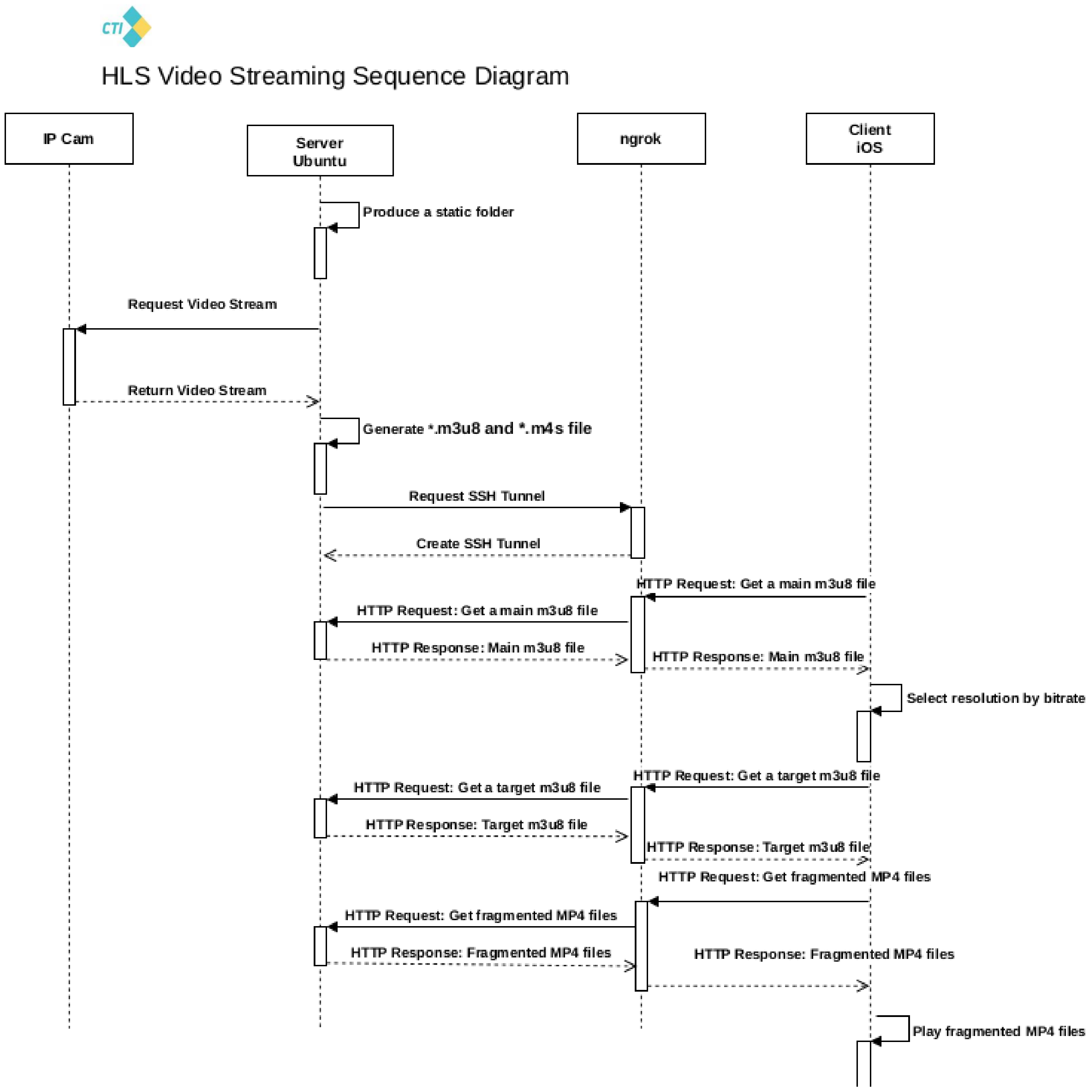

HLS Video Streaming Sequence Diagram
IP Cam
Server Ubuntu
ngrok
Client iOS
Produce a static folder
Request Video Stream
Return Video Stream
Generate *.m3u8 and *.m4s file
Request SSH Tunnel
Create SSH Tunnel
HTTP Request: Get a main m3u8 file
HTTP Request: Get a main m3u8 file
HTTP Response: Main m3u8 file
HTTP Response: Main m3u8 file
Select resolution by bitrate
HTTP Request: Get a target m3u8 file
HTTP Request: Get a target m3u8 file
HTTP Response: Target m3u8 file
HTTP Response: Target m3u8 file
HTTP Request: Get fragmented MP4 files
HTTP Request: Get fragmented MP4 files
HTTP Response: Fragmented MP4 files
HTTP Response: Fragmented MP4 files
Play fragmented MP4 files

