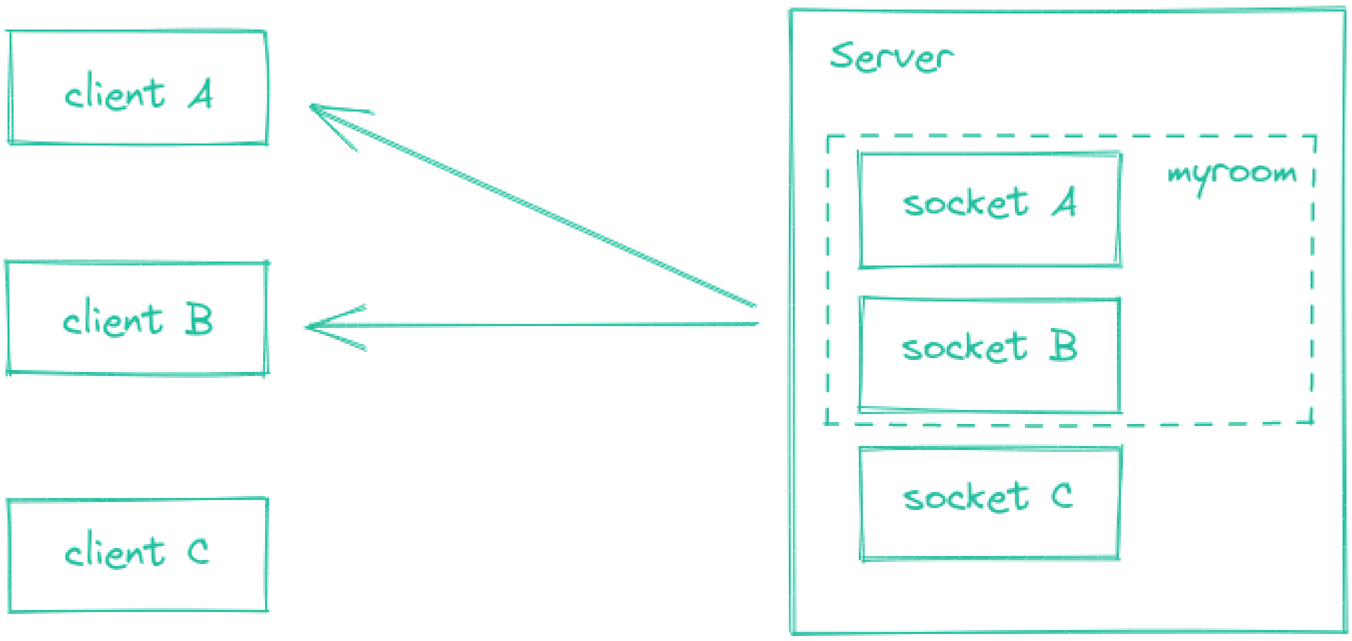
Version: 4.x

Rooms

A *room* is an arbitrary channel that sockets can `join` and `leave`. It can be used to broadcast events to a subset of clients:



> **INFO**
>
> Please note that rooms are a **server-only** concept (i.e. the client does not have access to the list of rooms it has joined).

Joining and leaving

You can call `join` to subscribe the socket to a given channel:

```
io.on("connection", (socket) => {
  socket.join("some room");
});
```

And then simply use `to` or `in` (they are the same) when broadcasting or emitting:

```
io.to("some room").emit("some event");
```

Or exclude a room:

```
io.except("some room").emit("some event");
```

You can also emit to several rooms at the same time:

