# Deploying Janus

When you're going to deploy Janus (e.g., to try the demos we made available out-of-the-box), there's one thing that is important to point out: while Janus does indeed provide an HTTP RESTful interface (documented in RESTful, WebSockets, RabbitMQ, MQTT, Nanomsg and UnixSockets API (rest.html)), it does **NOT** also act as a webserver for static files. This means you'll need a different webserver to host static files, including HTML/PHP/JSP/etc. pages, JavaScript files, images and whatever is part of your web application.

That said, deploying Janus is, in principle, quite simple: just start Janus on a machine, put the HTML and JavaScript that will make use of it on a webserver somewhere, make sure the JavaScript code is configured with the right address for the server and you're done!

Let's assume, for the sake of simplicity, that your webserver is serving files on port `80`. By default, Janus binds on the `8088` port for HTTP. So, if Janus and the webserver hosting the are co-located, all you need to get your application working is configure the web application to point to the right address for the server. In the demos provided with these packages, this is done by means of the `server` variable:

```
var server = "http://" + window.location.hostname + ":8088/janus";
```

which basically tells the JavaScript application that the Janus API can be contacted at the same host as the website but at a different port (8088) and path (/janus). In case you configured the server differently, e.g., 7000 as the port for HTTP and /my/custom/path as the API endpoint, the `server` variable could be built this way:

```
var server = "http://" + window.location.hostname + ":7000/my/custom/path";
```

In case the webserver and Janus are **NOT** colocated, instead, just replace the `window.location.hostname` part with the right address of the server, e.g.:

```
var server = "http://www.example.com:8088/janus";
```

It's important to point out, though, that this more "static" approach only works if the webserver is serving files via HTTP. As soon as you start involving **HTTPS**, things start to get more complicated: in fact, for security reasons you cannot contact an HTTP backend if the page is made available via HTTPS. This means that if you're interested in serving your web application via HTTPS, you'll need to enable the HTTPS embedded webserver in Janus as well, and configure the JavaScript code to refer to that itself, e.g.:

```
var server = "https://" + window.location.hostname + ":8089/janus";
```

assuming `8089` is the port you configured Janus to use for HTTPS. To make this more "dynamic", e.g., allow both HTTP and HTTPS instead of just sticking to one, you might make use of something like this:

```
var server = null;
if(window.location.protocol === 'http:')
	server = "http://" + window.location.hostname + ":8088/janus";
else
	server = "https://" + window.location.hostname + ":8089/janus";
```

that is evaulate the right address to use at runtime.

Anyway, there's a much easier way to address these scenarios, which is explained in the next section.

# Deploying Janus behind a web frontend

To avoid most of the issues explained above, an easy approach can be deploying Janus behind a frontend (e.g., Apache HTTPD, nginx, lighttpd or others) that would act as a reverse proxy for incoming requests. This would allow you to make the Janus API available as a relative path of your web application, rather than a service reachable at a different port and/or domain.

Configuring the web application, as a consequence, would be even easier, as all you'd need to do would be to provide a relative path for the API, e.g.:

```
var server = "/janus";
```

which would automatically work whether the page is served via HTTP or HTTPS. In fact, all the HTTPS requests would be terminated at the webserver, which would then always send simple HTTP messages to the server itself.

An easy way to do so in Apache HTTPD is by means of the following directives:

```
ProxyRequests Off
ProxyVia Off
ProxyPass /janus http://127.0.0.1:8088/janus retry=0
ProxyPassReverse /janus http://127.0.0.1:8088/janus
```

Different versions of HTTPD or different webservers may require a different syntax, but the principle is usually always the same: you instruct the webserver to act as a proxy for a local endpoint, in this case a Janus instance colocated at the webserver and configured with the default settings.

A way to do the same with nginx, as explained by some Janus users here (https://groups.google.com/forum/#!topic/meetecho-janus/dIv-4s0HOdw), is the following directive:

```
location /janus {
        proxy_pass http://127.0.0.1:8088/janus;
}
```

# A quick and easy web server

While opening WebRTC-powered web applications by just opening the application HTML files from file system works with some browsers, it doesn't in others. Specifically, this works in Firefox but not in Chrome (see issue #291 (https://github.com/meetecho/janus-gateway/issues/291)). Anyway, considering that you will eventually want other people besides you to use your Janus services, this means that to test and use Janus you'll want/need to host your applications on a webserver.

If you're not interested in configuring a full-fledged webserver, but are only interested in a quick and easy way to test the demos, you can make use of the embedded webservers some frameworks like PHP and Python provide. To start a webserver for the demos, for instance, just open a terminal in the `html` folder of the project, and type:

```
php -S 0.0.0.0:8000
```

or:

```
python -m SimpleHTTPServer 8000
```

This will setup a webserver on port `8000` for you to use, meaning you'll just need to have your browser open a local connection to that port to try the demos:

```
http://yourlocaliphere:8000
```

You can do the same on a different port to also access the HTML version of the Doxygen generated documentation, starting the embedded webservers from the `docs/html` folder instead:

```
php -S 0.0.0.0:9000
```

or:

```
python -m SimpleHTTPServer 9000
```

# Using Janus with WebSockets

Configuring the use of WebSockets rather than the REST API in the JavaScript library is quite trivial, as it's a matter of passing a `ws://` address instead of an `http://` (`http://`) one to the constructor. That said, most of the same considerations provided for the REST API apply here as well, e.g., to just use `window.location.hostname` if the webserver and Janus are colocated:

```
var server = "ws://" + window.location.hostname + ":8188/";
```

to specify the port if you change it:

```
var server = "ws://" + window.location.hostname + ":7000/";
```

and/or the right address of the server in case the webserver and Janus are **NOT** colocated:

```
var server = "ws://www.example.com:8188/";
```

Notice how the path ( `/janus` by default for HTTP) is not provided for WebSockets, as it is ignored by the server.

The considerations for deploying Janus behind a proxy/webserver, though, differ if you use WebSockets, as most webservers don't provide an easy way to proxy WebSocket requests, and usually require custom modifications for the purpose. Recent versions of HTTPD (>= 2.4.5), with the right module (proxy_wstunnel), do allow you to also proxy WebSockets requests the same way you do with HTTP, which can be useful to do the same WSS-to-WS proxying in a frontend. Here's a sample configuration:

```
<IfModule mod_proxy_wstunnel.c>
        ProxyPass /janus-ws ws://127.0.0.1:8188 retry=0
        ProxyPassReverse /janus-ws ws://127.0.0.1:8188
</IfModule>
```

that will allow you to expose a `wss://myserver/janus-ws` or `ws://myserver/janus-ws` address, and have all communication forwarded to and from Janus at `ws://127.0.0.1:8188` .

Similar configurations are probably available for other systems as well, so in case this is something you're interested in, we recommend you follow the best practices related to that made available by the web server developers.

# Using fallback addresses

As anticipated in the JavaScript API (JS.html) section, you can also pass an array of servers to the Janus library initialization. This allows you, for instance, to pass a link to both the WebSockets and REST interfaces, and have the library try them both to see which one is reachable, e.g.:

```
var ws_server = "ws://" + window.location.hostname + ":8188/";
var http_server = "http://" + window.location.hostname + ":8088/janus";
var servers = [ws_server, http_server];
```

which is especially useful if you're not sure whether or not WebSockets will work in some specific networks. Please notice that, for the individual servers listed in the array, the same considerations given above (e.g., in terms of relative vs. absolute linking) still apply.

Such an approach can also be used when you've deployed several different instances of Janus, and you want the library to try some and fallback to others if any of them is not reachable for any reason.

---