Figure 1: HLS Video Streaming Sequence Diagram

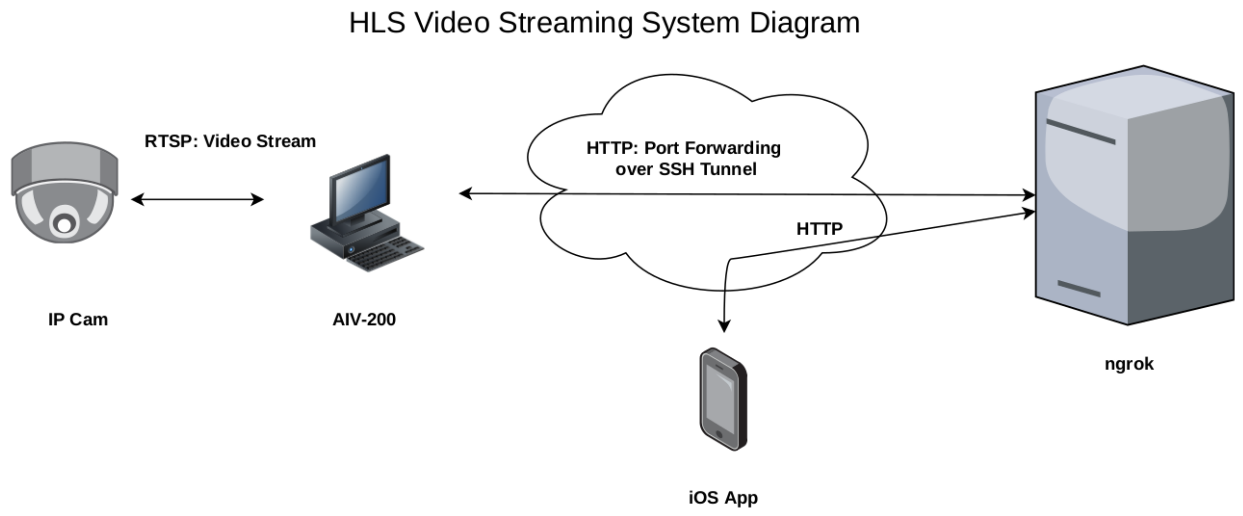

HLS Video Streaming System Diagram
RTSP: Video Stream
HTTP: Port Forwarding over SSH Tunnel
HTTP
IP Cam
AIV-200
ngrok
iOS App

Figure 2: HLS Video Streaming System Diagram

2. Install Dependecies

2.1. Install FFmpeg

```
$sudo nap install ffmpeg
```

Note: The above command will install FFmpeg version 4.3.1 as of Sep. 10, 2021.

2.2. Check FFmpeg version

```
$ffmpeg -version
```

If Python virtual environmeqnt is used, the result may show old version.

Check the FFmpeg location;

```
$which ffmpeg
```

If the result is not /snap/bin/ffmpeg, change the which command result file name. For instance, change the file name to ffmpeg_old.