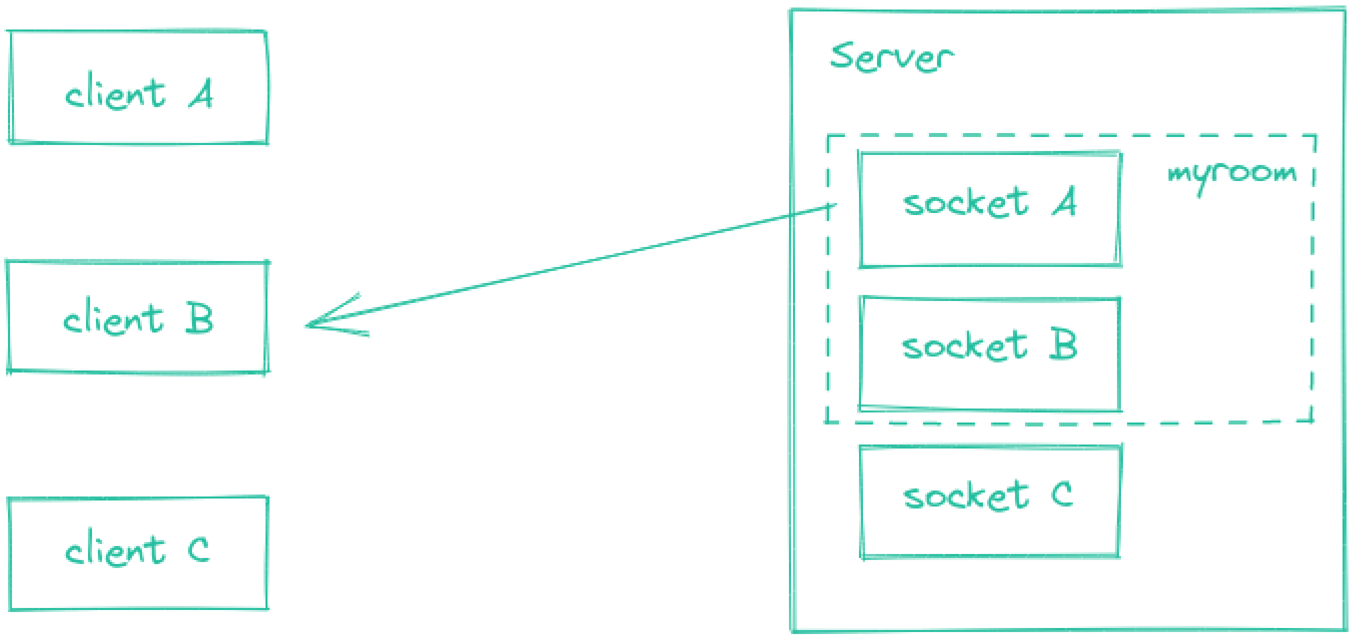
```
io.to("room1").to("room2").to("room3").emit("some event");
```

In that case, a union is performed: every socket that is at least in one of the rooms will get the event **once** (even if the socket is in two or more rooms).

You can also broadcast to a room from a given socket:

```
io.on("connection", (socket) => {
  socket.to("some room").emit("some event");
});
```

In that case, every socket in the room **excluding** the sender will get the event.



To leave a channel you call `leave` in the same fashion as `join`.

Sample use cases

- broadcast data to each device / tab of a given user

```
function computeUserIdFromHeaders(headers) {
  // to be implemented
}

io.on("connection", async (socket) => {
  const userId = await computeUserIdFromHeaders(socket.handshake.headers);

  socket.join(userId);

  // and then later
  io.to(userId).emit("hi");
});
```

- send notifications about a given entity

```
io.on("connection", async (socket) => {
  const projects = await fetchProjects(socket);

  projects.forEach(project => socket.join("project:" + project.id));

  // and then later
  io.to("project:4321").emit("project updated");
});
```

Disconnection

Upon disconnection, sockets `leave` all the channels they were part of automatically, and no special teardown is needed on your part.

You can fetch the rooms the Socket was in by listening to the `disconnecting` event:

