MediaGrator

STREAMING MEDIA MADE EASY

Contents

[Web RTC Architecture 3](#_Toc485322947)

[Server 4](#_Toc485322948)

[Node JS Architecture 6](#_Toc485322949)

[config.js 6](#_Toc485322950)

[Package 8](#_Toc485322951)

[Deploy Site to Heroku 11](#_Toc485322952)

[Prerequisites 11](#_Toc485322953)

[Overview 11](#_Toc485322954)

[Declare app dependencies 11](#_Toc485322955)

[Specify the version of node 11](#_Toc485322956)

[Specifying a start script 11](#_Toc485322957)

[Build your app and run it locally 11](#_Toc485322958)

[Deploy our application to Heroku 12](#_Toc485322959)

[Redis To Go installation 13](#_Toc485322960)

[Provisioning the add-on 13](#_Toc485322961)

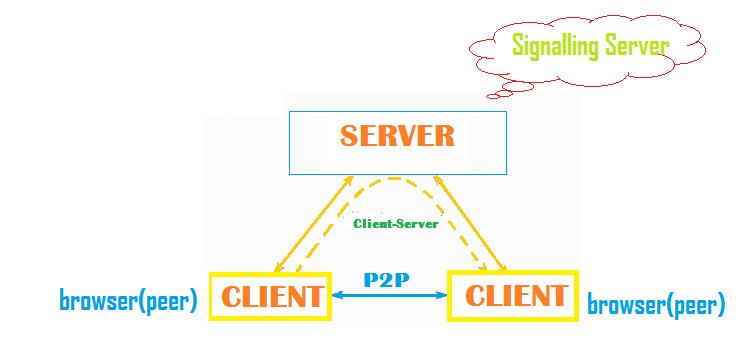
[Redis with Node.js 13](#_Toc485322962)

# Web RTC Architecture

WebRTC (the Web Real Time The Communications) - a standard that allows you to stream audio and video from the browser and the browser in real time without having to install plug-ins or other extensions. Standard allows you to turn the browser into a video conferencing endpoint terminal, simply open the web page in order to start communicating.

WebRTC will have an impact on the future of Unified Communications. WebRTC was started by Google with the goal to build a standards-based real-time media engine implemented in all of the available browsers. A browser with WebRTC a web services application can direct the browser to establish a real-time voice or video RTP connection to another WebRTC device or to a WebRTC media server. WebRTC APIs and the media engine define the communications path.

In the below, without Integrating RTC technology with existing content and services has been difficult and time consuming, particularly on the web and when it comes to cost, it requiring expensive audio and video technologies to be licensed or developed in house



WebRTC enables browser-to-browser audio and video conferencing. The user can initiate a call by clicking on an icon representing the other endpoint. What is significant and great is that a separate conferencing client isn’t needed. And, the only technology needed by the partner is a standard, up-to-date browser. What all technologies are working under hood or behind all this? The WebRTC engine within the browser uses HTML5 and Java scripting to develop fairly simple routines to capture, control, and send audio and video between two browsers.

How WebRTC is going to help us in exchange of real-time media between two browsers. Finally, workflow for this type of communication is like at the media source, input devices are opened for capture. Media from the input devices is encoded and transmitted across the network. At the media destination, the packets are decoded and formed into a media stream. The media stream is sent to output devices. Luckily, the browser hides most of this complexity behind three primary APIs:

* acquisition of audio and video streams
* communication of audio and video data
* communication of arbitrary application data