3. Create the server side program on Ubuntu

3.1. Create hls_videostreaming_rtsp_opencv.py

===

```
import subprocess

import cv2

import traceback

import numpy as np

import time


from VideoGet import VideoGet


WIDTH = 360

HEIGHT = 240

FRAME_SIZE = str(WIDTH) + 'x' + str(HEIGHT)

DUMMY_FRAME = np.zeros([HEIGHT, WIDTH, 3], dtype=np.uint8)  # Dummy frame for black screen

DUMMY_FRAME[:, :, 2] = 255
```

Change the URL

```
source = 'rtsp://admin:admin123@192.168.2.63'

videoGetter = VideoGet(source).start()


cv2.namedWindow("Employee Out", cv2.WINDOW_NORMAL)
```

```
command_out = ['ffmpeg',
        '-y',  # (optional) overwrite output file if it exists
        '-f', 'rawvideo',
        '-vcodec', 'rawvideo',
        '-s', FRAME_SIZE,       # size of one frame '360x240'
        '-video_size', FRAME_SIZE,
        '-pix_fmt', 'yuv420p',   # OpenCV uses BGR format(bgr24). Default value is
yuv420p
        '-framerate', '23',      # frames per second
        '-i', '-',               # The imput comes from a pipe
        '-vcodec', 'h264',      # MPEG4 video codec ''mpeg4
        '-c:v', 'libx264',
        '-c:a', 'copy',
        '-bufsize', '1835k',     # Output Buffer memory size
        '-hls_init_time', '2',   # seconds.  Set the initial target segment length in seconds.
Default value is 0.
        '-hls_time', '2',        # seconds.  Set the target segment length in seconds. Default
value is 2.
        '-hls_list_size', '2',   # Set the maximum number of playlist entries. If set to 0 the
list file will contain all the segments. Default value is 5.
        '-preset', 'veryfast',   # Encoding speed to compress. The slower preset provides
better compression (compression is quality per filesize). Default value is medium
```

```
                         # ultrafast, superfast, veryfast, faster, fast, medium – default preset, slow, slower, veryslow
            '-tune', 'zerolatency',  # Change settings based upon the specifics of your input.
                         # film – use for high quality movie content; lowers deblocking
                         # animation – good for cartoons; uses higher deblocking and more reference frames
                         # grain – preserves the grain structure in old, grainy film material
                         # stillimage – good for slideshow-like content
                         # fastdecode – allows faster decoding by disabling certain filters
                         # zerolatency – good for fast encoding and low-latency streaming
            '-x264-params', 'keyint=20',   # Keyframe interval, also known as GOP length. Recommended default: 250
            '-hls_wrap', '10',
            '-hls_allow_cache', '0', # Explicitly set whether the client MAY (1) or MUST NOT (0) cache media segments.
            '-hls_segment_type', 'fmp4',
            '-start_number', '1',
            '-f', 'hls',
            '/home/yusuke/Documents/CTI_One_Corp/2_Work/AIV-200/ServerSide-Python/aiv200-test/cti/static/hls/ipcam/ipcam_hls.m3u8']
```

Change the path

```
pipe_out = subprocess.Popen(command_out, bufsize=4092, stdin=subprocess.PIPE)


count = 1
```

```
try:

    while True:

        start_time = time.time()

        frame = videoGetter.get_frame()

        frameOriginal = frame.copy()

        cv2.imshow("Employee Out", frameOriginal)


        image = cv2.resize(frame, (WIDTH, HEIGHT), interpolation=cv2.INTER_AREA)


        # Convert BGR to YUV420P

        image = cv2.cvtColor(image, cv2.COLOR_BGR2YUV_I420)

        pipe_out.stdin.write(image.tostring())

        pipe_out.stdin.flush()


        key = cv2.waitKey(1) & 0xFF


        # if the `q` key was pressed, break from the loop

        if key == ord("q"):

            break


        count += 1
```

```
            time.sleep(0.03)

        end_time = time.time()

        seconds = end_time - start_time

        print("FPS:", int(1 / seconds))


except Exception as err:  # This is bad! replace it with proper handling

    print("Error #####: ", err)

    print("Error #####: ", traceback.format_exc())


videoGetter.stop()

# do a bit of cleanup

cv2.destroyAllWindows()

pipe_out.stdin.close()
```