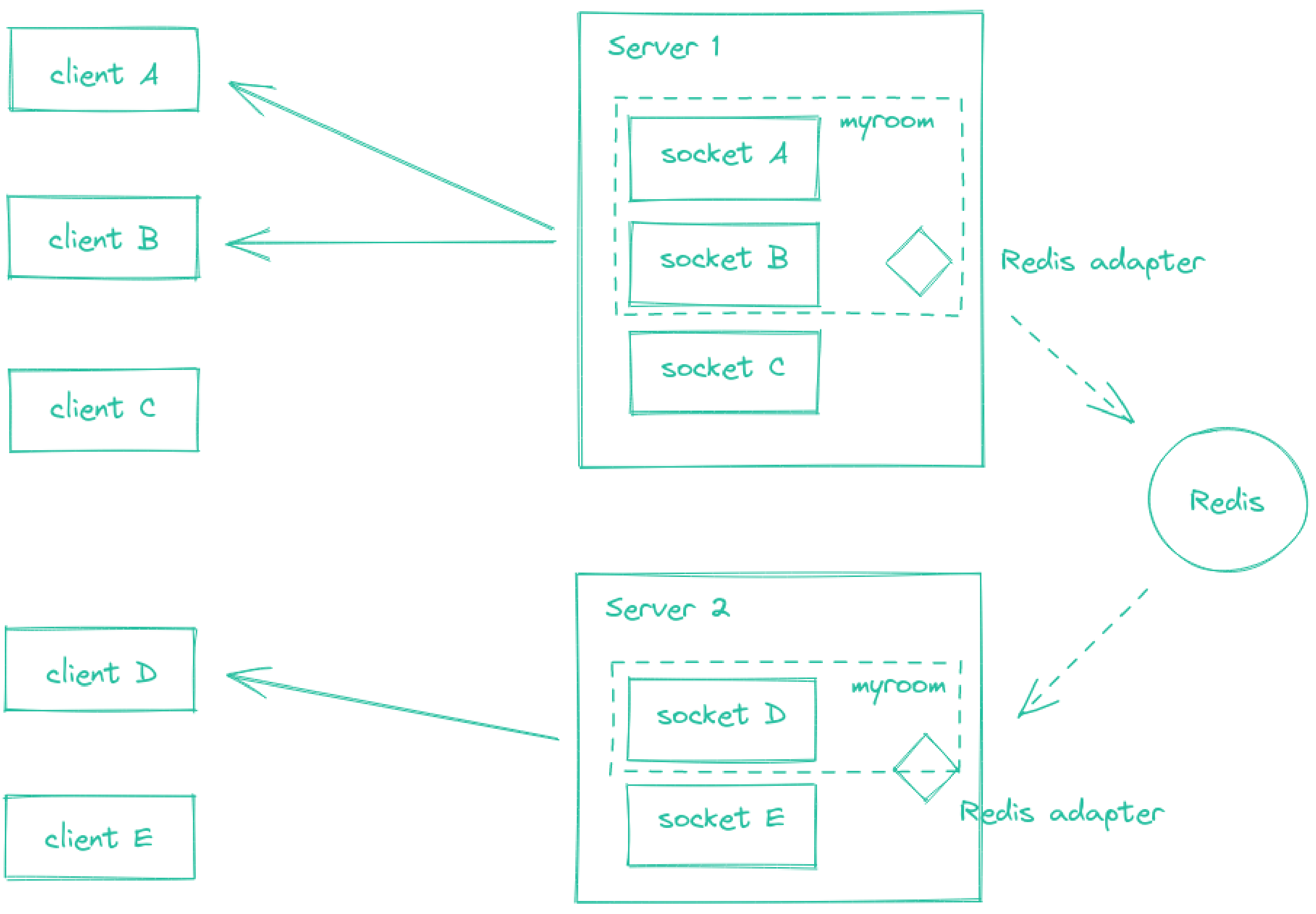
```
io.on("connection", socket => {
  socket.on("disconnecting", () => {
    console.log(socket.rooms); // the Set contains at least the socket ID
  });

  socket.on("disconnect", () => {
    // socket.rooms.size === 0
  });
});
```

With multiple Socket.IO servers

Like global broadcasting, broadcasting to rooms also works with multiple Socket.IO servers.

You just need to replace the default Adapter by the Redis Adapter. More information about it here.



Implementation details

The "room" feature is implemented by what we call an Adapter. This Adapter is a server-side component which is responsible for:

- storing the relationships between the Socket instances and the rooms
- broadcasting events to all (or a subset of) clients

You can find the code of the default in-memory adapter here.

Basically, it consists in two ES6 Maps:

- `sids`: `Map<SocketId, Set<Room>>`
- `rooms`: `Map<Room, Set<SocketId>>`

Calling `socket.join("the-room")` will result in:

- in the `sids` Map, adding "the-room" to the Set identified by the socket ID
- in the `rooms` Map, adding the socket ID in the Set identified by the string "the-room"

Those two maps are then used when broadcasting:

- a broadcast to all sockets (`io.emit()`) loops through the `sids` Map, and send the packet to all sockets
- a broadcast to a given room (`io.to("room21").emit()`) loops through the Set in the `rooms` Map, and sends the packet to all matching sockets

You can access those objects with:

```
// main namespace
const rooms = io.of("/").adapter.rooms;
const sids = io.of("/").adapter.sids;

// custom namespace
const rooms = io.of("/my-namespace").adapter.rooms;
const sids = io.of("/my-namespace").adapter.sids;
```

Notes:

- those objects are not meant to be directly modified, you should always use `socket.join(...)` and `socket.leave(...)` instead.
- in a multi-server setup, the `rooms` and `sids` objects are not shared between the Socket.IO servers (a room may only "exist" on one server and not on another).

Room events

Starting with `socket.io@3.1.0`, the underlying Adapter will emit the following events:

- `create-room` (argument: room)

- `delete-room` (argument: room)
- `join-room` (argument: room, id)
- `leave-room` (argument: room, id)

Example:

```
io.of("/").adapter.on("create-room", (room) => {
  console.log(`room ${room} was created`);
});

io.of("/").adapter.on("join-room", (room, id) => {
  console.log(`socket ${id} has joined room ${room}`);
});
```

Edit this page

*Last updated on **Aug 22, 2025***