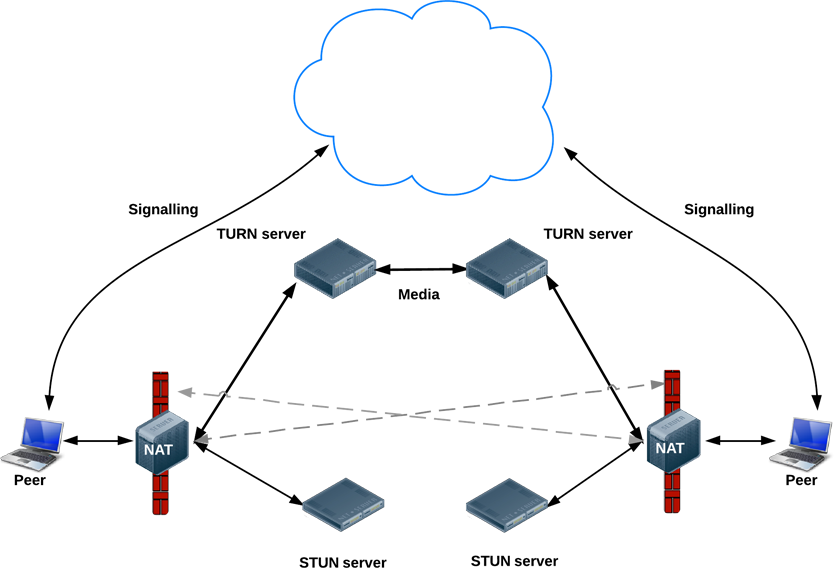
All it takes is some lines of JavaScript code, and any web application can enable a rich teleconferencing experience with peer-to-peer data transfers. That’s the promise and the power of WebRTC!

## Server

A true functional WebRTC application needs about 2-3 server’s setup to get the complete system up.

Don't be afraid, these are just servers which are easy to setup and will give you complete control on video/audio chat of the participants.

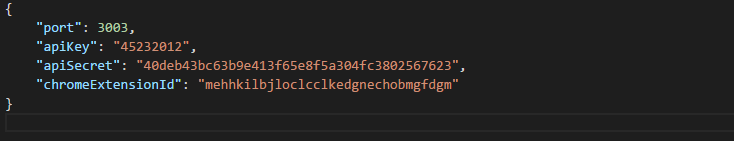
* **SIGNALLING SERVER-** The signaling server is your own implementation for managing and communicating between users. It also helps exchange information to get the live video feed started. There are several methods for setting up the back-end. I personally prefer to use Node.js server with Socket.io.
* **STUN SERVER-** The second and third serves are the STUN and TURN servers. These servers help users connect to each other to handle the actual live video and data channel messages. Where they are different is STUN helps users connect directly to each other so they can communicate.
* **TURN SERVER -** When the STUN server can’t make the connection due to firewalls or other network issues then that’s when the TURN server is used. TURN acts as the middleman to connect the users. Some TURN servers can also act as STUN servers. If this is the case then a separate STUN server is not required.



# Node JS Architecture

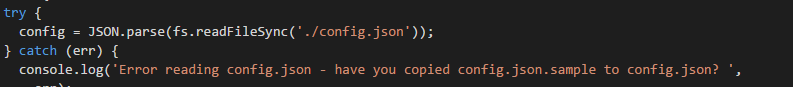
## config.js

The config.js file contains all necessary parameters for application to run - port, open tok Api key, api Secret key for authentication.

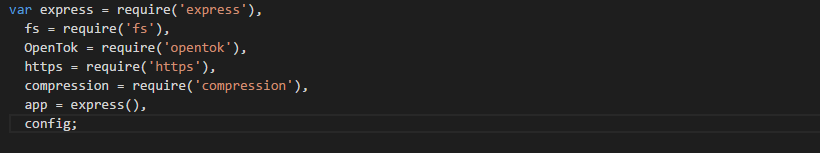


app.js

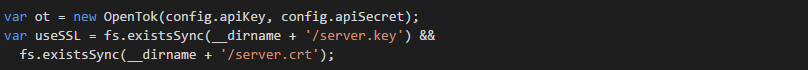
We require configuration data whenever you need it using the get function of our module and passing the environment variable if the environment isn’t Heroku.



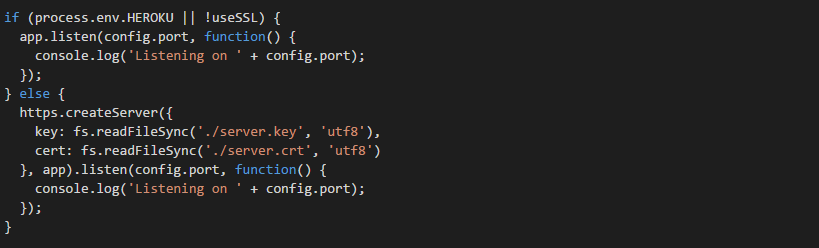
We can require any js file, you just need to declare what you want to expose



Get the open tok api key and api secret key from config file.