==

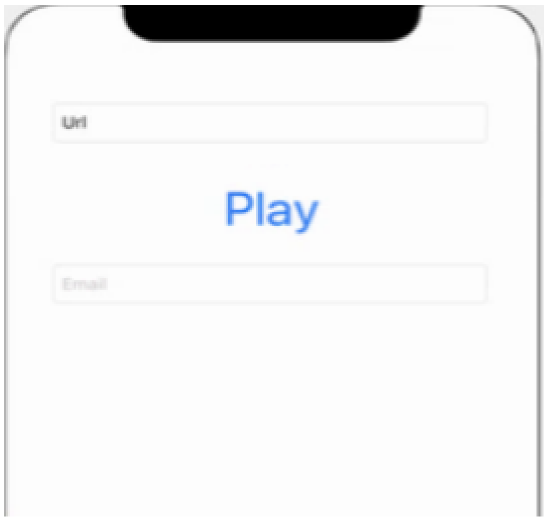
3.4. Execute hls_videostreaming_rtsp_opencv.py for testing

```
$python3 hls_videostreaming_rtsp_opencv.py
```

4. Create the iOS App side program in Xcode on macOS

4.1. Create a new project as Storyboard, not Swift Interface

4.2. Create a button named "Play" and a TextField



4.3. Modify ViewController.swift

```
==============================================

import UIKit

import AVFoundation

import AVKit


class ViewController: UIViewController {


    @IBOutlet weak var urlField: UITextField!

    override func viewDidLoad() {

        super.viewDidLoad()

        // Do any additional setup after loading the view.

    }
```

```
  @IBAction func Play(_ sender: UIButton) {

        let theUrl = urlField.text!

        print(theUrl)

 /*      guard let url = URL(string:
"https://devstreaming-cdn.apple.com/videos/streaming/examples/bipbop_adv_exa
mple_hevc/master.m3u8") else {
            return
        }

 */
        guard let url = URL(string: theUrl) else {
            return
        }
        // Create an AVPlayer, passing it the HTTP Live Streaming URL.
        let player = AVPlayer(url: url)

        // Create a new AVPlayerViewController and pass it a reference to the player.
        let controller = AVPlayerViewController()
        controller.player = player
```

```
            // Modally present the player and call the player's play() method when
complete.

        present(controller, animated: true) {

            player.play()

        }


    }


}
```

==

4.4. Modify Info.plist to disable TransportSecurity

Add the following lines

==

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

<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">

<plist version="1.0">

<dict>

        <key>NSAppTransportSecurity</key>

        <dict>

                <key>NSAllowsArbitraryLoads</key>
```

Add these codes

```
			<true/>
		</dict>
		<key>UIApplicationSceneManifest</key>
		<dict>
			<key>UIApplicationSupportsMultipleScenes</key>
			<false/>
			<key>UISceneConfigurations</key>
			<dict>
				<key>UIWindowSceneSessionRoleApplication</key>
				<array>
					<dict>
						<key>UISceneConfigurationName</key>
						<string>Default Configuration</string>
						<key>UISceneDelegateClassName</key>
						<string>$
	(PRODUCT_MODULE_NAME).SceneDelegate</string>
						<key>UISceneStoryboardFile</key>
						<string>Main</string>
					</dict>
				</array>
			</dict>
		</dict>
	</dict>
```

```
</plist>
```

===

5. SSH and Port Forwarding

5.1. Create an account on ngrok;

https://ngrok.com/

5.2. On a browser, access the setup page;

https://dashboard.ngrok.com/get-started/setup

5.3. Download ngrok client software (zip file)

5.4. Decompress the zip file

5.5. Open a terminal where the zip file was decompressed

5.6. Run ngrok command with the authtoken, which is displayed in the setup page

