Version: 4.x

On this page

Server Initialization

Once you have installed the Socket.IO server library, you can now init the server. The complete list of options can be found here.

> **TIP**
>
> For TypeScript users, it is possible to provide type hints for the events. Please check this.

Initialization

Standalone

CommonJS ES modules TypeScript

```
const { Server } = require("socket.io");

const io = new Server({ /* options */ });

io.on("connection", (socket) => {
  // ...
});

io.listen(3000);
```

You can also pass the port as the first argument:

CommonJS ES modules TypeScript

```
const { Server } = require("socket.io");
```

```
const io = new Server(3000, { /* options */ });

io.on("connection", (socket) => {
  // ...
});
```

This implicitly starts a Node.js HTTP server, which can be accessed through `io.httpServer`.

With an HTTP server

CommonJS | ES modules | TypeScript

```
const { createServer } = require("http");
const { Server } = require("socket.io");

const httpServer = createServer();
const io = new Server(httpServer, { /* options */ });

io.on("connection", (socket) => {
  // ...
});

httpServer.listen(3000);
```

With an HTTPS server

CommonJS | ES modules | TypeScript

```
const { readFileSync } = require("fs");
const { createServer } = require("https");
const { Server } = require("socket.io");

const httpsServer = createServer({
  key: readFileSync("/path/to/my/key.pem"),
  cert: readFileSync("/path/to/my/cert.pem")
});

const io = new Server(httpsServer, { /* options */ });

io.on("connection", (socket) => {
  // ...
```

```
});

httpsServer.listen(3000);
```

See also: Node.js documentation

With client-certificate authentication:

Server

```
import { readFileSync } from "fs";
import { createServer } from "https";
import { Server } from "socket.io";

const httpsServer = createServer({
  key: readFileSync("/path/to/server-key.pem"),
  cert: readFileSync("/path/to/server-cert.pem"),
  requestCert: true,
  ca: [
    readFileSync("/path/to/client-cert.pem")
  ]
});

const io = new Server(httpsServer, { /* options */ });

io.engine.on("connection", (rawSocket) => {
  // if you need the certificate details (it is no longer available once the
handshake is completed)
  rawSocket.peerCertificate = rawSocket.request.client.getPeerCertificate();
});

io.on("connection", (socket) => {
  console.log(socket.conn.peerCertificate);
  // ...
});

httpsServer.listen(3000);
```

Client

```
import { readFileSync } from "fs";
import { io } from "socket.io-client";

const socket = io("https://example.com", {
  key: readFileSync("/path/to/client-key.pem"),
```

```
  cert: readFileSync("/path/to/client-cert.pem"),
  ca: [
    readFileSync("/path/to/server-cert.pem")
  ]
});
```

With an HTTP/2 server

CommonJS ES modules TypeScript

```
const { readFileSync } = require("fs");
const { createSecureServer } = require("http2");
const { Server } = require("socket.io");

const httpServer = createSecureServer({
  allowHTTP1: true,
  key: readFileSync("/path/to/my/key.pem"),
  cert: readFileSync("/path/to/my/cert.pem")
});

const io = new Server(httpServer, { /* options */ });

io.on("connection", (socket) => {
  // ...
});

httpServer.listen(3000);
```

See also: Node.js documentation

With Express

CommonJS ES modules TypeScript

```
const express = require("express");
const { createServer } = require("http");
const { Server } = require("socket.io");

const app = express();
const httpServer = createServer(app);
```

```
const io = new Server(httpServer, { /* options */ });

io.on("connection", (socket) => {
  // ...
});

httpServer.listen(3000);
```

> **CAUTION**
>
> Using `app.listen(3000)` will not work here, as it creates a new HTTP server.

More information here.

With Koa

CommonJS | ES modules | TypeScript

```
const Koa = require("koa");
const { createServer } = require("http");
const { Server } = require("socket.io");

const app = new Koa();
const httpServer = createServer(app.callback());
const io = new Server(httpServer, { /* options */ });

io.on("connection", (socket) => {
  // ...
});

httpServer.listen(3000);
```

More information here.

With Nest

See the documentation here.

> **CAUTION**

> NestJS v7 and below relies on Socket.IO v2, while NestJS v8 relies on Socket.IO v4. Please use a compatible client.

With Fastify

You need to register the `fastify-socket.io` plugin:

CommonJS | ES modules | TypeScript

```
const fastify = require("fastify");
const fastifyIO = require("fastify-socket.io");

const server = fastify();
server.register(fastifyIO);

server.get("/", (req, reply) => {
  server.io.emit("hello");
});

server.ready().then(() => {
  // we need to wait for the server to be ready, else `server.io` is undefined
  server.io.on("connection", (socket) => {
    // ...
  });
});

server.listen(3000);
```

With µWebSockets.js

```
import { App } from "uWebSockets.js";
import { Server } from "socket.io";

const app = new App();
const io = new Server();

io.attachApp(app);

io.on("connection", (socket) => {
  // ...
});
```

```
app.listen(3000, (token) => {
  if (!token) {
    console.warn("port already in use");
  }
});
```

Reference: https://github.com/uNetworking/uWebSockets.js

Options

The complete list of available options can be found here.

Edit this page

*Last updated on **12/19/2022***