## Hybrid application

A hybrid application is one that both listens for HTTP requests, as well as makes use of connected microservices. The `INestApplication` instance can be connected with `INestMicroservice` instances through the `connectMicroservice()` method.

```
const app = await NestFactory.create(AppModule);
const microservice = app.connectMicroservice<MicroserviceOptions>({
  transport: Transport.TCP,
});

await app.startAllMicroservices();
await app.listen(3001);
```

To connect multiple microservice instances, issue the call to `connectMicroservice()` for each microservice:

```
const app = await NestFactory.create(AppModule);
// microservice #1
const microserviceTcp = app.connectMicroservice<MicroserviceOptions>({
  transport: Transport.TCP,
  options: {
    port: 3001,
  },
});
// microservice #2
const microserviceRedis = app.connectMicroservice<MicroserviceOptions>({
  transport: Transport.REDIS,
  options: {
    host: 'localhost',
    port: 6379,
  },
});

await app.startAllMicroservices();
await app.listen(3001);
```

To bind `@MessagePattern()` to only one transport strategy (for example, MQTT) in a hybrid application with multiple microservices, we can pass the second argument of type `Transport` which is an enum with all the built-in transport strategies defined.

```
@@filename()
@MessagePattern('time.us.*', Transport.NATS)
getDate(@Payload() data: number[], @Ctx() context: NatsContext) {
  console.log(`Subject: ${context.getSubject()}`); // e.g. "time.us.east"
  return new Date().toLocaleTimeString(...);
}
```

```
@MessagePattern({ cmd: 'time.us' }, Transport.TCP)
getTCPDate(@Payload() data: number[]) {
  return new Date().toLocaleTimeString(...);
}
@@switch
@Bind(Payload(), Ctx())
@MessagePattern('time.us.*', Transport.NATS)
getDate(data, context) {
  console.log(`Subject: ${context.getSubject()}`); // e.g. "time.us.east"
  return new Date().toLocaleTimeString(...);
}
@Bind(Payload(), Ctx())
@MessagePattern({ cmd: 'time.us' }, Transport.TCP)
getTCPDate(data, context) {
  return new Date().toLocaleTimeString(...);
}
```

> info **Hint** `@Payload()`, `@Ctx()`, `Transport` and `NatsContext` are imported from `@nestjs/microservices`.

#### Sharing configuration

By default a hybrid application will not inherit global pipes, interceptors, guards and filters configured for the main (HTTP-based) application. To inherit these configuration properties from the main application, set the `inheritAppConfig` property in the second argument (an optional options object) of the `connectMicroservice()` call, as follow:

```
const microservice = app.connectMicroservice<MicroserviceOptions>(
  {
    transport: Transport.TCP,
  },
  { inheritAppConfig: true },
);
```