```
$./ngrok authtoken 1ykI0UINF1vKwf3664nnnnnnnnnnnnnnnnnnnnnn
```

Note: Keep this terminal for 6.3. Start SSH tunneling

6. Execute programs

6.1. Run the AIV-100 web server on Ubuntu

6.2. Execute ipcam_videostreaming.py on Ubuntu

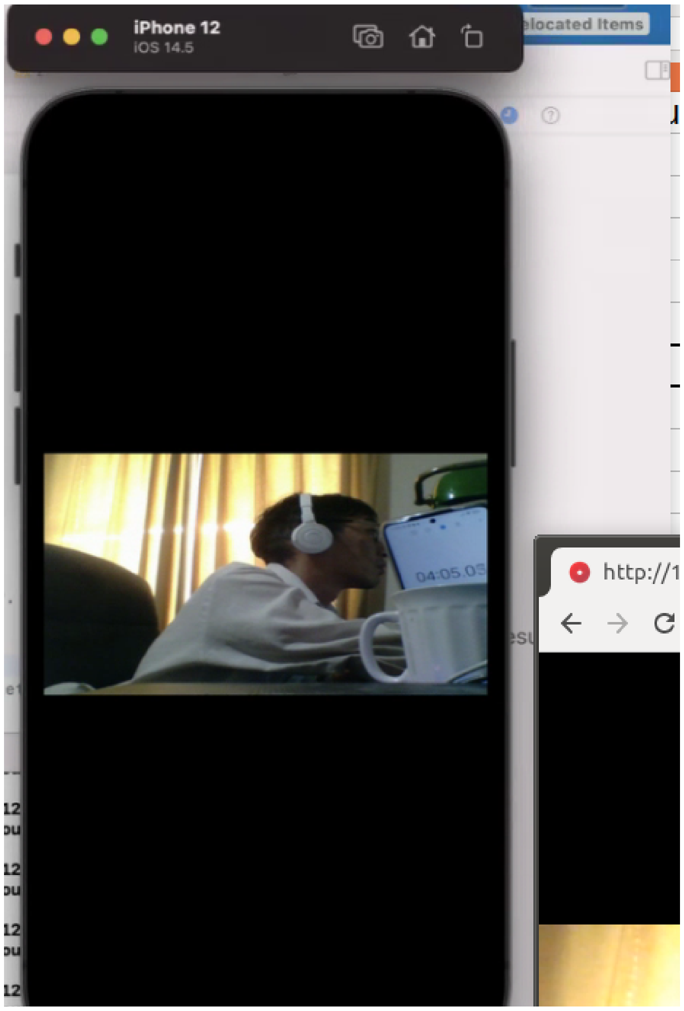
```
$python3 ipcam_videostreaming.py
```

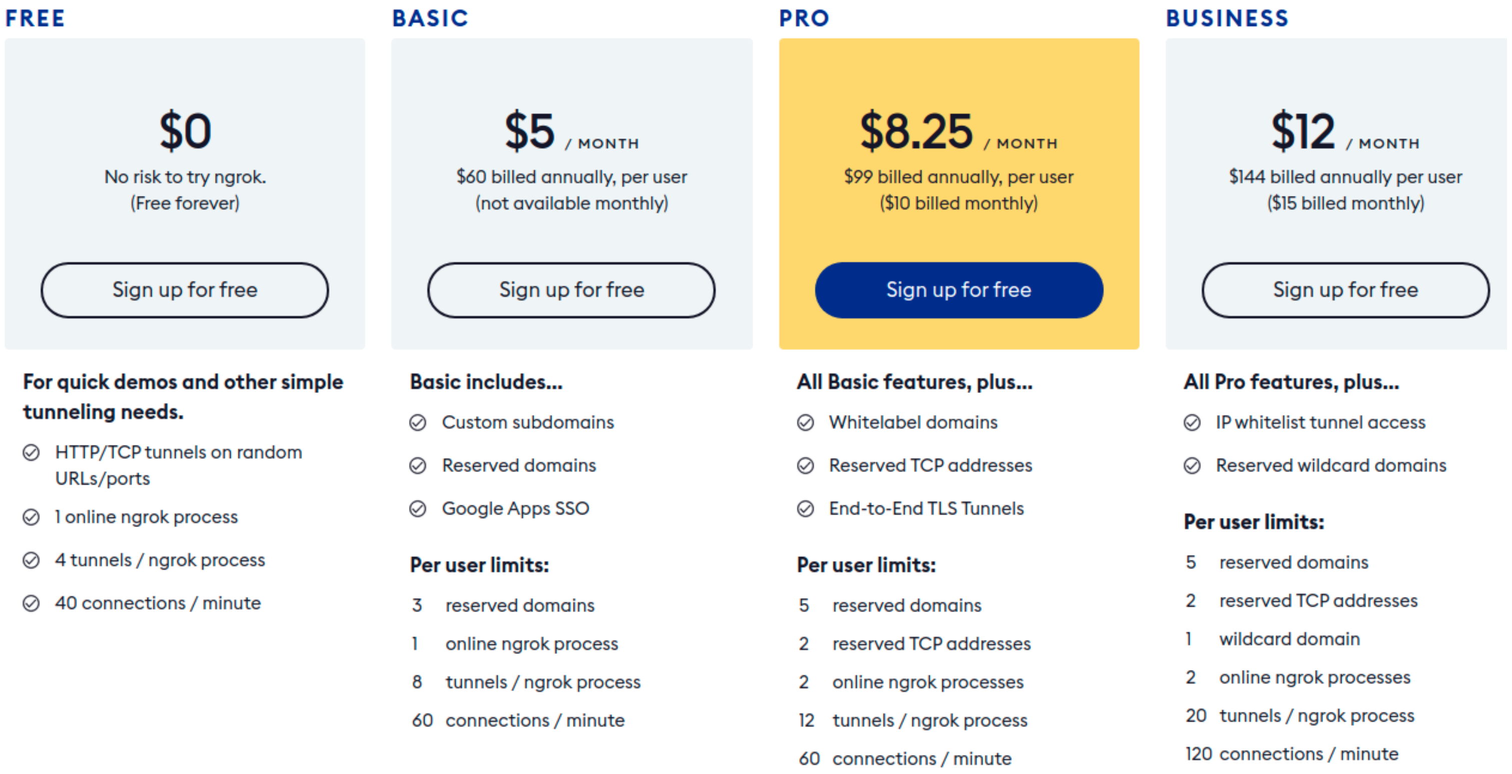
6.3. Start SSH tunneling in the terminal