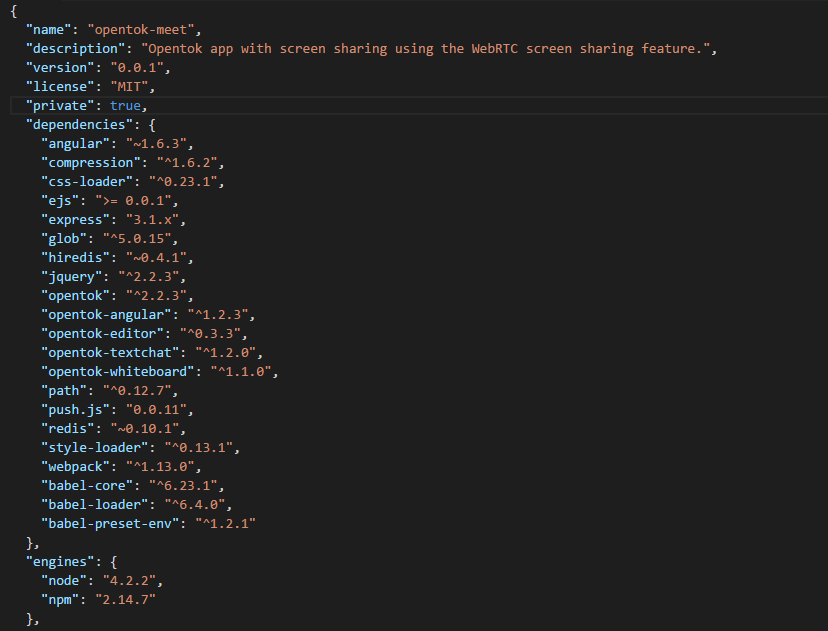


Create server and listening on the config file’s specified port

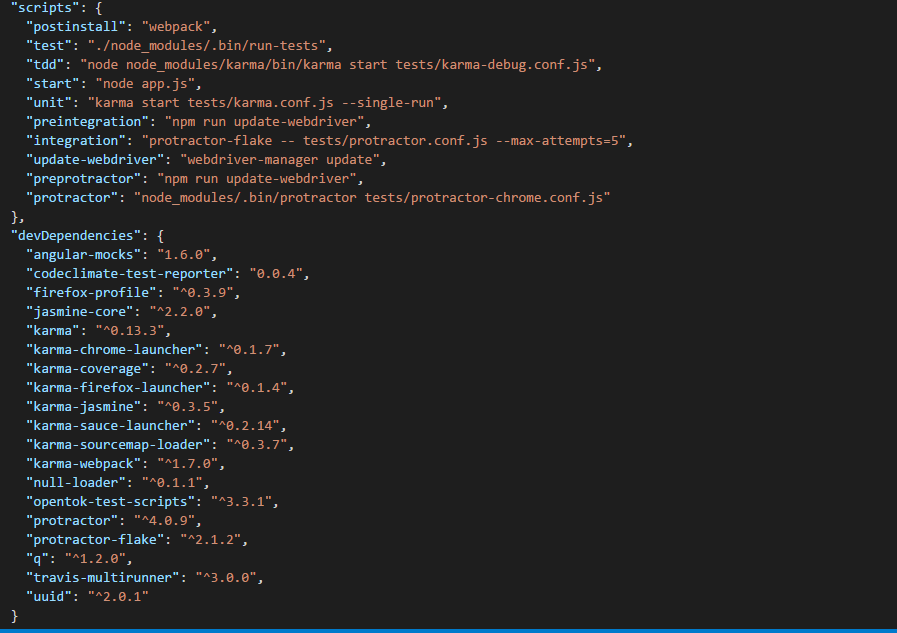


## Package

This document is all you need to know about what's required in your package.json file. It must be actual JSON, not just a JavaScript object literal.



The most important things in your package.json are the name and version fields. The name and version together form an identifier that is assumed to be completely unique. Changes to the package should come along with changes to the version.



# Deploy Site to Heroku

## Prerequisites

In order to deploy site, we have a couple of things:

* Have git installed.
* Node.js and npm installed.
* Heroku Account – sign up from <https://signup.heroku.com/>
* Download the Heroku CLI from <https://cli.heroku.com/>
* Run heroku login in terminal or command prompt and fill in our Heroku credentials

## [Overview](https://devcenter.heroku.com/articles/deploying-nodejs#overview)

Heroku Node.js support will be applied because the application has a package.json file in the root directory.