```
$./ngrok http 8090
```

6.3. Execute iOS App on Mac

The result is;





FREE

$0

No risk to try ngrok.
(Free forever)

Sign up for free

For quick demos and other simple tunneling needs.

- HTTP/TCP tunnels on random URLs/ports
- 1 online ngrok process
- 4 tunnels / ngrok process
- 40 connections / minute

BASIC

$5 / MONTH

$60 billed annually, per user
(not available monthly)

Sign up for free

Basic includes...

- Custom subdomains
- Reserved domains
- Google Apps SSO

Per user limits:

3 reserved domains
1 online ngrok process
8 tunnels / ngrok process
60 connections / minute

PRO

$8.25 / MONTH

$99 billed annually, per user
($10 billed monthly)

Sign up for free

All Basic features, plus...

- Whitelabel domains
- Reserved TCP addresses
- End-to-End TLS Tunnels

Per user limits:

5 reserved domains
2 reserved TCP addresses
2 online ngrok processes
12 tunnels / ngrok process
60 connections / minute

BUSINESS

$12 / MONTH

$144 billed annually per user
($15 billed monthly)

Sign up for free

All Pro features, plus...

- IP whitelist tunnel access
- Reserved wildcard domains

Per user limits:

5 reserved domains
2 reserved TCP addresses
1 wildcard domain
2 online ngrok processes
20 tunnels / ngrok process
120 connections / minute

(END)