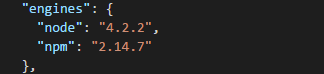
## Declare app dependencies

The package.json file defines the dependencies that should be installed with our application. This file located in root directory of our app.

## Specify the version of node

The version of Node.js that will be used to run your application on Heroku, should also be defined in your package.json file

our package.json file will look something like this:



## Specifying a start script

To determine how to start our app, Heroku first looks for a Procfile. If no Procfile exists for a Node.js app, we will attempt to start a default web process via the start script in our package.json.

The command in a web process type must bind to the port number specified in the PORT environment variable.

## Build your app and run it locally

Run the npm install command in our local app directory to install the dependencies that you declared in our package.json file



Start our app locally using the heroku local command, which is installed as part of the Heroku CLI.



## Deploy our application to Heroku

* git init

Empty Git repository will be initialized in .git/ folder.

Then run:

* git add .

This command allows Git to track your files changes.

Now commit your files to the initialized Git repo:

* git commit -m "Simple server functionality added"

We'll create our first Heroku application now:

* heroku create

Heroku will generate a random name for your application. In my case it's enigmatic-citadel-9298. Don't worry. You can change it later.

Now we can deploy our project. Every Heroku app starts with no branches and no code. So, the first time we deploy our project, we need to specify a remote branch to push to:

* git push heroku master

The application is now deployed. Ensure that at least one instance of the app is running:

* heroku ps:scale web=1

And now, before we open it, it's time to choose a proper name for our first creation. We called it mediagrator:

* heroku apps:rename mediagrator

Everything is done. You can try it now:

* heroku open

This command will open your Heroku project in your web browser. In this particular case, server address is https:// mediagrator.herokuapp.com/. Now we can share our first web application with any person we want.

## Redis To Go installation

RedisToGo is an add-on for providing functionality for Redis with graphs, backups, persistence and fine-tuned Redis. They currently support over 53,000 Redis instances.

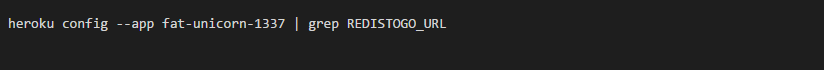
RedisToGo is accessible via an API and has supported with Node.js.

### Provisioning the add-on

RedisToGo can be attached to a Heroku application via the CLI:



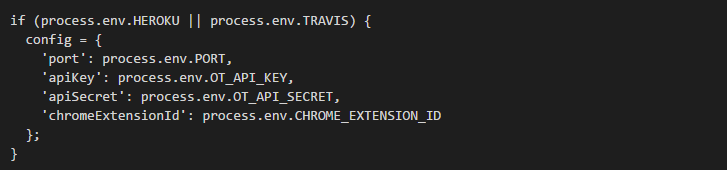
Once RedisToGo has been added a ADDON\_CONFIG\_NAME setting will be available in the app configuration. This can be confirmed using the heroku config:get command.



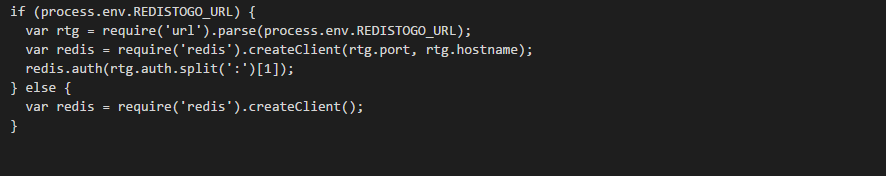
After installing RedisToGo the application should be configured to fully integrate with the add-on.

## Redis with Node.js

When node\_redis connects to a Redis instance we need it to authenticate to Redis To Go in production.



Everything should still work fine in development, but we still need to implement the Redis To Go connection. To do this we need to extract the port, hostname, and authentication string from REDISTOGO\_URL using Node’s built-in url lib:]



For some reason Node’s url lib won’t split up the auth section’s username and password, so that’s what is going on in the rtg.client.auth.split(":")[1] that is passed to the auth command.